RATE-DISTORTION OPTIMAL MESH SIMPLIFICATION FOR COMMUNICATIONS

THÈSE N° 2260 (2000)

PRÉSENTÉE AU DÉPARTEMENT DE SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDERALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Laurent BALMELLI

Ingénieur informaticien diplômé EPF
de nationalité suisse et originaire de Paradiso (TI)

acceptée sur proposition du jury:

Prof. M.Vetterli, directeur de thèse
Prof.Y.Dodge, rapporteur
Prof. M. Gross, rapporteur
Dr J.Kovacevic, rapporteur
Prof. Th. Liebling, rapporteur

Lausanne, EPFL
2002

Abstract

The thesis studies the optimization of a specific type of computer graphic representation: polygon-based, textured models. More precisely, we focus on meshes having 4-8 connectivity. We study a progressive and adaptive representation for textured 4-8 meshes suitable for transmission. Our results are valid for 4-8 meshes built from matrices of amplitudes, or given as approximations of a subdivision surface. In the latter case, the models can have arbitrary topology.

In order to clarify our goals, we first describe a transmission system for computer graphics models (Chapter 1). Then, we review approximation techniques (Chapter 2) and study the computational properties of 4-8 meshes (Chapter 3). We provide an efficient method to store and access our dataset (Chapter 4). We address the problem of 4-8 mesh simplification and give an efficient $\Theta(n \log n)$ algorithm to compute progressive and adaptive representations of 4-8 meshes using global error (Chapter 5). We study the joint optimization of mesh and texture (Chapter 6). Finally, we conclude and give future research directions (Chapter 7).

Resumé de la thèse

Cette thèse étudie l'optimisation d'un type spécifique de modèle en graphisme: les représentations basées sur polygones texturés. Plus précisement, nous nous concentrons sur les meshes ayant une connectivité 4-8. Nous étudions une representation progressive et adaptative pour les meshes de type 4-8, adéquate pour la transmission. Nos résultats sont valables pour les meshes de type 4-8 construits à partir d'une matrices d'amplitudes ou donnés comme approximations d'une surface de subdivision. Dans le dernier cas, les modèles peuvent avoir une topologie arbitraire.

Afin de clarifier nos buts, nous décrivons premièrement un système de transmission pour modèles graphiques (Chapitre 1). Ensuite, nous présentons une révision des techniques d'approximations (Chapitre 2) et nous étudions les propriétés computationelles des meshes de type 4-8 (Chapitre 3). Nous décrivons une méthode efficace pour stocker et accéder nos données (Chapitre 4). Nous adressons le problème de simplification de mesh de type 4-8 et nous donnons un algorithm en $\Theta(n \log n)$ pour calculer des représentations progressives et adaptatives en minimisant l'erreur globale (Chapitre 5). Nous étudions l'optimisation jointe des meshes et textures (Chapitre 6). Finalement, nous donnons une conclusion, ainsi que de futurs directions de recherche (Chapitre 7).

Contents

Chapter 1

Introduction

"A picture is worth a thousand words."
– CHINESE PROVERB,

"Although this may seem a paradox, all exact science is dominated by the idea of approximation."
– BERTRAND RUSSELL,

1.1 Motivation

In this chapter, we introduce the reader to the background of the thesis. We delineate a conceptual transmission system for a specific type of computer graphics representation, i.e. textured polygon-based models. We give short descriptions of the system main components. Our aim is neither to give a precise specification, nor to describe an implementation of such system. We use it as a context within which the results of this thesis can fit. Then, we give a plan for the thesis.

1.1.1 Computer graphics everywhere

Until the early 1980's, computer graphics was a small and specialized field dealing with the display of monochromatic and edgy graphic primitives on huge and heavy screens. Today, even people who do not use computers encounter computer graphics in television commercials and in cinematic special effects. Complete movies are generated using only computer graphics techniques. Thus, computer graphics are no longer a rarity. It is part of all computer user interfaces and is indispensable for visualizing two-dimensional and three-dimensional objects.

Computer graphics should not be confused with the related field of image processing. While the former concerns the *synthesis* of real and imaginary objects from their computer-based model, the latter treats the converse process: The *analysis*, or the reconstruction of models from their picture. Subareas of image processing include, for example, *computer vision*, *scene analysis* or *pattern recognition*. Today computer graphics are largely interactive. Static pictures are a good means of communicating information (following the chapter epigraph), but dynamic pictures convey a higher level of understanding - "a moving picture is worth a thousand static ones" (Foley,1988) is more appropriate for computer graphics.

1.1.2 Synthesis of images from computer graphics models

A computer graphics model defines an *object*, or more generally a *scene* of objects as a set of primitives. A choice for those primitives usually depends on the user's needs. In practice, primitives are defined in graphic languages or applications. For example, an object can be given by its mathematical definition (when available), by a set of simpler entities (like cubes, spheres, etc.) or more simply by a set of sample points, called *vertices*, and a method to interconnect them. Additionally, a model specifies properties for its components such as colors or texture specifications [26].

Assume a model defining a surface with a bivariate function. Different approaches exist to synthesize an image from the specification. *Ray tracing* determines the visibility of the surface by tracing imaginary rays from the *viewer eyes*. The image is a snapshot of the model from a *viewpoint*. For each pixel in the window on which the image is *rasterized*[1], an edge ray is fired from the center of projection into the scene. The pixel color is set to that of the object part at the closest point of intersection. This technique provides the most realistic representations, but the complexity of casting one ray per pixel and computing its trajectory is, unfortunately, computationally demanding and inapplicable for interactive graphics [38].

Consider now a surface model given by a set of vertices, possibly obtained by sampling its mathematical description. Additionally, a set of polygons defined on these vertices is specified. *Polygonal representations* are more efficient since vertices are *projected* and their polygons

[1]*Rasterization* names the process used to display an object on the screen. Today's computer screens are *raster displays* and represent any graphic primitives, curved or straight, with a matrix of squares named *pixels*.

rasterized on the window, filling arbitrary large pixel areas (both steps, the projection and the rasterization, are part of a sequence of operations called *rendering*). *Polygon rendering* is the most common synthesis technique for today's computer graphics and is supported by computer hardware. Its efficiency allows for *rendering* 20k polygon models at interactive rates, for example at more than 30 frames (i.e. images) per second. Modern graphic hardware architectures include a pipeline to render polygonal representations. More specifically, these architectures support triangular meshes, rather than a model defined with arbitrary polygons. Triangles are generally more convenient since they are defined with three coplanar vertices. The obvious requirement to render planar facets is more difficult to verify for general polygons.

1.1.3 Scalable and adaptive graphics

As hinted in the previous section, the computational effort involved in polygon rendering is a function of the number of triangles in the model, referred to as *model complexity*. Obviously, the more triangles, the more intensive the rendering process. Even though graphic hardware systems are efficient, interactive rates can only be sustained with relatively small number of triangles. Rendering highly complex models is still a challenge for current architectures.

To cope with available computational resources, researchers have proposed *multiresolution decompositions* of graphic models by investigating *simplification algorithms*. Using multiresolution techniques, models can be rendered at multiple levels of details. For each level, we obtain an approximation using a limited number of vertices and triangles. The representation can then be adapted to the computational resources of the host computer.

From another point of view, model *compression* has been studied to reduce storage requirements. Generally, compression does not aim at reducing models' complexity, but at providing a more compact description. Apart from vertices, polygonal representations provide *connectivity information*, i.e. the description of a set of triangles. Works in compression attempt to minimize connectivity information by exploiting the redundancy of connected sets of triangles [17, 85]. The vertex coordinates are subsequently encoded using, for example, predictive coding. Finally, the remaining values are compressed using well-established quantization techniques [31].

Computational efficiency of optimization algorithms, either for simplifying or compressing models, is crucial. The need to optimize highly complex models is inherent in many disciplines. *Geographical Information Systems* (GIS) deal, for example, with large datasets [50], mainly terrain data and aerial photographs. Processing, storing and browsing these datasets requires efficient algorithms and data structures.

1.1.4 Towards a transmission system for computer graphics

With increased use of the Internet, exponential growth of computer networks, and demand for multimedia communications, the need for computer graphics fitting a connected world is obvious. Applications of computer graphics in communications are rapidly appearing. For example, the upcoming standard for low-bitrate video encoding MPEG4 [83] uses polygonal meshes to improve compression and display quality. For GIS, efficiently accessing remote databases of complex models is a must. Graphic models for communications must deal with both the diversity of network characteristics and the heterogeneity of host computers with varying storage and computational resources.

The optimization methods described in Section 1.1.3 (i.e. simplification and compression) provide ways to adapt graphic models to the multiple constraints inherent to transmission systems. The field of transmitting graphic models is still in its infancy and many questions remain open. Encoding methods must simultaneously provide compactness for model representation (for example, to fit the available channel bandwidth) and also meet the rendering capacity at the receiver. Also, they must deal with the heterogeneity of model components. For example, in the case of GIS, consider jointly transmitting terrain data and related aerial photographs. Last but not least, systems must be scalable to highly complex models and therefore provide with algorithms that efficiently exploit the transmitter and receiver resources, both in computation and storage.

These exciting issues provides a ground for this thesis and motivates our efforts to delineate a conceptual transmission system for computer graphics models. Recall that our intention is not to give the full specification of such a system, but to identify its main components and the issues related to the transmission of graphic models. We only consider model with a specific connectivity and propose several optimization techniques for these models. For images (e.g. given as textures), we however assume having encoding methods. The results in this thesis give solutions to the identified issues, according to our model assumptions. In the next sections, we first define in detail the problems to address. Then, we introduce our solutions and give a plan for the thesis.

1.2 A transmission system for graphic models

1.2.1 Model assumptions

We consider textured, polygon-based representations of graphic models. These representations are mainly composed of:

- *Geometry, connectivity information.*
- *Texture data.*

The geometry is formed by a set of vertices. The connectivity defines how the vertices are connected to create a set of polygons. Both geometry and connectivity form a *mesh*, i.e. a wireframe representation. We will assume a particular connectivity for our models and will give details in later sections. Possible attributes for vertices such as colors and normal vectors[2] can also be specified.

A *texture* can be given as an image to be mapped onto the polygons[3]. Texture images are used to represent fine details and greatly increase the realism of polygonal models. Suppose, for example, that you want to create a wooden box. Wood being subtle in detail, a tremendous number of triangles would be necessary to faithfully reproduce its structure. Consider a simple cube model (Figure 1.1a) and assume you own a picture of the face of a real wooden box, then *texturing* is used to map the image on the polygons to create a "wood effect" (Figure 1.1b). Identically, a small sample of wallpaper texture can be tiled on the object surface (Figure 1.1c).

Texture data is composed of a set of images, possibly a single one, as well as side information describing how texturing is performed. For example, when multiple textures are given for the model, one has to specify the relations between the texture elements (possibly replicated) and the polygons. Current graphic hardware systems support rendering of textures at a much inferior cost than the one required to render polygons. Texturing operates mainly as a low-cost geometry replacement.

In order to describe realistic objects, our graphic representation is therefore composed of two complementary types of information. Their respective storage sizes vary between models. For example, a model can have simple geometry and high-resolution texture information (for example the wood box in Figure 1.1b). In this case, texture data is much larger that geometry information. Conversely, a model can have high geometric complexity and low-resolution texture. Both types of information are different in nature and are often optimized separately.

1.2.2 Transmitted information

In the previous section, we defined our graphic representation as composed of two types of information: geometry (along with connectivity) and texture. Consider now that these components are gathered in a single *source* to transmit. Then, an appropriate definition for the source is

- *The information to transmit in order to synthesize a view of the model at the receiver.*

The source can be either

- *Static* when its definition (geometry, connectivity and texture) does not change in time.
- *Dynamic* when its definition is time-varying.

[2]We will later explain how normal vectors are used in the rendering process.

[3]Note that alternate definitions for texture exist, see [26] for more details.

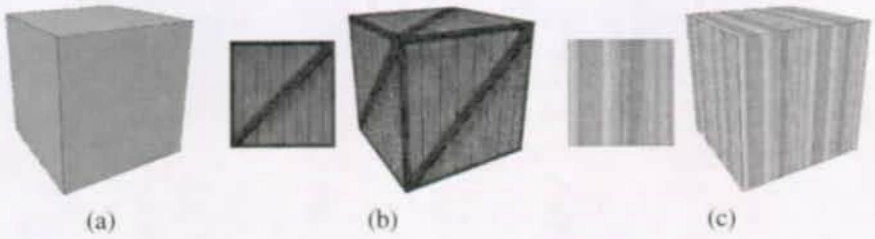


Figure 1.1: Increasing realism using textures: (a) A box modeled with a simple cube. (b) The cube is mapped with a wood texture. (c) A wallpaper texture is tiled on the cube.

For the case of a dynamic source, we can think about an input system generating a texture acquired from a video camera and jointly sending geometry and connectivity information.

1.2.3 Rendering at the receiver

Recall that the source represents a portion of a tridimensional scene. Then, at the receiver an image is rendered from the source given a viewpoint. The *scene view* is defined as

- *Static* when a change in viewpoint does not require new data to be transmitted in order to render the scene.

Conversely, the view is defined as

- *Dynamic* when new data is required to render the scene after a change in viewpoint.

For example, consider that the user visualizes only a small part of a large model and that the transmitter restricts the transmission to currently visible information only. A significant change in viewpoint uncovers new regions of the model and requires new data to be sent. In general, we consider the rendering resources at the receiver as time-varying. For example, the resources can be measured as the number of *textured triangles* rendered per second, denoted by *tri/s*. This measure corresponds to typical characteristics of today's graphic hardware systems.

1.2.4 Channel characteristics and network model

We examine now the entity linking transmitter and receiver: the channel or more generally the underlying network. The rate of the bitstream generated by the transmitter must meet the bandwidth requirements of the transmission channel. Also, the design should assume that the channel performances vary in time and the system is able to adapt to bandwidth variations. The channel resources are expressed in *transmitted bits* per second (*b/s*). Usually, performances are in the magnitude of *kilobits* or *megabits* per second, respectively denoted by *Kb/s* and *Mb/s*.

Figure 1.2 depicts the network model underlying our transmission system. For the moment, we do not assume any particular protocol for the transmission. Each entity (transmitter, channel, receiver) is represented, as well as the constraints. In the figure, a single transmitter supplies three receivers bounded with different channel and resource characteristics. The arrows represent transmitted bitstreams at different rates. The solid, dashed and dotted arrows depict bitstreams at 128, 256, 512 Kb/s, respectively. Note that the values on the links in Figure 1.2 give the total channel capacities and not the bitrate values. The receiver $\mathbf{R}_1$ is fed at the highest bitrate since its downlink and rendering resources result in the greatest capacity combination in the network. $\mathbf{R}_2$ acquires the lowest bitrate despite a 512Kb/s downlink: Its weak rendering performances prevent a fast transmission. Finally, $\mathbf{R}_3$ receives a 256Kb/s bitstream, hence maximizing the link usage. In this example, we assume that the transmitter is computationally able to generate the bitstreams.

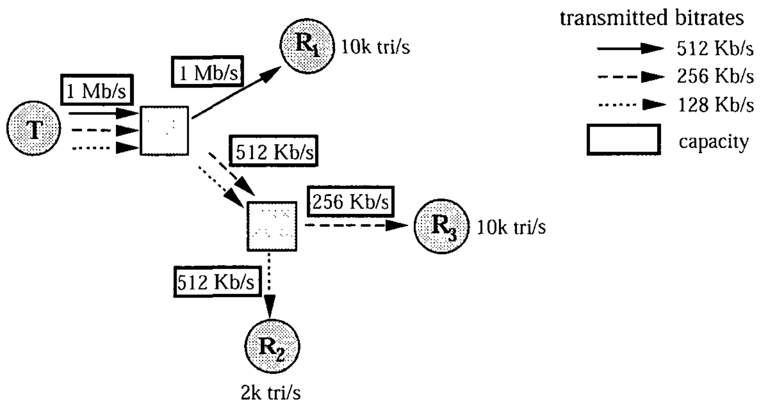


Figure 1.2: Network model and constraints: Each channel is *bandwidth-constrained.* The limitations are expressed in megabits per second (Mb/s) or kilobits per second (Kb/s). Additionally, each receiver has limited rendering performances. The client resources are expressed as rendered textured triangles per second (tri/s).

1.2.5 Transmitter-receiver interactions

This section describes the interaction between the transmitter and the receiver for each type of source (i.e. static or dynamic) and each type of view (i.e. static or dynamic). This interaction is typically implemented with a protocol. We have then four cases of interaction to examine. Each type of interaction, depicted in Figure 1.3, are explained below:

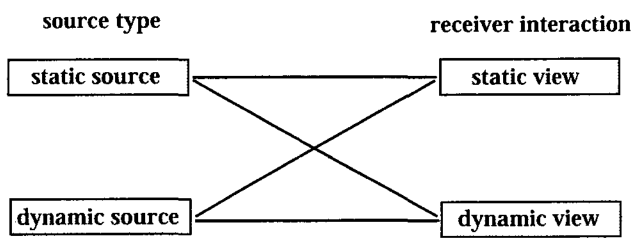


Figure 1.3: Interactions between the receiver and the transmitter for both types of sources and receiver feedback. In the case of a dynamic view, an explicit feedback from the receiver is needed.

- **Static view of a static source (SVSS)**: In this case, no feedback from the receiver is required. To cope with the rendering capabilities at the receiver side, the transmission can be stopped when sufficient information has been transmitted[4]. For example, the download of a complete model fits this case.

- **Static view of a dynamic source (SVDS)**: In this case, a feedback from the receiver is necessary to adapt to the *receiver resources and channel bandwidth variations*. An example for this case is a video-telephony system improved with geometry information.

- **Dynamic view of a static source (DVSS)**: The case is very similar to the SVDS. Again, feedback is needed from the receiver to adapt to the *receiver resources and channel bandwidth variations*. Also, feedback is needed to report viewpoint motions. For example, this type of interaction is needed when viewing very large models, i.e. only part of the scene can be transmitted. Caching algorithms can be used in order to reduce transmission requirements. Implementing caching in the SVDS case is indeed not obvious.

- **Dynamic view of a dynamic source (DVDS)**: In this case, feedback is required to report *receiver resource and channel bandwidth variations* and *viewpoint motions*. Information on the viewpoint is required to know which part of the scene must be encoded by the transmitter. For example, consider a model too large to be transmitted completely and that only a subpart (ideally the visible part) is encoded. A new transmission is necessary when the viewpoint is significantly changed (i.e. new parts of the scene are unveiled).

[4]We will later clarify under which constraints such assumption can be made.

Finally, note that the source can be simplified taking into account the viewpoint (*view-dependent optimization*, as in [44]). In the opposite case, methods are often called *world-space optimizations*. Since world-space based methods do not assume any viewpoint when optimizing the model, there are lighter in transmission since they allow for a larger range of views with the same information.

1.2.6 Optimization problems and constraints

In this section, we explain the main issues to address in our transmission system and examine the constraints to solve in each case of interaction presented in Section 1.2.5. For the design of a transmission system, we must consider the following problems:

1. *The transmission time must be minimized such that scenes are rendered as fast as possible at the receiver.*
2. *The frame rate (number of images displayed per second) must be sustained, according to the receiver capabilities.*
3. *The quality of the rendering must be maximized according to the transmitter, channel and receiver constraints.*

The optimization problem for the transmission can be studied from a *communication point of view* or from a *computer graphics point of view*. Hence, in the first case the encoding of the source is *bandwidth-constrained*, and adapts to the channel capacity. In the second case, the encoding is adapted to the rendering performances at the receiver.

- Problem 1 is solved with a *joint source-channel coding* of the scene. This problem is an active area of research in communications and several solutions have been proposed for the transmission of video [61].
- Problem 2 is solved with a *joint source-rendering coding*. This problem has been addressed in the computer graphics community with multiresolution techniques for graphic models [43, 30].
- Problem 3 is the general problem in a transmission system. In this case, a joint *source-channel-rendering* coding has to be addressed.

We have described four types of interaction between the transmitter and the receiver in Section 1.2.5. For each case, we explain which constraints have to be met:

- For the *SVSS interaction*, only the receiver constraints have to be met. The time to transmit the model will depend on the available bandwidth, therefore a compact encoding of the model will improve the performances. In the particular case of a static source, a pre-optimization of the model can be computed in order to free the system from the computational constraints at the transmitter.

- For the *SVDS interaction*, the constraints are: the transmitter computational resources, the channel bandwidth and the receiver constraints. The system must be efficient to quickly encode the time-varying source definition, and transmit at a rate coping with the channel constraints. Additionally, The complexity of the transmitted scene must match the receiver capabilities.
- For the *DVSS interaction*, the constraints are similar to the ones in the SVDS case. However, as previously explained, caching algorithms can be used in order to lighten the transmission. When sufficient parts of the scene are cached, then the interaction is reduced to the SVSS case. Note that caching might be difficult to implement when the scene is view-dependently encoded.
- Finally, for the *DVDS interaction*, all constraints of the SD case have to be met. However, in this case the receiver constraints are more involved since we must take into account the motion of the viewpoint. This aspect implies a complex receiver-transmitter protocol in order to adapt to the transmission rate.

1.2.7 Proposal for a transmission system

In the previous sections, we identified the elements of a transmission system, as well as their interactions. Also, we explained the main issues and gave the constraints to be met for each type of interaction. In this section, we propose a model for the system (Figure 1.4).

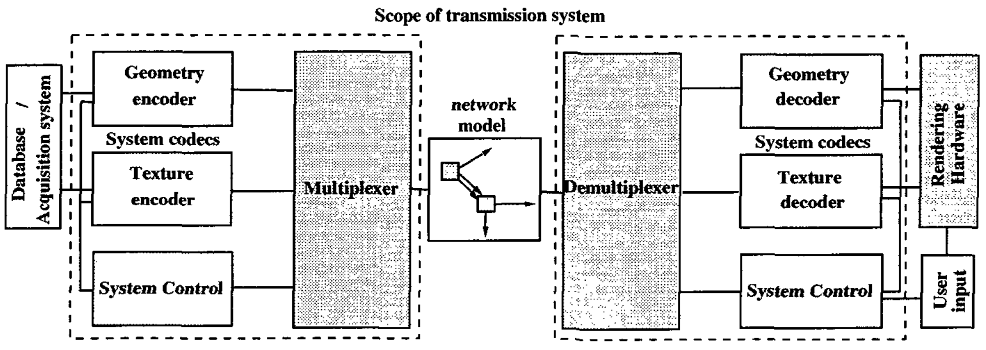


Figure 1.4: Scope and elements of our transmission system.

Our proposal for a transmission system is based on the recommendation for H.324, a low bitrate system for videotelephony [66]. H.324 addresses and specifies a common method for sharing video, data, and voice simultaneously using high-speed (V.34) modem connections over a single analog telephone line. It also specifies interoperability under these conditions, so that videophones, for example, based on H.324 will be able to connect and conduct a multimedia session. Of the three ITU standards that address videoconferencing –H.324, H.323 and H.320– H.324 has the broadest impact in the marketplace. That is because H.324 incorporates the most pervasive communication facility installed today, on a global basis. The proposal for H.324 is very complete and takes into account several practical aspects of the infrastructure (for example the network). Our system is inspired from the H.324 specification because it treats important aspects of the transmission of heterogenous data (in this case video, voice and data). We reuse its main components in our system. We explain now each relevant aspect of the system depicted in Figure 1.4:

System codecs Following our graphic model assumptions, the transmission system ideally contains one codec (encoder/decoder) per type of information. Specifically, a mesh codec optimizes and encodes the geometry and connectivity of the model, whereas an image codec processes the texture data. The set of algorithms, as well as the data structures involved in the processing and the transmission of the models, must be efficient and scalable. Their performances are crucial in the case of a dynamic source, and they bound the system ability to process large models.

System control This component is similar to the *Multimedia System Control H.245* [67] described in H.324. The system control configures the codecs parameters in order to generate a bitstream at optimal rate. In the general DVDS case (Section 1.2.5), this component runs a complex algorithm to determine the optimal bitrate according to the constraints of the system. It also generates the control information, embedded in the protocol involved between the transmitter and the receiver (Figure 1.6.

Multiplexer/demultiplexer A multiplexer creates an heterogenous bitstream containing both geometry, connectivity and texture information, as well as control data. This component operates as H.223 [68] in the specification of H.324. It transforms the codec outputs into a single bitstream according to a pattern given by the *system control*. The pattern allocates a fixed number of bits to each codec output and control data. The pattern definition is dynamic, therefore bits are redistributed under constraint changes.

Finally, note that the interaction cases in Section 1.2.5 are used to define an *application-layer* protocol [37] for the system. To transmit the bitstream, we assume that the use of existing transmission protocols such as UDP, TCP or RTP [33] is sufficient. For example, in the case of a SVSS interaction, TCP can be used since the transmission time is not bounded a priori. However in the SVDS and DVDS cases, protocols like UDP or RTP should be considered.

1.3 Thesis plan

In this section, we give a plan for the dissertation and give comments for each chapters. Apprendix 1.A also gives a short, more visual, plan for this thesis. It can be used by the reader to quickly find important sections, such as contributions and chapter summaries. We explain now each part of our solution.

1.3.1 A journey in piecewise-linear function approximation

In Chapter 2, we begin this thesis with a technical review of approximation algorithms in computer graphics. We apply a set of algorithms to a simple graphical model: monodimensional piecewise-linear functions, or *polylines*. We draw parallels with the multi-dimensional case, namely *mesh simplification*, as often as possible. Polyline simplification is much simpler than its multi-dimensional counterpart because it frees the study from several technical problems. It provides an easy way to understand mesh simplification approaches. It gives the reader an appropriate background for this thesis.

1.3.2 Computational analysis of 4-8 meshes

In Chapter 3, we introduce a class of meshes with subdivision connectivity known as *4-8 meshes*. The connectiviy of these meshes is built using a recursive procedure. We give a study in computational complexity of simplification operations. 4-8 meshes are also used to generate approximations of subdivision surfaces [86]. Our results are also valid for these meshes.

1.3.3 Quadtree data structure for efficient storage and access of 4-8 meshes

In Chapter 4, we propose a set of computationally efficient algorithms and tree-based data structures. *Computational and storage efficiency is crucial to obtain a scalable system and to allow for* large datasets to be processed. In particular, we propose a quadtree data structure with a *constant-time node traversal property*. The quadtree is linear in memory in the sense that it is stored as a linear array of nodes, i.e. the tree does not use pointers. We take advantage of these properties to state computationally optimal algorithms for 4-8 mesh optimization stored in quadtrees.

1.3.4 Progressive meshes in an operational rate-distortion framework

In chapter 5, we address the joint optimization problems described in Section 1.2.6. We provide an algorithm to compute progressive and adaptive representations of 4-8 meshes using global error. The study is led in an operational RD framework. The distortion is generally a distance measure between the original model and an approximation. The rate is defined according to the

optimization problem: For example, a joint source-channel coding problem may define the rate as the number of bits to encode the model. For a joint source-rendering problem, the rate may be defined as the number of triangles fed to the rendering engine at the receiver. We explain that our framework also allows us to address the general problem of the joint source-channel-rendering coding of the model.
We apply our algorithm to terrain data, or equivalently smoothly parametrized surfaces. Consequently, we avoid the issue of dealing with an explicit parameterization for the surface. However, our results are valid for 4-8 meshes built on a control mesh having arbitrary topology.

1.3.5 Joint geometry and texture optimization

In Chapter 6, we explore how geometry can be optimized with texture and vice-versa, i.e. how both entities interactions contribute to the quality of the rendered model.

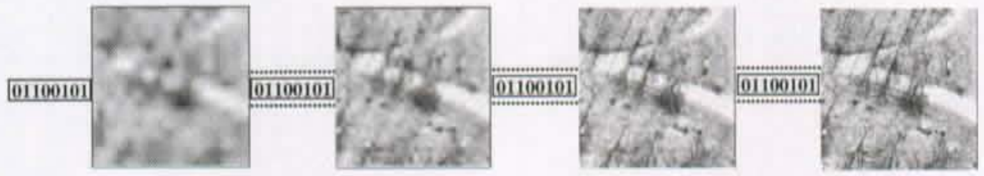


Figure 1.5: Refinement of an aerial photography: Initially a coarse approximation is obtained using a few transform coefficients. Then, the approximation is gradually refined (right to left).

Multiresolution techniques for images based on subband coding have been widely investigated by the image processing community and many solutions have been proposed [51, 92, 60]. Embedded coding schemes are derived from subband decomposition and the encoding rate can be adapted to bandwidth constraints, for example during transmission. Figure 1.5 depicts the progressive refinement of an aerial photograph. Initially, a coarse approximation is obtained using a few transform coefficients (right-hand side of the figure). Then, additional coefficients are added to refine the approximation (from right to left).

In this chapter, we propose to study the interaction between the geometry and connectivity of the model and the texture. We investigate a method to optimize jointly both model entities. Specifically, for a fixed bit budget we determine the optimal amount of geometry and texture information to maximize the rendered quality. Then, we give a computationally efficient greedy method based on marginal analysis to approximate the optimal quantities.

1.3.6 Conclusion

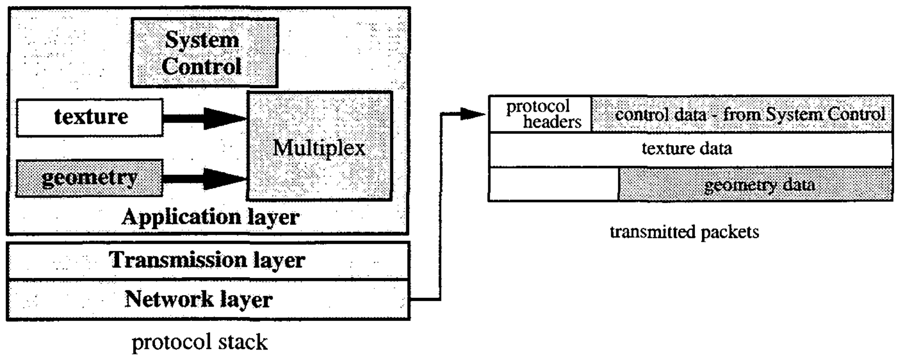


Figure 1.6: *The system control assigns bits for geometry, connectivity and texture inside the packets generated for the transmission by the multiplexer.*

In Chapter 7, we explain how our results fit into our model for a transmission system. Note that we do not address the modeling of a transmitter-receiver (TR) protocol. Instead, we explain how our solutions can overcome the constraint issues in the TR interaction cases (Section 1.2.5).

Recall that the outputs of the encoders in our transmission system are multiplexed into a single bitstream (Figure 1.6). A system control based on H.245 [67] sets the encoding characteristics according to the constraints incurred by the system and generates a bit allocation pattern to produce the outputs (see Section 1.2.7). We use our joint optimization method for geometry and texture to determine a quasi-optimal bit allocation for geometry, connectivity and texture information.

Finally, we propose several improvements for our transmission framework, as well as future research directions.

Appendix 1.A Thesis-at-a-glance

Chapter 2: A journey in piecewise-linear function approximation

Topic Review of monodimensional approximation algorithms (technical background for the thesis).

This chapter reviews a series of monodimensional approximation algorithms applied to a simple graphic model called *polyline*. The algorithms are compared in terms of computational performances, solution errors, and properties. We draw parallels between the algorithms and their couterparts in $\mathbb{R}^3$.

See also: Contributions (Section 2.1.3), Summary (Section 2.8)

Key elements: We show how to obtain a $\Theta(n \log n)$ decimation algorithm using global error. Then, we show how to conserve monotonicity of approximation errors across rate using variable-rate decimation.

Chapter 3: Computational analysis of 4-8 meshes

Topic Study of properties of 4-8 meshes in computational complexity.

We analyze a set of operations used in optimization algorithms for 4-8 meshes. These results are important in order to understand the complexity of our approximation problem.

See also: Contributions (Section 3.1.3), Summary (Section 3.6)

Key elements: We give several computational results when processing 4-8 mesh. In particular, we give an algorithm to find *merging domain intersections.*

Chapter 4: Quadtree data structure for efficient storage and access of 4-8 meshes

Topic Study of an efficient storage and access method for 4-8 meshes.

We build a quadtree data structure achieving minimal storage for the mesh. We propose an access method allowing us to obtain computationally optimal implementations of the algorithms investigated in Chapter 3.

See also: Contributions (Section 4.1.2), Summary (Section 4.8)

Key elements: We express the index differences between quadtree nodes in closed-form. We obtain a generalization for neighbor-finding techniques, called *traveral paths*. Traversal paths are typically used in quadtree algorithms, e.g. mesh simplification algorithms.

Chapter 5: Progressive meshes in an operational rate-distortion sense

Topic Construction of a simplification algorithm using global error with application to terrain data.

We use the results in Chapters 3 and 4 to build an algorithm to approximate 4-8 meshes under constraints (e.g. storage, bandwidth, etc).

See also: Contributions (Section 5.1.2), Summary (Section 5.6)

Key elements: We generalize the global error estimate identified in the polyline case (Chapter 2) to surface simplification. We obtain an efficient $\Theta(n \log n)$ decimation algorithm to simplify 4-8 meshes using global error.

Chapter 6: Joint mesh and texture optimization

Topic Investigation of the interplay between mesh and texture in a rendered model.

We address an important joint optimization problem: We propose to balance the amount of mesh and texture information in order to maximize the rendered quality. Our algorithm fits in the system control and multiplexing units of our tranmission system. We use it to compute the output rates of the geometry and texture encoders feeding the multiplexing unit.

See also: Contributions (Section 6.1.2), Summary (Section 6.5)

Key elements: We identify a strong interplay between mesh and texture in the rendered image quality.

Chapter 7: Conclusion

Topic Concluding remarks and future research directions.

We close this thesis in two steps: first, we place our results in our transmission framework. Then, we propose several improvements to the framework, as well as future research directions.

Chapter 2

A journey in piecewise-linear function approximation

2.1 Introduction

2.1.1 The exact world of approximation

This chapter aims at giving a snapshot of the field of mesh approximation through the simplified problem of piecewise-linear function approximation. We choose a geometric approach like in [8, 3, 47, 18] to lead the study. A signal processing approach (i.e. using filters) is possible [25], nevertheless our choice is coherent with the rest of this dissertation. We present a detailed review and comparison of *approximation algorithms* for piecewise-linear functions. Through the review, we will draw parallels with the related multidimensional problems (i.e. surfaces) whenever possible. Hence, we hope that by investigating relatively simple questions, we will provide the reader with a global picture of the research field of this thesis. Although this chapter takes place as a technical training to the thesis, all the results are original and contribute to the field. We give our contributions in Section 2.1.3.

2.1.2 Optimizing the polyline model

Model and metric Consider a set of samples, denoted by $J_0 = \{k_j\}$ with $j = 0 \ldots n - 1$, where each sample k_j is called *a knot* and corresponds to a couple (x_j, y_j) of coordinates. The piecewise-linear function interpolating the data points is called *a polyline*. In Figure 2.1a, J_0 depicts the polyline interpolating the set of $n = 8$ knots. More generally, we call J_i *an approximating polyline* constructed on a subset of knots and constrains the subset to contain the boundary knots k_0 and k_{n-1}. Consequently, the selection is performed in the set of *internal knots* $k_1..k_{n-2}$. This later ensures that all the approximations share the same domain. Therefore, the coarsest approximating polyline for J_0 is $J_{n-2} = \{k_0, k_{n-1}\}$, as depicted by the dotted polyline J_6 in Figure 2.1a.

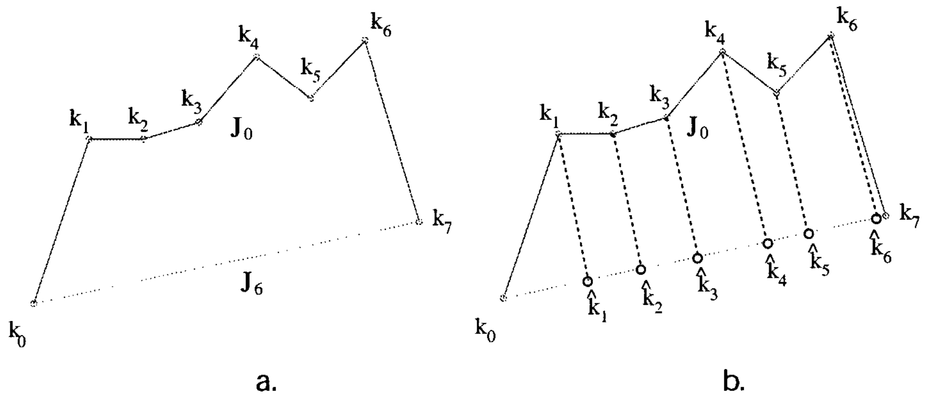


Figure 2.1: Polyline model: (a) This representation interpolates linearly a set of sample knots. The polyline J_0 connects the full set of samples, whereas the coarsest approximation J_6 (when having $n = 8$ knots) is formed by the minimal set of knots (k_0 and k_7) delimiting the domain boundaries. (b) The model assumes the knowledge of the underlying function values at the knots only, therefore the error between the approximations J_6 and J_0 is found by summing the squared differences between the knots and their projections in the approximating segment.

From now on, we restrict the initial set of knots forming the polyline J_0 to be uniformly distributed on the integer line $\mathbb{N}$, i.e. $x_i \in \mathbb{N}$. Moreover, the discrete representation gives no information about the value of the underlying function in between the sample knots. Therefore, to measure the distance between J_0 and any approximation J_i we use the squared l_2 norm evaluated at the sample points only. Specifically for each decimated knot k_j, its orthogonal projection $\hat{k}_j$ in the approximating segment is computed, as depicted in Figure 2.1b. Assume that $\mathbf{j}$ is the index set of the decimated knots for an approximation J_i, then the total *distortion* is defined as the functional

$$D(J_i) = \sum_{j \in \mathbf{j}} (k_j - \hat{k}_j)^2 \tag{2.1}$$

Additionally, we introduce the operator $C(\cdot)$ returning the cost of a polyline J_i. A simple functional for the cost counts the number of linear segments forming the polyline. We simply call this number *the rate of the polyline*. In the example of Figure 2.1a, we have $C(J_0) = 7$ and $C(J_6) = 1$.

Approximation problem The central problem addressed in this chapter is the following: Assume a polyline J_0 (i.e. at full rate), then given a budget of C_i segments, we are interested in finding the approximation J_i such that

$$J_i = \arg \min_{C(J)=C_i} D(J). \tag{2.2}$$

This journey through piecewise-linear function approximation aims at exploring several approaches to this problem. We will review a series of algorithms and evaluate their performances on several important criteria. In the next sections, we first summarize our contributions. Then, we introduce our metric and dataset. Finally, we explain how the different approaches can be compared.

2.1.3 Contributions

In this section, we summarize our contributions:

Algorithm classification We give a classification for the algorithms solving the central problem (2.2). It has the advantage of clearly differentiating the approaches. Also, it gives clues about their performances. Our classification includes the main strategies explored in literature [8, 3, 47, 18].

Comparison in a computation, rate-distortion framework The algorithms are compared in terms of computational cost and solution qualities in an operational rate-distortion framework (more details follow).

Trellis approach for the optimal solutions We propose a trellis structure (usually used in compression [78]) to obtain the optimal solutions to (2.2). Previous formalizations for piecewise-linear approximation used a graph theory approach [47, 77].

Solutions properties We compare the algorithm solutions from a properties point of view. In particular, we address the problem of monotonicity of the distortion across rates, which has not been considered previously. The considered properties are important for our transmission framework.

Global error estimate and variable-rate decimation We show that it is possible to build a computationally inexpensive global error estimate in the tree-constrained case and use an algorithm to *update it*. We include this technique in a variable-rate decimation algorithm, i.e. at each optimization step, a set of knots is decimated. This method allows us to obtain the *optimal constrained solution in an operational rate-distortion sense*. Moreover, the constrained operational rate-distortion curve is strictly monotonic.

2.1.4 Algorithm classification and dataset

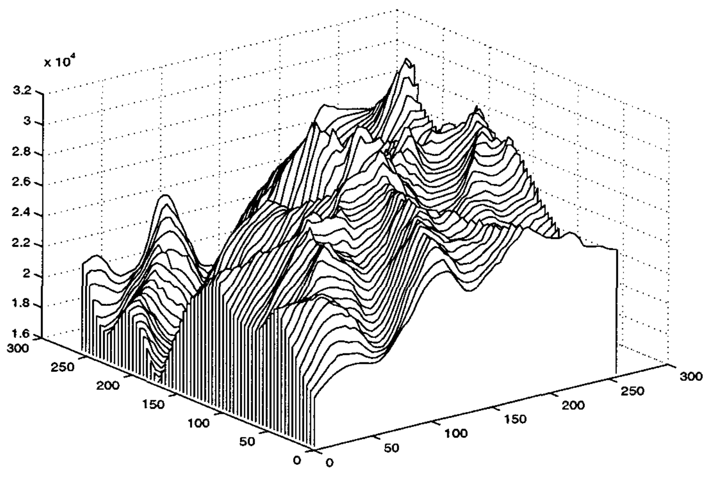


Figure 2.2: Set of 257 curves used for the experiments, where each curve has 257 knots. The set is an elevation matrix representing real terrain data (region in Valais, Switzerland) [16].

Consider an algorithm to solve (2.2), then its performances can be evaluated according to three criteria:

1. the computational and storage cost to find all the solutions J_i for $1 \leq C_i \leq n-1$;
2. the distortion incurred by the solutions (represented by an operational rate-distortion curve, see below);
3. the properties of the solutions.

Point (1) is evaluated given a model of computation. The cost is given as a function of the input size n. We base the cost evaluations (computation and storage) on the *random access machine* (RAM) model [82] and consider their asymptotic behavior. We measure the number of comparisons and error computations (per knot) performed by the algorithm instead of considering more basic operations, as described in the original model (see Appendix 2.A). We assume that for our set of operations, each one has constant and roughly equal cost.

For point (2), we compare the distortions (2.1) incurred by the polyline approximations returned by the algorithm. In this chapter, we use a set of 257 discrete curves, each having 257 knots. The set is a 257 by 257 matrix of elevation data [16] representing a real terrain in Switzerland (region of Valais). The dataset has smooth as well as edgy regions and is adequate to obtain a comparison of the algorithm performances. The set of curves is displayed in Figure 2.2. The measured distortions are normalized between 0 and 10^5 in order to fix the maximum gain to 50 dB. The gain in dB is obtained with

$$E_i = 10\log_{10}(\frac{e_{\max}}{e_i + 1}) \qquad \text{(dB)}, \tag{2.3}$$

where $e_{\max} = D(J_{n-2})$ and e_i measures the error in squared l_2 norm at rate C_i, i.e. $e_i = D(J_i)$. We plot the set of errors $\{E_i\}$ as a function of the rate r ranging $1 \leq C_i \leq n - 1$. The representation is called the *operational rate-distortion (RD) curve.*

Finally for point (3), we review in Section 2.2 a set of important properties for the solutions generated by the different approaches. We identified two properties relevant to our transmission framework: *error monotonicity across rate* (Section 2.2.3) and *progressiveness* of the solutions (Section 2.2.4).

2.2 Approaches to approximation

2.2.1 Idea of successive approximations

The smoothness of a polyline J_i is a function of its number of segments, or equivalently of the number of knots supporting the linear approximation. The more segments the smoother the representation. This assertion is obviously also valid for surfaces, where unit elements are triangles in general (see for example Figure 6.4). Given a polyline J_0, successive approximations are obtained by varying the constraint C_i in (2.2). Then, each approximating polyline J_i is represented by a selection of knots in J_0 satisfying the constraint. Figures 2.3a-f represent a set of successive approximations at increasing rate. In each figure, the selected knots supporting the approximating polyline are represented by the white points.

The idea of successive approximations is the key to our transmission problem. The cost (in storage, bandwidth or computational resources) of an approximation increases monotonically with the rate of the model (whether we have a polyline or a surface). Therefore, the *approximation* of models (i.e. the reduction of unit elements, either segments or triangles) and the related problems are at the core of this thesis.

2.2.2 Error dependency

Recall that we defined a polyline as a set of connected knots forming a planar approximation of an underlying function for which only a set of discrete samples is known. Consider the initial polyline J_0, then only in this particular case, the error incurred at each knot will also

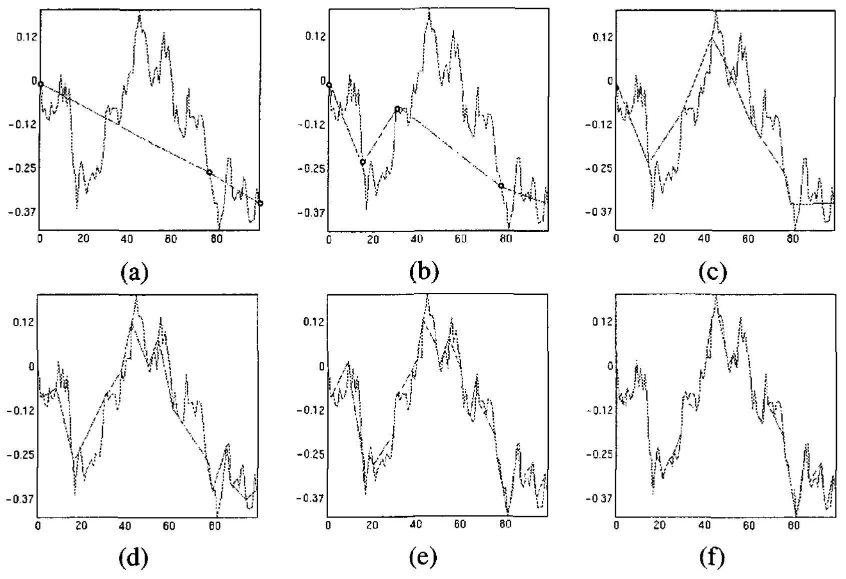


Figure 2.3: Successive approximations using: (a) 3 knots, (b) 5 knots, (c) 9 knots, (d) 17 knots, (e) 28 knots, (f) 45 knots.

be the global error increase when the knot will be decimated. However, because of the error dependency (Section 2.2.2), the global error through iterations is *not equal to the sum of the local errors of the decimated knots*, as explained below.

The polyline representation is equivalent to the expansion of the knots on a linear spline basis. Figure 2.4a shows the underlying spline basis functions and the element supporting knot k_1 is put in evidence. Consider now the approximation $J_1 = \{k_0, k_1, k_3, k_4, k_5\}$ in Figure 2.4b: The domain of the k_1's basis element is now larger and vanishes at knot k_3, instead of knot k_2 as in J_0 (Figure 2.4a). Consequently, the errors incurred when decimating k_1 or k_3 in J_1 are different from their respective values in J_0. The same observation can be made regarding knots k_1 and k_4 in $J_2 = \{k_0, k_1, k_4, k_5\}$ (Figure 2.4c). Therefore the errors incurred at the knots depend on the polyline construction, meaning that, when applying an approximation scheme, such as the successive decimation of J_0, each approximation changes locally the knot errors. In the particular case of linear spline basis, we can observe that the domain of each basis element extends only to closest undecimated neighbor knots (see Figures 2.4a-c). As a consequence, only the error at these two knots is modified by the decimation.

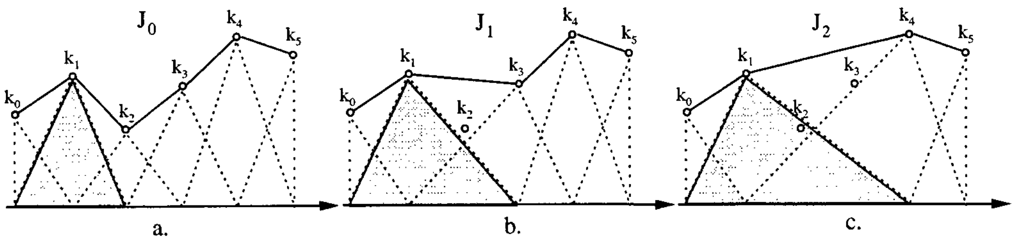


Figure 2.4: Illustration of the error dependency due to the connected representation of graphic models: (a) The underlying spline basis connecting the knots in J_0. The support of knots k_1's basis is put in evidence. Parts (b) and (c) show the changes of the support for two approximations of the polyline (J_1 and J_2, respectively) in part (a).

The above observations for polylines also hold for higher dimensional models such as surfaces. The dependency of the error caused by the connected representation of graphic models has important consequences on approximation algorithms. The first one is that after each optimization step (for instance, inserting or decimating a knot), some errors in the model must be updated. Tracking the outdated errors in the polyline case is simple, whereas in the case of surfaces this problem becomes more complex as will be seen in Chapter 3. A second important consequence of the error dependency is addressed in the next section.

2.2.3 Monotonicity of solution errors

Assume a polyline J_0 with $|J_0| = n$ and an algorithm to solve problem (2.2) using the squared l_2 norm as distortion functional for a range of constraints $1 \leq C_i < \ldots < C_{i-1} \leq n - 1$ (the corresponding solutions being denoted by J_i respectively). Then, in general the distortion of the solutions does not increase monotonically when the rate decreases.

Consider the simple example in Figure 2.5a: For a polyline J_0 with 5 knots, the series of optimal configurations (i.e. minimizing the square l_2 norm distance) are depicted in Figures 2.5b-d. The characteristics for each configuration, namely the costs in segments and the global errors, are given in Table 2.1. This example shows that the monotonicity of the operational RD curve is not conserved even in the optimal case. Monotonicity is an important property for the solutions of an approximation algorithm. It ensures that, each refined approximation decreases the global error. Although a norm conserving the monotonicity could be defined, typical "interesting" norms such as l_1, l_2 and l_∞ do not verify this property in general. We will see, however, that it is possible to achieve monotonicity in a particular case, even using the above nonmonotonic norms. Finally, note that the monotonicity is obtained on average.

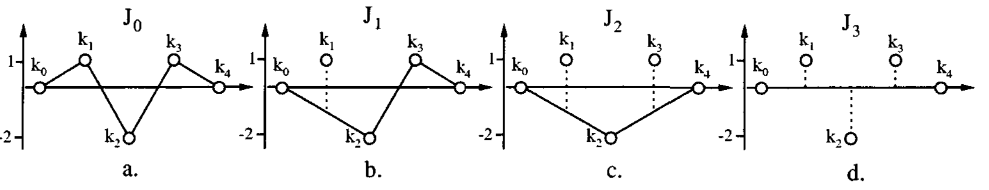


Figure 2.5: Optimal decomposition and global error: (a) The polyline J_0, and the optimal approximations (b) J_1 (with $C_1 = 3$), (c) J_2 (with $C_2 = 2$), (d) J_3 (with $C_3 = 1$). For each approximation, the error incurred at the knots is depicted with a dotted line.

Configuration	Cost $C(J_i)$	Distortion $D(J_i)$	Figure
J_0	4	0	2.5a
J_1	3	4	2.5b
J_2	2	8	2.5c
J_3	1	6	2.5d

Table 2.1: Global error $D(j_i)$ in squared l_2 norm and cost $C(J_i)$ in segments for the configurations shown in Figures 2.5a-d.

2.2.4 Progressiveness

Recall that a polyline J_i is simply defined as a set of knots. Progressiveness is achieved when the successive approximations for J_0 verify

$$J_{n-2} \subset \ldots \subset J_1 \subset J_0, \tag{2.4}$$

i.e. the decomposition is given by a set of *embedded knot configurations*. Progressive descriptions are particularly interesting for our transmission framework since, given a model J_i, it suffices to transmit the missing knots in order to reconstruct J_{i-1}. Therefore, the embedding of solutions is an important property of an approximation algorithm.

In the polyline case, it suffices to sort the knots k_j by increasing index j to reconnect them (i.e. forming the set of segments) without ambiguity. In the case of surfaces, the *connectivity* of the vertex set (i.e. how the vertices are connected to form a set of triangles) might be harder to reconstruct without jointly sending side-information. We will clarify how the connectivity in the multidimensional case can be constructed in later sections.

2.3 Best solutions using dynamic programming

2.3.1 Optimal algorithm using a causal approach

In this section, we start our exploration of piecewise-linear function approximation. In the monodimensional case, there is an algorithm to obtain the optimal solutions to (2.2). The well-known optimization method called *dynamic programming* (DP) is used to find these optimal solutions. Unfortunately, this method does not generalize to higher dimensions and we will discuss the multidimensional case in Section 2.3.3. DP solves problems by combining the solution to subproblems. Here, "programming" refers to a tabular method and not to writing computer code. The algorithm described here, which we named `uoptimal`, uses a causal approach to retrieve the optimal solutions to (2.2).

The general idea about DP is that it solves subproblems exactly once and stores the result in a table for further look-up. There are two ingredients that a problem must have for DP to be applicable: *optimal substructure* and *overlapping subproblems*. We explain below these requirements and prove that our polyline approximation problem fulfills them.

Optimal substructure The problem should exhibit an optimal substructure. Such a property is verified when an optimal solution contains within it optimal solutions to subproblems. Assume that such a solution under constraint C_i includes knot k_l, then the polyline J_i has the form

$$J_i = \{k_0, \dots, k_l, \dots, k_{n-1}\}, \tag{2.5}$$

where the dots suggest additional knots. Consider now subpolylines $J_{i,1}$ and $J_{i,2}$ defined as

$$J_{i,1} = \{k_0, \dots, k_l\}, \quad J_{i,2} = \{k_l, \dots, k_{n-1}\}, \tag{2.6}$$

then clearly $D(J_i) = D(J_{i,1}) + D(J_{i,2})$, therefore both subsolutions $J_{i,1}$ and $J_{i,2}$ must also be optimal in order for J_i to be optimal as well. This proves that our problem shows optimal substructure.

Overlapping subproblems In order to take advantage of DP, our problem must also exhibit overlapping subproblems. Our algorithm based on DP takes a causal approach: Starting at knot k_0, the knots k_i are successively examined (i.e. by increasing index j) and used to form temporary[1] subpolylines of increasing rate. We denote by $J_{i,j}$ a subpolyline of rate i at step j of the algorithm, with $0 \leq j \leq n-2$. At step 0, knot k_1 is considered and the solution

$$J_{1,0} = \{k_0, k_1\} \tag{2.7}$$

[1] i.e. Temporary in the sense that they only cover a part of the domain.

is formed. At the next step j, the solutions obtained at the preceding stage are upgraded with knot k_{j+1}. Additionally, the "single segment" solution $J_{1,j+1}$ is added, leading to

$$J_{1,1} = \{k_0, k_2\}, \quad J_{2,1} = \{k_0, k_1, k_2\}. \tag{2.8}$$

We have a trellis approach to the problem (Figure 2.6a-b). Trellises are usually used in compression [78]. Previous works [47, 77] formulated the problem in graph theory, however our solution is more intuitive: The top row in the structure counts the algorithm steps and the column shows the rate of the temporary subpolylines. Each point represents a configuration $J_{i,j}$, whereas the lines in between show the upgrades of the solutions. The storage cost of the algorithm is directly represented by the number of configurations (i.e. points) in the structure. Therefore, it is easy to see that the algorithm requires quadratic storage.

The second step of the algorithm is represented by the trellis in Figure 2.6a. In this figure, point A has two adjacent lines, corresponding to the upgrade of both $\{k_0, k_1\}$ and $\{k_0, k_2\}$ with knot k_3. Both temporary solutions have equal rate and reach the same knot, therefore it suffices to choose the one having lowest distortion to obtain the optimal approximation between knots k_0 and k_3 using 2 segments. This remark illustrates two *overlapping subproblems*. In our polyline approximation problem, overlapping subproblems are *temporary solutions of equal rate ending at the same knot.* These solutions can be compared without restriction since they fulfill the optimal substructure condition described in the previous paragraph. Therefore, only the minimal distortion solution will be stored at point A and used in the following steps of the algorithm. We now quickly compute the size of the solution space and then continue our example.

Space of solutions Assume a polyline having m internal knots, then there are

$$\binom{m}{p} = \frac{m!}{(m-p)!p!} \tag{2.9}$$

possible configurations using p of the m internal knots, or equivalently having rate $p + 1$. Then, at each step j, there are $\binom{j}{i}$ solutions of rate $i + 1$, with $0 \leq i \leq j$. Fortunately, exploiting overlapping subproblems avoids examining the exponential number of combinations. We now continue our example and compute the algorithm complexity in Section 2.3.2.

Assume that J_0 is built on 4 knots, and therefore that step 3 (Figure 2.6b) is the last step of the algorithm. For point B in the figure, we must choose $J_{2,3}$ as the lowest distortion solution between the overlapping subproblems

$$\{k_0, k_1, k_4\}, \quad \{k_0, k_2, k_4\}, \quad \{k_0, k_3, k_4\}. \tag{2.10}$$

Assume now that the optimal solution at point A after step 2 is

$$J_{2,2} = \{k_0, k_1, k_3\}, \tag{2.11}$$

then for point C in the figure, we have to choose for $J_{3,3}$ between

$$\{k_0, k_1, k_3, k_4\}, \quad \{k_0, k_1, k_2, k_4\}. \tag{2.12}$$

The last column of points in the trellis therefore represents the optimal solutions for each rate. The above example shows how, at each step, DP uses the results for overlapping subproblems in order to retrieve the optimal solutions minimizing the global distortion. The algorithm pseudo-code is given in Appendix 2.B.

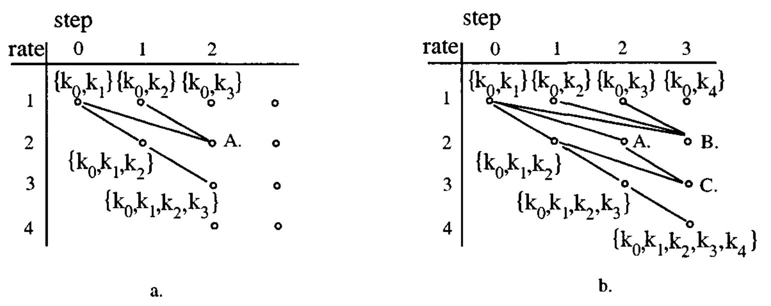


Figure 2.6: Trellis structure describing the causal algorithm using dynamic programming: Each point represents a configuration. (a) Step 2 of the algorithm: For the configuration at point A, we choose between $\{k_0, k_1, k_3\}$ and $\{k_0, k_2, k_3\}$. (b) Step 3 of the algorithm: At point B and C, we choose between three and twp configurations, respectively. The choices are represented by the edges adjacent to each point.

2.3.2 Dynamic programming complexity

With the algorithm review given in the preceding section, the computational complexity can be easily evaluated: Without considering overlapping subproblems, we would have to examine exactly

$$\sum_{i=0}^{j} \binom{j}{i} \tag{2.13}$$

configurations at each step. Taking advantage of overlapping subproblems is well illustrated by decomposing the binomial $\binom{j}{i}$, i.e.

$$\sum_{i=0}^{j} \sum_{l=0}^{j-i} \binom{i-1+l}{i-1}, \tag{2.14}$$

where each element $\binom{i-1+l}{i-1}$ represents a set of overlapping subproblems of rate i. Therefore, we need to update and compare only $j - i + 1$ configurations for each rate i, yielding a total of

$$2\sum_{i=0}^{j} j - i + 1 = j^2 + j \tag{2.15}$$

operations (update and comparisons) for step j. Additionally, for each step the error contributions between knot k_l and k_{j+1}, with $l = 0 \ldots j$ must be evaluated in order to perform all updates at step j, giving

$$\sum_{l=1}^{j} l = \frac{j(j+1)}{2} \tag{2.16}$$

errors to compute. Since the algorithm runs in $n - 1$ steps, the total computational complexity to obtain the optimal solutions for all rates is given by

$$\sum_{j=0}^{n-2} j^2 + j + \frac{j(j+1)}{2} = \frac{1}{2}n^3 - \frac{3}{2}n^2 + n. \tag{2.17}$$

The algorithm yields the optimal solutions in squared l_2 norm in $\Theta(n^3)$ computational time (2.17) and requires $\Theta(n^2)$ storage. However, the algorithm does not conserve the monotonicity of solutions across rates, since its application on the polyline in Figure 2.5a leads to a nonmonotonic operational RD curve. Since we cannot ensure that the configurations are embedded in general, this scheme does not yield a progressive decomposition of the polyline.

2.3.3 Discussion of the multidimensional case

As said previously, approximation schemes based on DP do not generalize to the multidimensional case. The problem of the optimal selection, i.e. minimizing the distortion metric, of a set of vertices in the general setting (without assuming a hierarchy within the set) is known to be NP-hard [2, 1].

2.4 Reducing complexity using greediness

2.4.1 Greedy strategy

Although solving (2.2) with `uoptimal` leads to the optimal solution, its computational and storage complexities are rather dissuasive for large datasets. In this section, we continue our journey by exploring *greedy approximation techniques*. Greedy techniques are computationally and spatially efficient schemes attempting to reach a close to the optimal solution by making choices based only on local considerations (i.e. the choice that seems the best at the current iteration), hence choices that do not depend on future appraisals.

Although it is usually difficult to tell whether a greedy scheme can solve optimally a particular approximation problem, we can distinguish two ingredients exhibited by most problems that lend themselves to a greedy strategy: *optimal substructure* and *greedy-choice property*. As explained in Section 2.3.1, the first requirement is verified for our problem. We explain the second one below:

Greedy-choice property For a problem with this property, the globally optimal solutions can be found by making locally optimal choices. In our polyline approximation problem, we must prove that greedy choices at each step can lead to a global optimal solution. To do so, we must show that such a choice reduces the problem to similar but smaller subproblems. In other words, it suffices then to show that our problem exhibits *optimal substructure*! We gave an example in Section 2.3.1 illustrating that this property actually holds for our problem. Although a greedy strategy does not yield the optimal solutions in general, we will give an example that can be optimally solved.

A final and important remark about greedy schemes is that, due to their shortsighted choices based on local considerations, these methods tend to accumulate errors through the iterative process.

2.4.2 Greedy refinement and decimation

The greedy algorithms presented here use a *local error estimate* to optimize the polyline. In practice, at each iteration error criteria are computed only at the knots and the decimated knots are not taken into account. In contrast, a *global error estimate* would take the decimated knots into account. Expectedly, algorithms using a global approach are computationally more intensive and usually have greater complexity magnitude than the ones using local criteria. In the multidimensional case (e.g. mesh approximation), most approaches use local errors because computational complexity is of greater concern. We will introduce a set of *constrained approximation approaches* in Section 2.5. Interestingly, it is possible in this case to compute a global error estimate (Section 2.5.4) by just using an update mechanism.

There are two ways to replace the optimal knot selection performed by `uoptimal` : We can either *iteratively refine* the coarsest approximating polyline J_{n-2} (i.e. the single segment solution) by inserting knots, or the polyline J_0 can be *iteratively decimated* until no internal knot remains. Greedy refinement schemes have been popular in Cartography since the early 70's [18, 10, 20, 72] and good results are reported using the l_∞ norm [89]. Greedy decimation schemes have been investigated in computational geometry [8, 57, 62]. Figures 2.7a-d illustrate the first scheme (*greedy refinement*): Figure 2.7a shows the coarsest approximation J_3. In Figure

2.7b, the knot incurring the highest distortion[2] (dotted line in the Figure 2.7a) is inserted. In Figure 2.7c and 2.7d, the knots are iteratively inserted in the same manner. The pseudo-code for the greedy refinement algorithm, named `urefine` , is given in Appendix 2.C.

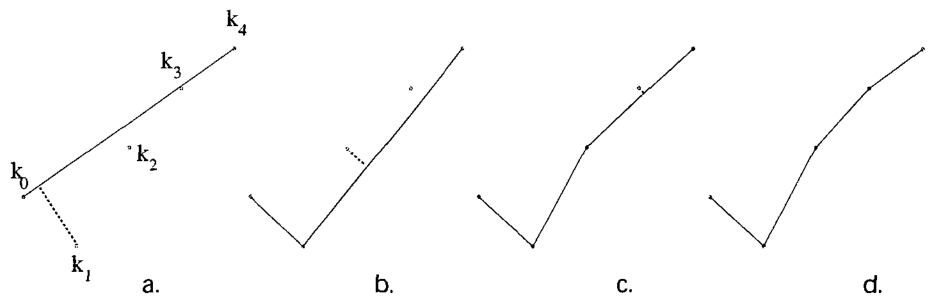


Figure 2.7: Greedy refinement algorithm: (a) The coarsest, single-segment solution. (b) Knot k_1, incurring the highest distortion at this step, is inserted. Similarly, (c) knot k_2 and (d) knot k_3 are inserted.

Figures 2.8a-d illustrate the second scheme (*greedy decimation*): Figure 2.8a shows the initial polyline J_0. In Figure 2.8b, the knot incurring the minimal distortion is decimated (k_3) and the scheme is iterated similarly in Figures 2.8c-d. The pseudo-code for the greedy decimation algorithm, named `udecimate` , is given in Appendix 2.D.

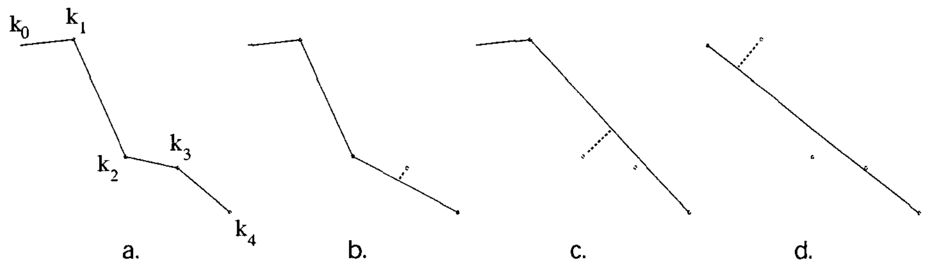


Figure 2.8: Greedy decimation algorithm: (a) The initial polyline J_0. (b) Knot k_3, incurring the minimal distortion is inserted, then iteratively (c) knots k_2 and (d) k_1 are decimated.

[2]As seen previously in Section 2.1.2, the error is computed using the knot projection into the approximating segment.

Both schemes, refinement and decimation, can be a replacement for `uoptimal`. In the next section, we compare the properties of both greedy strategies and compare them to the optimal algorithm using DP.

2.4.3 Complexity and results comparisons

An attractive aspect of greedy strategies is their simplicity: Both algorithms introduced in the previous section operate in an iterative manner. Their computational time is spent into choosing the best knot to insert/decimate at each step and updating knot errors after the new approximating polyline has been constructed. In both cases, the storage complexity is linear, since they require to store, at most, as many errors as knots in J_0. The computational complexities are given below:

Greedy refinement computational complexity Denote by m the number of internal knots (i.e. $m = n - 2$), then initially we need $2m$ operations to compute and retrieve the maximum error. On average, the middle knot is chosen, splitting the initial single segment approximation into 2 segments of equal size. Thus, we roughly need to recompute twice

$$\frac{m-1}{2} \tag{2.18}$$

errors, one set on each side of the inserted knot. Note that we need as many comparisons to obtain again the maximum error per segment. All maximum errors are sorted in a vector in order to make the optimal choice. Since each refinement splits a segment into 2 parts, two errors are inserted in the sorted vector at each step. We need approximately $2\log_2 j$ operations[3] to perform this task at step $j > 0$. Moreover, at most $\frac{m}{2}$ steps require new insertions. This simple scheme suggests that the total number of operations to compute *and* retrieve the maximum errors –yielding the factor 2 in front of the first sum in (2.19)–, in order to perform the complete refinement, is given by

$$2\sum_{j=0}^{\log_2 m-1}(2^j\frac{m-2^j+1}{2^j}) + 2\sum_{j=1}^{m/2}\log_2 j = 2\sum_{j=0}^{\log_2 m-1}(m-2^j+1) + 2\log_2(\frac{m}{2}!). \tag{2.19}$$

The sum corresponds to a geometric series[4] having solution

$$2m\log_2 m - 2m + 2\log_2 m + 2. \tag{2.20}$$

Pose $\zeta = \log_2 e$ and let's approximate the factorial term using the Stirling formula, i.e. $\frac{m}{2}! = \sqrt{\pi m}(\frac{m}{2e})^{\frac{m}{2}}$. Hence, we have

$$2\log_2(\frac{m}{2}!) \approx m\log_2 m + \log_2 m - m\log_2(2e) = m\log_2 m + \log_2 m - m(1+\zeta), \tag{2.21}$$

[3]To be more accurate, we would need $\log_2 j + \log_2(j+1)$ operations.

[4]Note that the initial $2m$ operations are comprised in the series.

then the computational complexity on average is given by

$$3m\log_2 m - m(3+\zeta) + 3\log_2 m + 2. \tag{2.22}$$

We obtain logarithmic functions is (2.22) for the following reason: In the average case, we assume that each insertion merely splits a segment in two equal parts. What happens when this assumption is not met? This algorithm has a worse case too: Assume that at each step, the rightmost and leftmost knot is always picked up (choose one side only and conserve it through the whole process), then at each step j we need $2(m-j)$ operations to compute the errors and retrieve the maximum distortions. Thus, the complexity in the worst case is given by

$$2\sum_{j=1}^{m} m - j = m^2 - m. \tag{2.23}$$

In conclusion, `urefine` has complexity $\Theta(n\log n)$ on average and $\Theta(n^2)$ in the worst case. Note finally that the worst case is very unlikely.

Greedy decimation computational complexity Assume again that m denotes the number of internal knots, then initially we need to compute and sort m errors, leading to

$$m + \sum_{j=1}^{m} \log_2 j = m + \log_2(m!), \tag{2.24}$$

then using $m! = \sqrt{2\pi m}(\frac{m}{e})^m \approx m\log_2 m - \zeta m + \frac{1}{2}\log_2 m$, we have a total of

$$m\log_2 m + (1-\zeta)m + \frac{1}{2}\log_2 m \tag{2.25}$$

operations. At each step j, the error of only two knots is modified by the decimation (recall the example in Figure 2.4). Therefore, we first need to retrieve the outdated ones form the sorted error vector, compute the new ones and insert them again. Each step j totalizes then $2 + 4\log_2(m-j)$ operations (assuming that the vector keeps size $m-j$ during step j). After the initialization, we still need to run $m-1$ steps, then the computational complexity of the iterating process is given by

$$\sum_{j=1}^{m-1}[2 + 4\log_2(m-j)] = 4\log_2((m-1)!) + 2(m-1). \tag{2.26}$$

Using $(m-1)! = \sqrt{2\pi(m-1)}(\frac{m-1}{e})^{m-1}$, we have that

$$4\log_2((m-1)!) \approx 4m\log_2(m-1) - 2\log_2(m-1) - 4\zeta m, \tag{2.27}$$

therefore, we obtain the average computational complexity to perform the complete decimation by summing (2.25) and (2.26), i.e.

$$m \log_2(m(m-1)^4) + (3 - 5\zeta)m + \log_2(\frac{\sqrt{m}}{(m-1)^2}) - 2. \tag{2.28}$$

In conclusion, udecimate has complexity $\Theta(n \log n)$ on average.

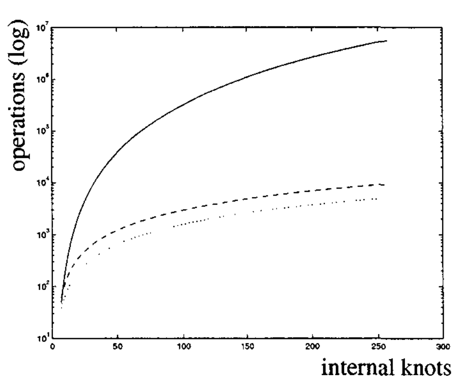


Figure 2.9: Computational complexities comparison: The solid, dashed and dotted curves depict the computational complexities (number of error computations, comparisons and updates) for uoptimal , udecimate and urefine (respectively) as a function of the number of internal knots.

Computational complexities comparisons Both analyses above give the computational complexities "up to the operation", which allows for their comparison in the graph of Figure 2.9. In the figure, the solid, dashed and dotted curves depict the computational complexities (number of error computations, comparisons and updates) for uoptimal , udecimate and urefine (respectively) as a function of the number of internal knots. We can read in the graph that, although udecimate and urefine have the same complexity magnitude, urefine has a smaller factor.

Error comparison In Figure 2.10a, we compare the distortion of the solutions returned by uoptimal , urefine and udecimate using our dataset (Figure 2.2). The solid (top) curve shows the errors obtained with the optimal algorithm using DP. The dashed and dotted curves depict the errors returned by the decimation and refinement approaches, respectively. The maximum gain using DP over the greedy algorithms is 3.86 dB. Recall that, in Section 2.4.1, we mentioned that greedy techniques tend to accumulate errors through iterations because of their shortsighted, locally optimal choices. At low rates, the refinement algorithm, having run few iterations, performs better than its decimation counterpart (Figure 2.10b). At high rates however, the opposite situation is observed (Figure 2.10c).

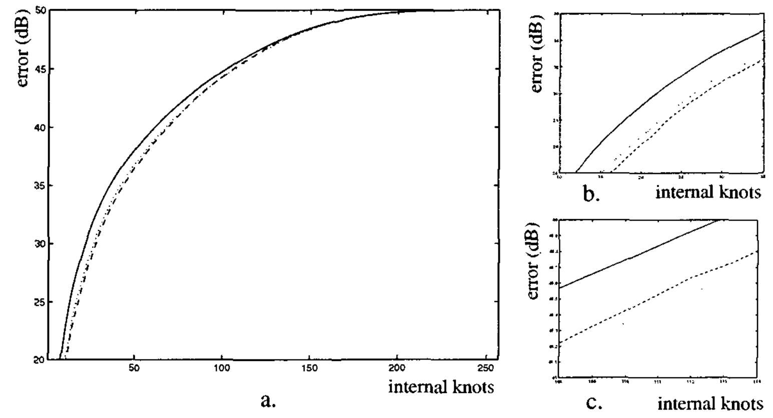


Figure 2.10: Solution errors comparison: (a) The solid (top), dashed and dotted curves show the errors obtained with `uoptimal`, `udecimate` and `urefine`, respectively. (b) At low rates, the refinement method performs better than the decimation approach, whereas (c) at high rates, the opposite situation is observed.

Solution properties comparison As seen previously, the configurations returned by the optimal algorithm neither conserve error monotonicity nor verify progressiveness. In the greedy cases, the solutions are naturally embedded due to the iterative approximation process. Therefore, the successive approximations verify the progressiveness property. In Section 2.4.4, we will see however that this property is only partly verified in the multidimensional case. Unfortunately, none of the greedy techniques achieves monotonicity of the solution errors across rates. Counterexamples for each case are again obtained with the example function in Figure 2.5a.

Finally, it is interesting to see that, when applying either the greedy refinement or decimation scheme on the function in Figure 2.5a, the optimal solutions (the ones shown in Figures 2.5b-d) are obtained. This fact proves that our polyline approximation problem verifies the greedy-choice property introduced in Section 2.4.1.

2.4.4 Discussion of the multidimensional case

Greedy decimation and refinement algorithms are suitable for surface approximation. These approaches has been investigated by several researchers [40, 30, 44, 43]. In the multidimensional case however, two major issues appear:

1. The connectivity of the vertex set (defining the set of triangles) must be computed using additional algorithms.
2. Efficient mechanisms must be investigated in order to find the outdated vertex errors after each optimization step.

Consider the simple example of a set of vertices in the plane distributed in a square region, as depicted in Figures 2.11a-d. To solve the first issue, Delaunay-type algorithms are used to compute the connectivity of the vertices, leading to a so-called *triangulated irregular network* (TIN). Figures 2.11a-d depict the triangulations returned by the original Delaunay algorithm for

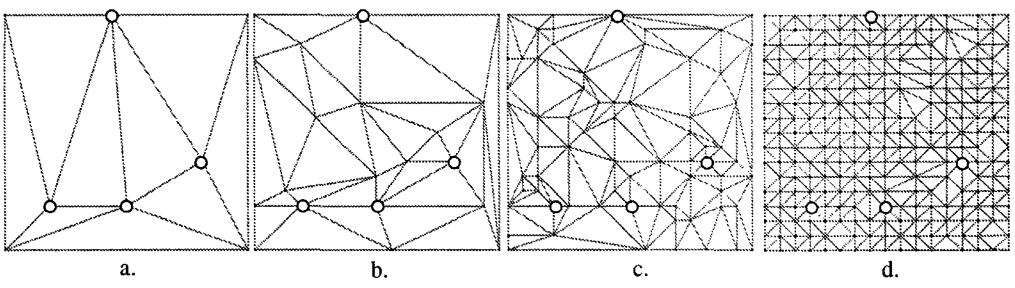


Figure 2.11: Successive triangulated irregular networks (TIN): (a) A set of 8 vertices is connected using Delaunay algorithm. (b) When refined, the retriangulated set does not contain the initial triangles connecting the vertices marked with white points. Each refinement in (c) and (d) leads a new triangulation. Therefore, the successive approximations obtained using TINs are not progressive.

varying sizes of the vertex set. We will explain later the meaning of the marked vertices (white dots) in the figures.

The typical cost for triangulating a set of n vertices using this technique is $O(n \log n)$. Each time a vertex is inserted or decimated, the set must be triangulated again. In Figure 2.11a, we depicted the triangulation for a set of 8 vertices, where the interior vertices are represented with white dots. In Figure 2.11b, additional vertices are present such that a total of 32 vertices are connected. The key observation is that the initial triangles connecting the vertices in Figure

2.11a are no more part of the refined triangulation in Figure 2.11b (see the white vertices location in the figure). Therefore in the case of TINs, *successive triangulations are not embedded*: For example, the triangulation in Figure 2.11b cannot be obtained by splitting triangles of the triangulation of Figure 2.11a. Figure 2.11c and 2.11d show that, each addition of vertices leads to a new triangulation. In other words, the *connectivity is not progressive* and must be specified for each approximation. For this reason, TIN are a priori not suitable for progressive transmission (or rendering).

For the second issue, the fact that the set can be arbitrarily connected makes difficult to find the vertices whose error is outdated. Therefore, additional computational time must be spent to update the surface characteristics and inevitably impedes the performances of approximation algorithms.

2.5 Tree-constrained approximation

2.5.1 Introduction

In Section 2.4.4, we have seen that in the multidimensional case, the freedom involved by inserting or decimating arbitrary vertices has an annoying side-effect: The progressiveness is not generalized to the connectivity of the vertex set. In the following sections, we explore a technique to overcome this drawback: We impose a *hierarchy*, or *constraints* on the model. This approach has been successfully investigated in the early 80's in [3]. We rewrite then our insertion and decimation algorithms for our polyline approximation problem in this restricted setting. We will again discuss the multidimensional case in a later section.

Building a hierarchy on the knots A way to impose constraints is to establish a one to one correspondence between the knots and the nodes of a binary tree: Assume a polyline built on n knots k_j, with $j = 0 \ldots n-1$, where $n = 2^d + 1$, $d > 0$ and a binary tree of depth d containing $2^d - 1$ nodes, numbered $p = 0 \ldots 2^d - 2$. Then, given p we have k_j with

$$j = 2^{d-l-1} + p_l 2^{d-l}, \tag{2.29}$$

where $l = \lfloor \log_2(p+1) \rfloor$ and $p_l = p - 2^l + 1$. The index p_l can be seen as a local index for the node within its level l. The tree nodes map the set of $1 \le m \le n-2$ internal knots, as depicted in Figure 2.12. In the figure, the arrows link knots with their corresponding tree node.

The full rate polyline J_0 corresponds to a full tree. In order to satisfy the constraints, any approximating polyline J_i must yield a subtree, denoted by $|J_i| < |J_0|$. Therefore, an immediate implication is that the space of possible configurations for approximating polylines is severely reduced. For instance, without constraints a polyline with m internal knots have $\binom{m}{1}$ candidate approximations with 2 segments. In the constrained case, there is only one single two segments approximation: The configuration $\{k_0, k_{\frac{n-1}{2}}, k_{n-1}\}$ corresponding to the tree root $p = 0$. Fortunately, the number of possible configurations increases exponentially with rate. In the next paragraph, we compare the space of constrained and unconstrained solutions.

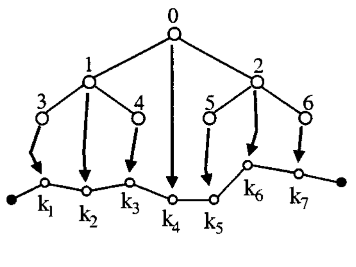


Figure 2.12: Illustration of the mapping between tree nodes and polyline knots.

Space of solutions In the case of constrained approximation, a way of enumerating the space of solutions having k internal knots is to count the number of trees, denoted by $t(k)$, yielding such a configuration. This number is valid for a polyline with *at least* $2^{k} - 1$ internal knots, i.e. $|J_0| \geq 2^k + 1$.

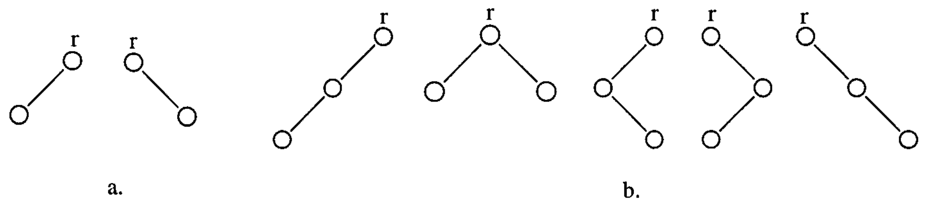


Figure 2.13: Partial trees representing constrained solutions (the index r denotes the tree root node): (a) The two solutions with two internal knots. (b) The five solutions with three internal knots.

When $k = 1$, then only the singleton tree exists, i.e. t(1) = 1. For $k = 2$, we have only two solutions, as depicted in Figure 2.13a. The case $k = 3$ is depicted in Figure 2.13b: To count the number of possible trees, we add exactly one single branch to each solution obtained at step $k = 2$, leading to the five configurations shown in the figure. To derive the general case, we can observe that there are k ways to add one branch to the $t(k-1)$ solutions obtained at the preceding step. Thus the following recurrence equations give the size of the constrained solution space:

$$\begin{aligned} t(3) &= 5, \\ t(k) &= k \cdot t(k-1), \qquad k > 3, \end{aligned} \tag{2.30}$$

Hence, the space of solutions for $|J_0| \geq 2^k - 1$ is given by

$$t(k) = \begin{cases} 1, 2, 5 & \text{for } k = 1, 2, 3 \text{ respectively}, \\ \frac{5}{6}\Gamma(k) = \frac{5}{6}(k+1)! & k > 3, \end{cases} \tag{2.31}$$

In Figure 2.14, we compare the sizes of the solution spaces in the unconstrained and constrained cases, respectively. The x axis gives the number of selected internal knots, whereas the y axis returns the number of possible configurations. The polyline in this example has $2^7 - 1$ internal knots, i.e. $|J_0| = 129$. The top curve represents the number of unconstrained solutions given by (2.9) and the bottom curve shows the number of constrained solutions obtained with (2.31).

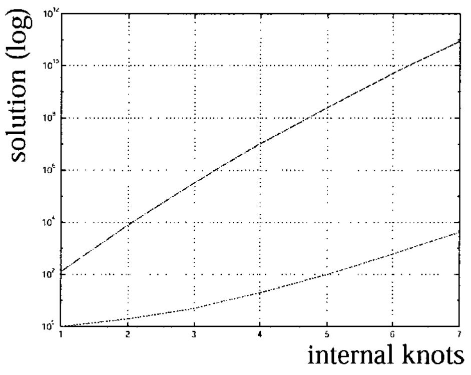


Figure 2.14: Comparison between the solution space sizes in the unconstrained (top curve) and the constrained (bottom curve) settings. The curves are given for an example polyline of 129 knots. The x axis gives the number of selected internal knots and the y axis returns the corresponding possible number of configurations.

2.5.2 Application of greedy methods

In this section, we adapt our greedy refinement and decimation methods to the constrained setting and call the algorithms `crefine` and `cdecimate` , respectively. Both schemes have linear storage complexity as in the unconstrained cases. Recall that, to fulfill the constraints, each approximating polyline must correspond to a partial tree of the full tree representing J_0. In Figures 2.15a-d, we illustrate the constrained greedy refinement method. In each figure, the tree represents the constraints on the 3 internal knots. When a knot is inserted, the corresponding node in the tree is colored in black. In Figure 2.15b, the only possible configuration using 2 segments is represented. The partial tree in this case is the singleton tree, i.e. the root node. Then in Figure 2.15c and 2.15d, knots k_1 and k_3 are successively inserted and their corresponding partial tree is represented. We evaluate now the computational complexity of the method.

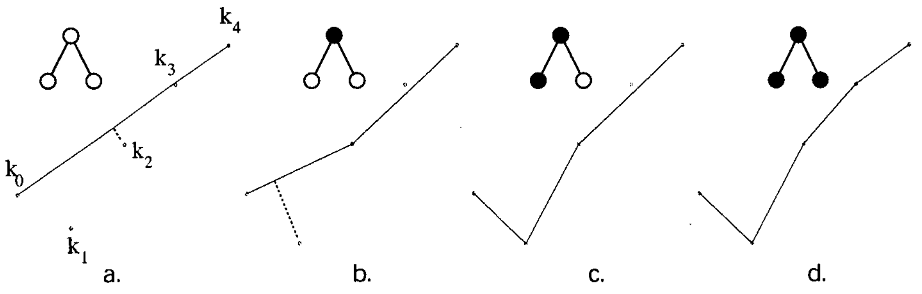


Figure 2.15: Constrained greedy refinement: In each figure, the tree represents the constraints on the 3 internal knots. When a knot is inserted, the corresponding node in the tree is colored in black. (a) The coarsest, single-segment solution. (b) At the first iteration, only knot $k_{\frac{n-1}{2}} = k_2$ can be inserted to fulfill the constraints. (c) knot k_1 and knot k_3 are successively inserted. Each approximation corresponds to a partial tree.

Computational complexity for constrained greedy refinement The analysis of the computational complexity is very similar to the one in the unconstrained case (Section 2.4.3). In the constrained case, we only need to compute two errors per step and insert them in a sorted vector, which leads to

$$\sum_{j=1}^{m/2}(2 + 2\log_2 j) = m + 2\log_2(\frac{m}{2}!), \tag{2.32}$$

then using the approximation in (2.21), we obtain a total of

$$m\log_2 m + \log_2 m - \zeta m \tag{2.33}$$

operations to perform the complete refinement in the constrained case. Consequently, `crefine` has computational complexity $\Theta(n\log n)$. The algorithm pseudo-code is similar to the one given in Appendix 2.C, with the difference that the knot selection is constrained.

In Figures 2.16a-d, we illustrate the constrained greedy decimation method. As in the insertion example, the trees in the figures represent the configurations corresponding to the approximating polylines. Figure 2.16a shows polyline J_0. In Figures 2.16b and 2.16c, knots (k_1 and k_3) corresponding to leaf nodes are decimated in order of increasing distortion. Finally in Figure 2.16d, knot k_2 (corresponding to the tree root node) can be decimated since both children have been pruned. We compute now the complexity of the algorithm.

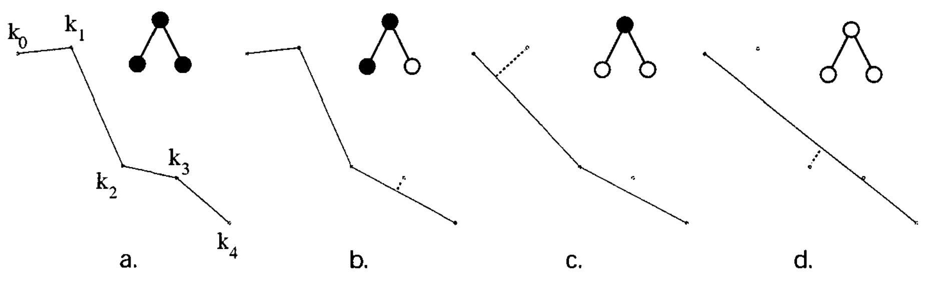


Figure 2.16: Constrained greedy decimation: In each figure, the tree represents the constraints on the 3 internal knots. When a knot is decimated, the corresponding node in the tree is colored in white. (a) The full rate polyline J_0 described by a full tree. (b) Knot k_3 and (c) knot k_1, both corresponding to a leaf in the tree are decimated. (d) Finally the last knot, corresponding to the tree root node, is removed.

Computational complexity for constrained greedy decimation Initially, the error for each knot corresponding to a leaf node must be computed and sorted, leading to

$$\frac{m}{2} + \sum_{j=1}^{m/2} \log_2 j = \frac{m}{2} + \log_2(\frac{m}{2}!) = \frac{m}{2}\log_2 m + \frac{1}{2}\log_2 m - \zeta\frac{m}{2}. \tag{2.34}$$

Because of the constraints, nonleaf nodes cannot be deleted until their two children have been removed. Then, each time a node is decimated, we must check whether its sibling is still present. When present, then no operation is further performed. Otherwise, the error for its parent in the tree is computed and sorted. Therefore, for $\frac{m}{2}$ knots, we count only one operation for the "sibling check". For the remaining knots, we must check whether their sibling is present as well. Then, we compute the errors and insert them in the sorted vector. We have then

$$\begin{aligned}\frac{m}{2} + \sum_{j=1}^{m/2-1} [2 + \log_2(\frac{m}{2} - i)] &= \frac{3}{2}m - 2 + \log_2((\frac{m}{2} - 1)!) \\ &= \frac{m}{2}\log_2(m-2) - \frac{1}{2}\log_2(m-2) + (2+\zeta)\frac{m}{2} - 2\end{aligned} \tag{2.35}$$

operations to decimate the polyline. The final complexity is obtained by summing (2.34) and (2.35), leading to

$$\frac{m}{2}\log_2(m(m-2)) + \frac{1}{2}\log_2(\frac{m}{m-2}) - 2(1+\zeta)\frac{m}{2} - 2 \tag{2.36}$$

operations for the algorithm. In conclusion `cdecimate` has computational complexity $\Theta(n \log n)$. The algorithm pseudo-code is similar to the one given in Appendix 2.D, with the difference that the knot selection is constrained.

Computational complexities and error comparison We compare the computational complexities of the constrained algorithms in Figure 2.17a. Expectedly, refinement (dashed curve) incurs less computational cost than decimation (dotted curve). We also display the computational complexities in the unconstrained cases (top solid curves). In Figure 2.17b, we compare the operational RD curves: The top (solid) curves show the errors obtained in the unconstrained cases, whereas the dashed and dotted curves show the results returned by the constrained refinement and decimation algorithms, respectively.

The obtained results are deceiving in terms of quality, which is partly due to the restriction of the solution space. The unconstrained algorithms improve the solutions by a least 15dB. In the constrained case, the approximation errors accumulated by the locally optimal choices of the greedy schemes are even more obvious, since we can clearly see the operational RD curves crossing around midrate. We will see in Section 2.5.4 that the poor performances are fortunately not only due to the solution space restriction, but also to the *local error estimate* used at each optimization step. We will explain how the results can be significantly improved by using a *global error estimate* (for the decimation scheme only) with a slight increase in computational complexity. We review now the properties of the solutions returned by the constrained algorithms.

Constrained solutions properties comparison As in the unconstrained case, the iterative process of the constrained algorithms naturally produces progressive approximations. In the multidimensional case, the hierarchy imposed over the set of knots is obtained with a subdivision scheme. In Section 2.5.3, we explain how the progressiveness is generalized to surfaces. Unfortunately, the unconstrained algorithms still do not verify the monotonicity of the operational RD curve. Again, it suffices to apply them on the function in Figure 2.5a to obtain a nonmonotonic curve.

Finally, the greedy-choice property introduced in Section 2.4.1 is again verified in the constrained scheme using the function in Figure 2.5a: The application of the refinement or the decimation scheme leads to the optimal solutions.

2.5.3 Multidimensional case: meshes obtained by subdivision

Assume that we have a set of vertices parametrized on a uniform grid of size $2^d + 1 \times 2^d + 1$, with $d > 0$, on the plane $\mathbb{R}^2$, it is possible to build a hierarchy on the set, as was done in the polyline case. The constraints are imposed using a *subdivision rule* for connecting the vertices into triangles. In the twodimensional case (as well as for some polytopes), there exists a natural

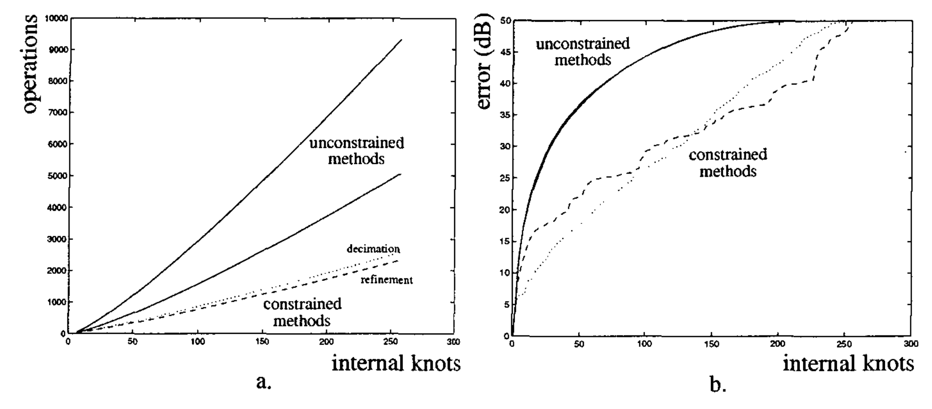


Figure 2.17: Constrained algorithm performances: (a) Computational complexities: the top (solid) curves are the complexities in the unconstrained cases, whereas the dashed and dotted curves represent the complexities for the refinement and decimation schemes, respectively. (b) Solution errors: The top (solid) curves are the operational RD curves given by the unconstrained methods. The dashed and dotted curves are returned by the refinement and the decimation algorithms, respectively.

correspondence between the vertex constraints and the quadtree[5]. We will not give more details about regular tilings here, as it will be the topic of Chapter 3.

When using a hierarchical procedure to connect the vertices in the mesh, the restricted space of vertex configurations leads to a restricted set of possible triangulations. Moreover, the successive triangulations are embedded, as depicted in Figures 2.18a-d. Figure 2.18a depicts how a set of 8 vertices is triangulated using 4-8 subdivision (Chapter 3). Figures 2.18b-d show the successive triangulations obtained when increasing the set size to 32, 128 and 256, respectively. In these figures, we depict the set of triangles obtained in Figure 2.18a. We can see that the successive approximations still contain the initial triangles. Hence, given an approximating triangulation, the next finer/coarser configuration can be obtained by splitting/merging a set of triangles.

When simplifying 4-8 meshes, e.g. using refinement or decimation, triangles are split into two smaller triangles and pairs of triangles are merged into one larger triangle, respectively. In Chapter 3, we study the properties of 4-8 meshes. In conclusion, imposing a hierarchy on the vertex set leads to a set of progressive triangulations, i.e. both vertices and their connectivity are

[5]A quadtree is a similar to the binary tree, with the difference that each node has four children instead of two.

embedded.

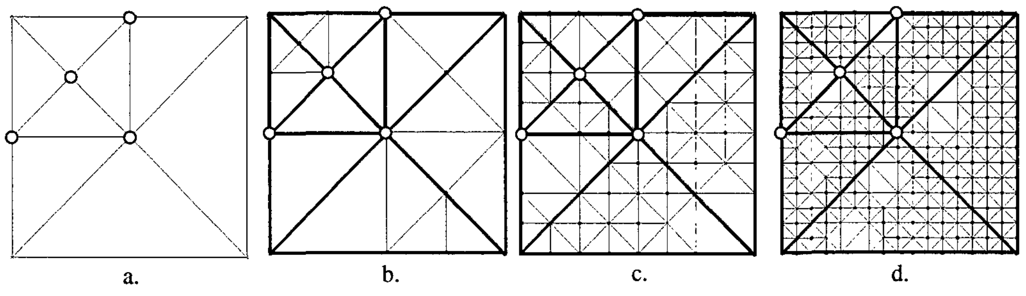


Figure 2.18: Set of progressive triangulations: The connectivity is constructed using 4-8 subdivision (Chapter 3). (a) The connectivity obtained with an initial set of 8 vertices. In the successive approximations using (b) 32, (c) 128 and (d) 256 vertices, the triangulations still contain the initial set of triangles (thick edges).

2.5.4 Using a global error estimate

We have seen that a major drawback when restricting the solution space (by imposing a hierarchy on the polyline knots) is an important decrease in solution quality (Figure 2.17b). Although it is clear that a smaller solution space degrades the performances, we have to be aware that choosing a knot to insert or to decimate simply by examining its error incurred in the current approximation is also very limiting. In the unconstrained case, basing choices on local errors only for the greedy schemes does not result in a great penalty (Figure 2.10). However in the constrained case, the effects are more important (Figure 2.17b).

A benefit of the model hierarchy is that a much more efficient error estimate can be built: Denote by J_{k_j} the set of knots corresponding to the subtree rooted at the node representing k_j in the tree. For example, as depicted in Figure 2.12, we have

$$J_{k_2} = \{k_1, k_2, k_3\}. \tag{2.37}$$

Assume now decimating all the knots in J_{k_j} and denote by $\check{J}_{k_j}$ this operation. For each decimated subtree $\check{J}_{k_j}$, we can now evaluate its distortion as

$$D(\check{J}_{k_j}) = \sum_{j \in \mathrm{j}} (k_j - \hat{k}_j)^2, \tag{2.38}$$

where the index set j identifies the knots in J_{k_j}. Then we can express $\Delta D(J_{k_i})$ as *the variation*

step	k_1	k_2	k_3	k_4
1	$\Delta D(J_{k_1})$	$\Delta D(J_{k_2})$	$\Delta D(J_{k_3})$	$\Delta D(J_{k_4})$
2	·	$\Delta D(J_{k_2}) - \Delta D(J_{k_1})$	$\Delta D(J_{k_3})$	$\Delta D(J_{k_4}) - \Delta D(J_{k_1})$
3	·	$\Delta D(J_{k_2}) - \Delta D(J_{k_1}) - \Delta D(J_{k_3})$	·	$\Delta D(J_{k_4}) - \Delta D(J_{k_1}) - \Delta D(J_{k_3})$
4	·	·	·	$\Delta D(J_{k_4}) - \Delta D(J_{k_2})$

Table 2.2: Knot errors for four steps of the decimation algorithm using a global error estimate: The error values for step 2 and 4 correspond to Figure 2.19a and 2.19b, respectively.

in distortion when all knots in J_{k_j} are decimated, hence

$$\Delta D(J_{k_j}) = D(J_{k_j}) - D(\check{J}_{k_j}), \tag{2.39}$$

where by definition $D(J_{k_j}) = 0$ and $D(\check{J}_{k_j})$ is given by (2.38). Finally, we define A_{k_j} as the set of *parents* of k_j: Assume that k_j corresponds to node index p in the tree, then A_{k_j} contains all knots corresponding to tree nodes along the path to the root. For example, we have $A_{k_1} = \{k_2, k_4\}$. We now rework the constrained greedy decimation algorithm and call `cglobal` the new algorithm. The pseudo-code will be given in Chapter 5.

Constrained greedy decimation revisited Initially, for each internal knot k_j the value $\Delta D(J_{k_j})$ is computed (step 1 in Table 2.2). Each node in the tree then records the couple

$$\{\Delta D(J_{k_j}), \Delta_{\max}\}, \tag{2.40}$$

where $\Delta_{\max}$ contains the maximal value in the subtree rooted at knot k_j. When k_j is decimated, then $\Delta_{\max} = -\infty$. At each step, the knot k^* with maximal[6] $\Delta D(J_{k_j})$ is chosen. Then, for all knots $k_j \in A_{k^*}$, the following update is performed:

$$\Delta D(J_{k_j}) - \Delta D(J_{k^*}). \tag{2.41}$$

For example, Figure 2.19a depicts the updates when decimating knot k_1 (corresponding to node 3 in the tree): The knots in A_{k_1} are encircled and their updated value is shown at step 2 in Table 2.2. At this step, the error estimate at each knot no longer represents the distortion at the knot, but the *global change in distortion* when this knot is decimated. Assume that, at step 3 knot k_3 is decimated, then the new values are again given in Table 2.2. Finally, when knot k_2 is decimated at step 4 (Figure 2.19b), then the value stored for knot k_4 (Table 2.2) is such as J_{k_2} would have been decimated entirely in a single step! This example clearly shows that, using (2.41), we obtain a global error estimate for each knot. We compute now the computational complexity of the algorithm and compare the solution errors to the ones obtained using a local error estimate.

[6]Recall that $\Delta D(J_{k_j})$ is negative by definition (2.39).

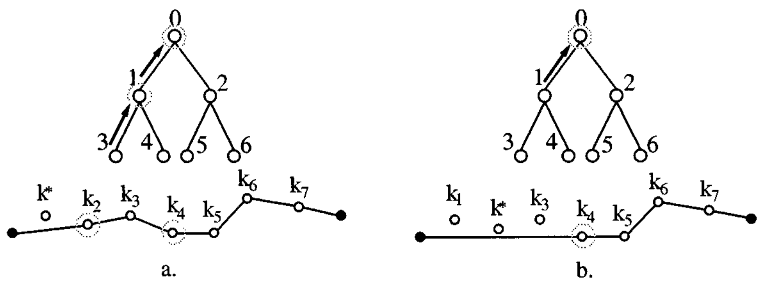


Figure 2.19: Global error estimate: (a) At step 2, k_1 is decimated and the encircled knots depict the set A_{k_1}. (b) Similarly, step 4 is represented. For each figure, the updated values for the errors are shown in Table 2.2.

Computational complexity and error comparison Initially, for each knot k_j, $\Delta D(J_{k_j})$ is evaluated and stored in the tree: Consider a polyline of length n, then the tree mapping the $m = n - 2$ interior knots has depth $d = \log_2(m+1)$. For each subtree J_{k_j}, we count the number of errors to evaluate *and* the number of additions to compute the sum. Additionally, we need $m - 1$ comparisons to retrieve the maximal values in the tree and assign $\Delta_{\max}$. Therefore, this step has cost

$$\begin{aligned}\sum_{j=0}^{d-1} 2^j(2^{d-j+1} - 3) + m - 1 &= d \cdot 2^{d+1} - 3 \cdot 2^d + 3 + m - 1, \\ &= 2(m+1)\log_2(m+1) - 2m - 1.\end{aligned} \tag{2.42}$$

During the iterated approximation process, for each node at level $d - 1 - j$, we must perform $d - j$ comparisons to get the optimal value, $2(d - j - 1)$ operations to update the parents and the values $\Delta_{\max}$. Thus, we have

$$\begin{aligned}\sum_{j=0}^{d-1} 2^{d-j-1}(3d - 3j - 2) &= 3d \cdot 2^d - 5 \cdot 2^d + 5 \\ &= 3(m+1)\log_2(m+1) - 5(m+1) + 5\end{aligned} \tag{2.43}$$

operations to perform the complete decimation of the polyline. In conclusion, the total computational cost given by summing (2.42) and (2.43), i.e.

$$5(m+1)\log_2(m+1) - 7m - 1. \tag{2.44}$$

Then using the global error estimate does not change the magnitude of the algorithm. Therefore `cglobal` has computational complexity $\Theta(n \log n)$. In Figure 2.20a, we compare again the computational complexities of our greedy set of algorithms. The complexity of `cglobal` is depicted with a dotted curve, whereas all the previous algorithms are represented with solid curves. In Figure 2.20b, we compare now the operational RD curves: The dotted curve represents the errors obtained using the global error estimate. Compared with the previous constrained versions, we measure now only a maximum distance of 3.92dB with the unconstrained algorithms. The global error estimate thus improves significantly the constrained approximation.

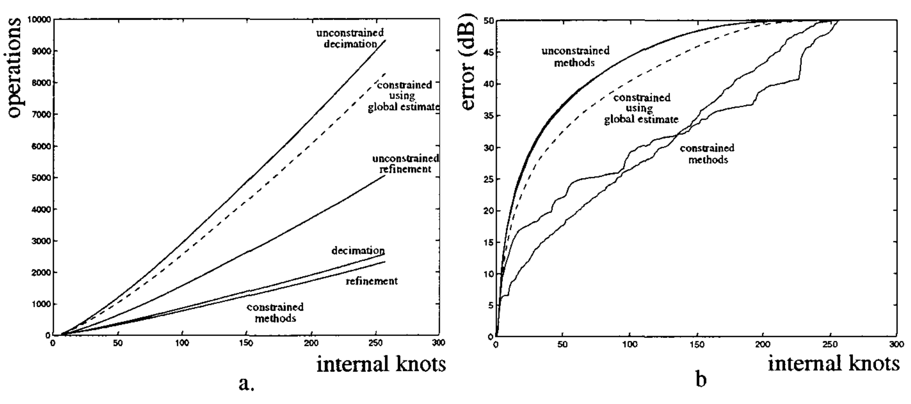


Figure 2.20: Performances using the global estimate: (a) Computational complexity and (b) solution errors. In each figure, the dotted curve depicts the performances of the new algorithm.

Solution properties Changing the error estimate does not change the properties of the solutions in this case. More precisely, the configurations are progressive and the solution errors are nonmonotonic across rates. Again, we can see that the greedy-choice property is still verified with the example in Figure 2.5a.

2.6 Optimal tree-constrained approach

2.6.1 Introduction

With this section, we arrive at the term of our exploration. We explain how the greedy decimation algorithm using our global error estimate can be further improved: In the constrained setting, we restrict ourselves to decimate knots corresponding to leaf nodes in the tree. Therefore, the rate of the polyline decreases by one segment at each iteration.

A more efficient approach is to decimate the vertex *minimizing the increase in distortion while maximizing the decrease in rate.* In our polyline problem, this corresponds to extend the decimation of single vertices (represented by leaf nodes) to a hierarchical set of vertices (represented by a subtree). Although this approach is not very common in Computer Graphics, it corresponds to well-established optimal tree pruning algorithms used in compression [31]. Our last algorithm, called `coptimal` , is inspired by an algorithm used in tree quantization called G-BFOS [9, 13]. We show that this algorithm ensures the monotonicity of solution errors. In [13], Chou *et al.* give a proof for the optimality of the algorithm.

2.6.2 Algorithm

Recall the decimation algorithm explained in Section 2.5.4 and consider now storing in each tree node the following structure:

$$\{\Delta D(J_{k_j}), \Delta C(J_{k_j}), \lambda(k_j), \lambda_{\min}\}, \tag{2.45}$$

where

$$\lambda(k_j) = -\Delta D(J_{k_j})/\Delta C(J_{k_j}) \tag{2.46}$$

and the functional $C(\cdot)$ returns the rate of the subpolyline represented by the subtree rooted at the node representing k_j. Consequently, $C(\check{J}_{k_j})$ is the rate when all knots in J_{k_j} are decimated. By definition we have

$$C(\check{J}_{k_j}) = 1, \tag{2.47}$$

i.e. all the knots are replaced with a single segment. Therefore, we can define $\Delta C(J_{k_j})$ as

$$\begin{aligned} \Delta C(J_{k_j}) &= C(J_{k_j}) - C(\check{J}_{k_j}), \\ &= C(J_{k_j}) - 1. \end{aligned} \tag{2.48}$$

The slope $\lambda(k_j)$ as defined in (2.46) represents the variation in distortion over the variation in rate when the *subtree* J_{k_j} is decimated. Finally, $\lambda_{\min}$ contains the minimal lambda in the subtree, i.e. we do not restrict the minimal value to be located in the leaves as in the previous algorithm.

The algorithm operates in the operational RD plane. Each point $(C(J_{k_j}), D(J_{k_j}))$ represents the knot configuration in the plane. The configuration is obtained by decimating subtree J_{k_j} (Figure 2.21). We aim at finding the points lower-bounding the convex hull of all configurations. These points form the optimal operational RD curve, as depicted in the figure.

Initially, we compute for each node the values $\Delta D(J_{k_j})$, $\Delta C(J_{k_j})$ and $\lambda(k_j)$ and store the minimal value for each subtree in $\lambda_{\min}$. For knots k_j's, the corresponding slopes are represented in Figure 2.22a. Then the algorithm is iterated as follows: The node k^* with minimal slope in

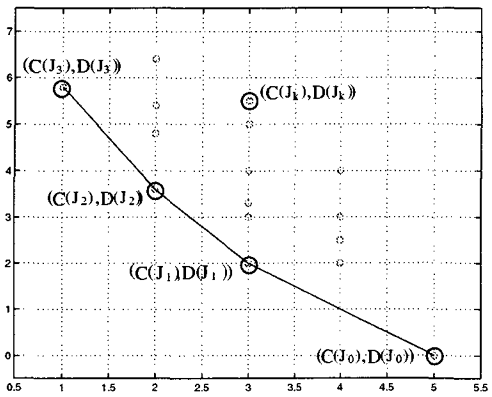


Figure 2.21: Configurations in the RD plane: Each point $(C(J_{k_j}), D(J_{k_j}))$ represents the configuration obtained by decimating k_j.

the RD plane is chosen, i.e.

$$k^\star = \arg\min_j \lambda(k_j), \tag{2.49}$$

and the corresponding set of knots J_{k_j} is decimated (Figure 2.22b). Then, we update the set of parents $A_{k^\star}$ such as $\forall k_j \in A_{k^\star}$, we do

$$\begin{aligned} &\Delta D(J_{k_j}) - \Delta D(J_{k^\star}), \\ &\Delta C(J_{k_j}) - \Delta C(J_{k^\star}). \end{aligned} \tag{2.50}$$

Finally, the slope $\lambda(k_j)$ and the value $\lambda_{\min}$ for the subtree are recomputed (Figure 2.22c). We iterate the algorithm until $k^\star$ corresponds to the root node.

Computational complexity and error The computational complexity of the algorithm is very similar to the one of its greedy counterpart (Section 2.5.4). However, we also need to compute $\Delta C(J_{k_j})$ and the slopes $\lambda(k_j)$. The first value is computed in closed-form, i.e.

$$\Delta C(J_{k_j}) = |J_{k_j}| - 2. \tag{2.51}$$

Additionally, we account one additional operation per slope. Thus, we add $\sum_{j=0}^{d-1} 2^{i+1}$ operations to (2.42), leading to a total of

$$2(m+1)\log_2(m+1) - 1. \tag{2.52}$$

Given m internal knots, we must perform m iterations in the worst case (when only leaf nodes achieve minimal λ). Therefore, for each node at level $d-1-j$, we still must do $d-j$ comparisons.

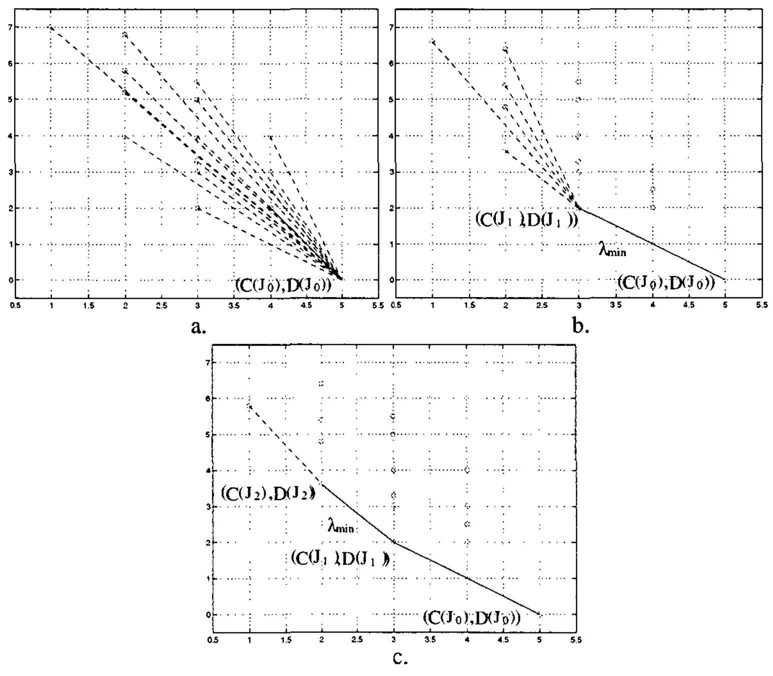


Figure 2.22: Iteration of the optimal constrained algorithm.

However, we need $4(d-j-1)$ operations to update the parents, compute the slopes and assign $\lambda_{\min}$. We have then

$$\begin{aligned}\sum_{j=0}^{d-1} 2^{d-j-1}(5d-5j-2) &= 5d\cdot 2^d - 7\cdot 2^d + 7, \\ &= 5(m+1)\log_2(m+1) - 7m\end{aligned} \tag{2.53}$$

operations to perform the optimal decimation in the worst case. The total cost of the algorithm is obtained summing (2.52) and (2.53), i.e.

$$7(m+1)\log_2(m+1) - 7m - 1. \tag{2.54}$$

Therefore, `coptimal` has computational complexity $\Theta(n\log n)$. The optimal algorithm only provides a weak improvement in terms of solution errors over its greedy counterpart (an average gain of 0.5 dB is measured). However, the variable-rate decimation scheme provides error monotonicity across rates, as shown in the next section.

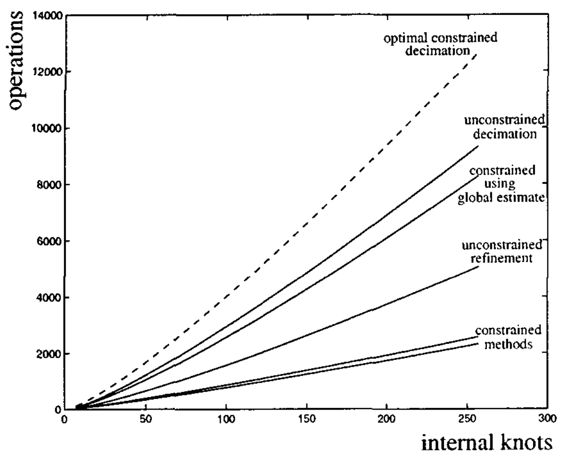


Figure 2.23: Computational complexity comparison: The dotted curve represents the complexity of the optimal constrained algorithm, whereas the solid curves depict the complexities of the remaining greedy algorithms.

Solution properties The main difference between `coptimal` and its greedy counterpart is that it can perform *variable-rate decimation*, i.e. we are not restricted to decimate knots corresponding to leaf nodes. It is clear that all the properties verified by the greedy version also hold for the optimal algorithm. Additionally, we show now with an example how the optimal algorithm also verifies the monotonicity of the solution errors. We will give a formal proof for it in Chapter 5.

Example 2.1

We apply now the optimal algorithm on the function in Figure 2.5a. Recall that all the previous algorithms returned a nonmonotonic operational RD curve with this function. At the initialization step, the slope $\lambda(k_j)$ for each k_j is computed (Figure 2.24a). According to table 2.1, We obtain the following values:

$$\lambda(k_1) = -\tfrac{-4}{1} = 4, \quad \lambda(k_2) = -\tfrac{-6}{3} = 2, \quad \lambda(k_3) = -\tfrac{-4}{1} = 4. \qquad (2.55)$$

The knot corresponding to the minimal λ is chosen, yielding knot k_2. Then, the optimal move in the RD plane with this function is to decimate all interior knots. The decrease in rate is 3 and the optimal operational RD curve is depicted in Figure 2.24b.

The above example illustrates well that the algorithm uses a variable-rate decimation scheme

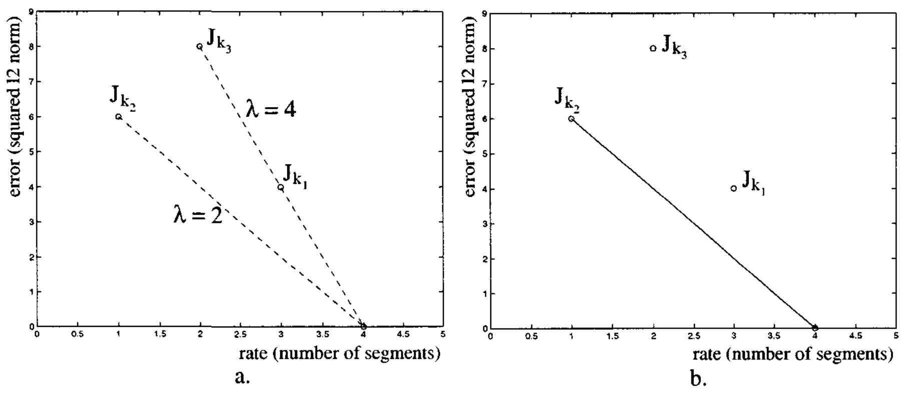


Figure 2.24: Application of the optimal constrained algorithm to the function in Figures 2.5a: (a) At the initialization, the slope $\lambda(k_j)$ for each knot is computed. (b) The optimal decimation is obtained with the knot corresponding to the smallest slope. Hence, we decimate J_{k_2}, i.e. all the interior knots in this case and the nonmonotonicity is avoided.

to avoid the nonmonotonicity of the operational RD curve. Since the decrease in rate is not constant as all the previous scheme, typically a sparser range of rates is available.

2.7 Review of algorithm properties

We provided a detailed analysis for the computational cost of the algorithms in this chapter. In order to give a clearer summary for the computational complexities, we examine again each algorithm cost having the same asymptotic behavior and derive a factor for their highest magnitude term: For `urefine` , the cost (2.22) yields

$$3m\log_2 m - m(3+\zeta) + 3\log_2 m + 2 \in \Theta(3m\log m). \tag{2.56}$$

For `udecimate` , the cost (2.28) yields

$$\begin{aligned} &m\log_2(m(m-1)^4) + (3-5\zeta)m + \log_2\left(\frac{\sqrt{m}}{(m-1)^2}\right) - 2 \leq \\ &\alpha m\log_2(m^5) = 5\alpha m\log_2(m) \in \Theta(5m\log m), \end{aligned} \tag{2.57}$$

where $\alpha > 0$ is a sufficiently large factor. For `crefine` , the cost (2.33) yields

$$m\log_2 m + \log_2 m - \zeta m \in \Theta(m\log m). \tag{2.58}$$

algorithm	*computational cost*	*storage cost*	*progressiveness*	*monotonicity*
uoptimal	$\Theta(n^3)$	$\Theta(n^2)$	·	·
urefine	$\Theta(3n\log n)$	$\Theta(n)$	√	·
udecimate	$\Theta(5n\log n)$	$\Theta(n)$	√	·
crefine	$\Theta(n\log n)$	$\Theta(n)$	√	·
cdecimate	$\Theta(n\log n)$	$\Theta(n)$	√	·
cglobal	$\Theta(5n\log n)$	$\Theta(n)$	√	·
coptimal	$\Theta(7n\log n)$	$\Theta(n)$	√	√

Table 2.3: Summary of algorithm properties.

For cdecimate , the cost (2.36) yields

$$\begin{aligned}&\frac{m}{2}\log_2(m(m-2)) + \frac{1}{2}\log_2(\frac{m}{m-2}) - 2(1+\zeta)\frac{m}{2} - 2 \leq \\ &\alpha\frac{m}{2}\log_2(m^2) = \alpha m\log_2(m) \in \Theta(m\log m),\end{aligned} \tag{2.59}$$

where $\alpha > 0$ is a sufficiently large factor. For cglobal , the cost (2.44) yields

$$5(m+1)\log_2(m+1) - 7m - 1 \in \Theta(5m\log m). \tag{2.60}$$

Finally, for coptimal the cost (2.54) yields

$$7(m+1)\log_2(m+1) - 7m - 1 \in \Theta(7m\log m). \tag{2.61}$$

The obtained factors are confirmed by Figure 2.23. Table 2.3 summarizes the properties of our set of algorithms. Note that we express the complexity of the greedy algorithms in terms of $n = m - 2$.

2.8 Summary

The simplicity of the polyline model permitted us to lead a full analysis and comparison of the main algorithmic approaches in order to solve the approximation problem (our results are summarized in Table 2.3). We explored a set of unconstrained and constrained algorithms and gave a detailed analysis for each of them in terms of computational complexity and solution errors.

In our set of algorithms, cglobal (Section 2.6) is the most appealing to our transmission problem. Consequently, the rest of this thesis mainly consists in the generalization to surface models of this approach. Our plan for this task has several facets, which can be described by the following observations:

- The simple connectivity of the polyline model eased our investigation. The multidimensional case raises greater issues, as explained in Chapter 3.
- Mesh connectivity is more complex. Hence, we need an efficient data structure to access the dataset. Once again, the polyline model freed us from this aspect. An efficient storage mechanism is the topic of Chapter 4.
- In this chapter, all the algorithm optimizations are based on a single criterion: *the coordinates of the knots*. This value can be seen as a property of the knot. Consequently, we can think about having a set of properties and perform joint optimizations on multiple criteria. In Chapter 5, we generalize `cglobal` to surfaces. Also, we explain how approximations based on multiple properties is performed. Chapter 6 gives an example of such a joint optimization scheme.

With the chapter, we depicted the panorama of mesh approximation using a monodimensional "projection" of the problem. The rest of this thesis is led in the multidimensional case.

Appendix 2.A Computational complexity model and notations

2.A.1 Random Access Machine (RAM) model

To define the unit of time required to evaluate the computational complexity of an algorithm, we need to choose a *model of computation.* A possible choice is the *random access machine* (RAM) model [82]. The RAM is a register-based model of computation that can access any of its registers in unit time (an idealization of the modern digital electronic computer). The characteristics of this model are:

- Each memory unit holds a real number and its access has constant cost.
- The elementary operations are the *comparison of two real numbers* and any mathematical operations.

All asymptotic bounds are given for this model.

2.A.2 Notations for computational complexity

Consider a function $f(n)$ returning the number of necessary operations for a particular algorithm to solve a problem of size n in our model of computation, the sets O, Ω and Θ are defined as follows:

Definition 2.1 (Computational Complexity Classification)
The computational cost $f(n) \in O(g(n))$ for an input size n when we can find $g(n)$ and sufficiently large $c > 0$ and $n > n_0$ such that $f(n)$ is upper bounded by $g(n)$, i.e.

$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \text{ s.t } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}. \quad (2.62)$$

The computational cost $f(n) \in \Omega(g(n))$ for an input size n when we can find $g(n)$ and sufficiently large $c > 0$ and $n > n_0$ such that $f(n)$ is lower bounded by $g(n)$, i.e.

$$\Omega(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \text{ s.t } 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}. \quad (2.63)$$

The computational cost $f(n) \in \Theta(g(n))$ for an input size n when we can find $g(n)$ and sufficiently large $c_1, c_2 > 0$,$c_1 < c_2$ and $n > n_0$ such that $f(n)$ is jointly lower bounded by $c_1 g(n)$ and upper bounded by $c_1 g(n)$, i.e.

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ s.t } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}. \quad (2.64)$$

The function sets in Definition (2.1) characterize the *asymptotic behavior* of the algorithm, as shown in the following example: Let $f(n)$ be the computational complexity of an algorithm in the worst case. We are interested in the behavior of $f(n)$ when $n \to \infty$. Obviously, we are interested in the algorithms whose asymptotic analysis yields a complexity with the smallest order of magnitude. However, assuming two algorithms A and B having the respective complexities $f_{\mathsf{A}}(n)$ and $f_{\mathsf{B}}(n)$ and $f_{\mathsf{A}}(n) \in O(f_{\mathsf{B}}(n))$. Considering Definition (2.1), it may be possible that, for some input size n, B outperforms A. Often, a small order of magnitude represents a particularly interesting characteristic for an algorithm, however we need to be aware of the shortcomings of such limited view. When neither the constant c nor n_0 are specified in (2.1), we must keep in mind that the complexity bound is effective only for sufficiently large inputs.

2.A.3 Computational optimality

The *complexity or size* of a problem P is defined as the minimal number of elementary operations needed by any algorithm to solve the problem. Consequently, an algorithm is *computationally optimal* if its complexity has the same order of magnitude as the problem size. The optimality has an *asymptotic property* of the algorithm, as stated in the definition below:

Definition 2.2 (Computational Optimality)
Given a problem P *of size* $g(n)$, *a computationally optimal algorithm having complexity* $f(n)$ *satisfies* $f(n) = O(g(n))$.

Following Definition (2.2), we see that for any algorithm A solving P, we have

$$f(n) = \Omega(g(n)). \tag{2.65}$$

Two natural lower bounds for $g(n)$ are n and the output size $s(n)$, then

$$g(n) = \Omega(\max(n, s(n))). \tag{2.66}$$

Then by definition, the complexity of any algorithm solving P is an upper bound for $g(n)$, or $g(n) = O(f(n))$. Finally, if an optimal algorithm exists, then

$$g(n) = \Theta(\max(n, s(n))). \tag{2.67}$$

Appendix 2.B Pseudo-code for `uoptimal`

We give here the pseudo-code for the unconstrained optimal algorithm using dynamic programming. We have the following notations:

- $\{k_l, k_j\}$ denotes the segment between knots k_l and k_j and $||\{k_l, k_j\} - J_{j+1,j}||_2^2$ is the distance in squared l_2 norm between the segment and the full rate polyline computed only on the internal knots between k_l and k_j.

ALGORITHM

```
for $j = 0..n-2$ do
    for $l = 0..j$ do
        $D(e_l) = ||\{k_l, k_j\} - J_{j+1,j}||_2^2$
    end
    for $i = 0..j$ do
        $J_{i+1,j} \leftarrow \arg\min_l (D(J_{i,l<j}) + D(e_l))$
    end
end
```

Appendix 2.C Pseudo-code for `urefine`

We give here the pseudo-code for the unconstrained greedy refinement algorithm. We have the following notations:

- $k_{\max}$ is the knot incurring the highest distortion.
- $D(k_i)$ is the distortion measured only at knot k_i.
- Given a knot $k_{\max}$, we consider two particular knots k_l and k_r representing the closest inserted knot to the left and right of $k_{\max}$ respectively. In order words, the first knots connecting the solution to the left and to the right and $k_{\max}$.
- The operator $\oplus$ denotes the insertion of knot in a configuration J_i.
- The operator $\leftarrow$ in $E \leftarrow \arg D(k_i)$ denotes the insertion of knot k_i in the set E sorted using the distortion $D(k_i)$.

ALGORITHM

```
init:
$D(k_{\max}) = -\infty$
for $j = 0..n-2$ do
    if $D(k_j) > D(k_{\max})$ do
        $k_{\max} = \arg D(k_j)$
    end
end
$J_{n-3} = J_{n-2}$
algorithm:
for $j = 0..n-2$ do
    if $(l < \max -1)$ do
        $E \leftarrow \arg\max\{D(k_{l+1}), \ldots D(k_{\max -1})\}$
    end
    if $(r > \max +1)$ do
        $E \leftarrow \arg\max\{D(k_{\max +1}), \ldots D(k_{r-1})\}$
        end
        $k_{\max} = \max E$
        $J_{n-(2+j)} \leftarrow J_{n-(1+j)} \oplus k_{\max}$
    end
```

Appendix 2.D Pseudo-code for `udecimate`

We give here the pseudo-code for the unconstrained greedy refinement algorithm. We have the following notations:

- k_{min} is the knot incurring the minimal distortion.
- $D(k_i)$ is the distortion measured only at knot k_i.
- Given a knot k_{min}, we consider two particular knots k_l and k_r representing the closest inserted knot to the left and right of k_{max} respectively. In order words, the first knots connecting the solution to the left and to the right and k_{max}.
- The operator $\setminus$ denotes the decimation of knot in a configuration J_i.
- The operators $\leftarrow$ and $\leftrightarrow$ denote the insertion and the replacement of knot k_i in a set of knots E sorted using distortion $D(k_i)$, respectively.

ALGORITHM

```
init:
for $j = 1..n-2$ do
    $E \leftarrow \arg D(k_j)$
end
for $j = 1..n-2$ do
    $k_{\text{min}} = \min E$
    $J_j \leftarrow J_{j-1} \setminus k_{\text{min}}$
    $E \leftrightarrow \arg D(k_l)$
    $E \leftrightarrow \arg D(k_r)$
end
```

Chapter 3

Computational analysis of 4-8 meshes

3.1 Introduction

3.1.1 Motivation

Meshes with *subdivision connectivity*, i.e. regular triangulations constructed using iterated subdivision rules, are popular in many applications, such as visualization [19] and finite element analysis [15], to name a few. Their irregular counterparts have also been extensively studied [49], but regular meshes are preferred because of their superior performance and flexibility for processing [35], transmission [52] and compression [48].

A particular class of regular triangulations are *4-8 meshes.* These meshes have been extensively used to visualize terrain data [4, 19, 58, 71]. In this context, 4-8 meshes are also called *quadtree triangulations* because quadtrees are often used to store them [55, 71, 76]. Terrain models are given as amplitude matrices (i.e. the parametrization is implicit) and 4-8 meshes are used to *connect* the vertices (Figures 3.1a-e). Recently, researchers have also used 4-8 meshes to compute approximations of subdivision surfaces [87]. Subdivision surfaces are an increasingly popular representation for piecewise-smooth surfaces. Algorithms for subdivisions surfaces use recursive subdivision rules to *create* vertices from a coarse control mesh. Examples of such rules are provided by Loop [59], Catmull-Clark [11, 14] and Velho [87]. Today, the properties of subdivision surfaces are an important area of investigation (e.g. [93]).

In both terrain visualization and subdivision surfaces, researchers often deal with large datasets. Therefore, *simplification algorithms* producing adaptive, multi-resolution representations are an important topic of investigation [19, 58, 71, 4]. Multi-resolution representations of meshes with subdivision connectivity have many advantages over their uniform counterparts. They allow for vertices to be concentrated in detailed regions, leading to efficient descriptions of the shape. Their multiple levels of resolution provide an efficient means to deal with resources-constrained rendering, storage or transmission. However, adaptivity comes at a price: These representations are more complex to process and to store. Also, basic operations such as vertex queries are more difficult to implement. In the next section, we review previous work on simplification algorithms for 4-8 meshes.

The properties of 4-8 meshes analyzed in this chapter derive from the vertex hierarchy and the particular connectivity of the vertex set. Hence, an analysis of the mesh as a tiling of $\mathbb{R}^2$ is sufficient, and the results are also valid for a set of connected vertices in $\mathbb{R}^3$. The amplitudes z of the underlying matrix are important, however, when simplification errors are computed for vertices, i.e. in simplification algorithms. The analysis is led using triangulated quads similar to those shown in Figures 3.1a-e. However, our results also apply to meshes approximating subdivision surfaces in $\mathbb{R}^3$ (as in [87]), except at extraordinary vertices forming the coarse control mesh. Recall that these vertices are similar to the ones used to connect the initial square of two triangles in Figure 3.1a.

3.1.2 Previous work on simplification algorithms

Simplification algorithms, leading to multi-resolution representations, aim to select a subset of the original vertices in order to efficiently represent the shape. We consider approaches either based on vertex decimation (e.g. when starting from a dense mesh) or vertex insertion (e.g. when starting from a coarse version of the mesh). Note that alternative approaches exist (e.g. edge split, edge collapse) and are also investigated. In Section 3.2.3, we explain that a hierarchy (very similar to the hierarchy in a tree structure) is imposed over the vertices by the 4-8 connection (Figure 3.1a-e). Hence, decimating/inserting an arbitrary vertex often implies jointly decimating/inserting additional vertices to preserve the hierarchy (think of pruning a tree node and all its descendants, or inserting a leaf node and all its parents). For any vertex, its set of descendants is called its *merging domain* and its set of parents is called its *splitting domain*. An advantage for preserving the hierarchy is that a global error measure can be computed in order to simplify the mesh. This characteristics will be exploited in Chapter 5.

Many simplification algorithms for 4-8 meshes based on decimation or insertion have been given in the context of terrain visualization [19, 58, 71]. However, these methods are based on insertion only [58, 71] or on restricted cases of decimation [19] (see below). Also, all previous algorithms have used local error metrics. For subdivision surfaces, most implementations are based on nonadaptive representations to avoid the added complexity and performance penalty traditionally associated with adaptive schemes. When simplifying a mesh, an error criterion is used to select vertices to insert or decimate. For example, an error can be computed at each vertex according to local variations in curvature over its merging domain or splitting domain, depending on if the aim is to decimate or insert the vertex, respectively. Therefore each simplification step modifies the model's shape, and some errors must be recomputed. In previous works, algorithms were given in order to recompute errors after a vertex insertion [58, 71] or restricted decimation [19]. However, no such algorithm is described in the general case of decimation[1].

Overall, no computational analysis of common 4-8 mesh operations (e.g. decimation, insertion, update of the modified errors) is available to the authors' knowledge. A detailed analysis of 4-8 mesh properties is useful in many aspects: It provides tools to design algorithms and forecast their cost. It also allows for more elaborated error metrics to be built, improving algorithm performances. As explained above, simplification algorithms reduce the number of vertices whereas algorithms for subdivision surfaces generate vertices using subdivision rules. Hence an analysis of 4-8 mesh properties helps provide a unifying framework for simplification and subdivision methods. In the next section, we explain our contributions and the organization of this chapter.

3.1.3 Contributions and plan

In this chapter, we present fundamental computational complexity results for processing 4-8 meshes. We summarize our contributions below and refer to the corresponding sections.

Computational analysis of mesh operations We compute the number of operations required to decimate and insert an arbitrary vertex in the hierarchy and show that, on average, these operations can be performed in $\Theta(\log n)$ time (Section 3.3.1). Call *ancestors* the vertices whose domain connectivity is modified after a decimation or an insertion. We explain how to find these vertices and show that $\Theta(\log n)$ exist in each case (Section 3.3.2).

Merging domain intersections We explain that an interesting problem is to determine *which* are the removed vertices in the merging domain of ancestors after decimating a vertex. This problem requires the computation of *merging domain intersections*, and its solution is the most important contribution in this chapter. The solution to this problem enables the building of a global error metric for algorithms using general decimation. We will present such an algorithm in Chapter 5. We explain how to describe merging domain intersections (Section 3.4.1), and we provide a model for the problem (Section 3.4.2). We compute its cost in closed form (Section 3.4.3) and show that, on average, $\Theta(\log^2 n)$ operations are sufficient to compute the intersections between the merging domain of a decimated vertex and the merging domains of all its ancestors (Section 3.4.4).

Computational lower bounds for algorithms using local error Consider simplification algorithms based on local error using either general decimation or insertion. We use our

[1]We explain the difference between restricted and general decimation in Section 3.2.3.

results to prove computational lower bounds for their execution (Section 3.5). More precisely, we show that $\Theta(n \log n)$ operations are necessary to fully decompose or refine a surface.

3.2 4-8 mesh construction

3.2.1 Connecting a matrix of amplitudes

We denote by M a mesh having a connectivity obtained by recursive 4-8 connection. Since the connectivity is regular, it is sufficient to represent M as a set of vertices, hence $M = \{v_0, \ldots, v_{N-1}\}$. We present a simple construction of a 4-8 mesh connecting an amplitude matrix z (e.g. terrain data), i.e. the coordinates x, y are implicit. For the sake of clarity, we represent our meshes as tilings of the plane $\mathbb{R}^2$. A 4-8 mesh connecting the dataset is created using the recursive procedure depicted in Figures 3.1a-e. Initially, a quadrilateral, or more simply quad, formed with two triangles is connected using the four corner vertices. Then, each triangle hypothenuse is bisected to connect a vertex at the midpoint. We denote each connection step by l and Figures 3.1b-e depict steps $l = 1, 2, 3, 4$, respectively. After $l = 2d$ connection steps, the mesh connects a matrix of amplitudes $2^d + 1$ and contains $n = 2 \cdot 4^d$ triangles. The unique vertex inserted at step $l = 1$ (Figure 3.1b) is called the *root vertex* and is denoted by v_0.

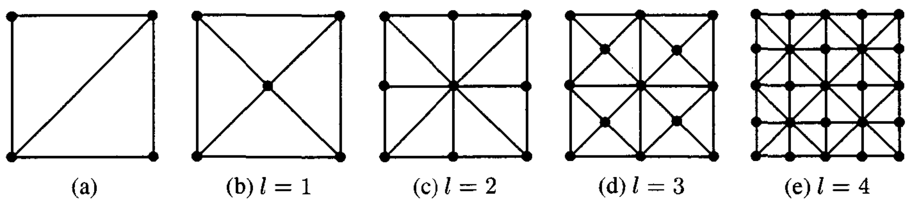


Figure 3.1: Connection of a matrix of amplitudes z using the 4-8 scheme: (a) Initially, a square formed by two triangles is created using the corner vertices. Then, triangle hypothenuses are bisected to connect a vertex at the midpoint. Each connection step is denoted by l and (b),(c),(d) and (e) shows steps $l = 1, 2, 3$ and 4, respectively.

The 4-8 scheme takes its name from an instance of regular tilings studied by Laves in Cristallography [54]. A 4-8 mesh corresponds to a $[4 \cdot 8^2]$ tiling. The notation suggests that each triangle has one vertex of valence 4 and two vertices of valence 8. In Figures 3.1a-e, this is verified at even steps l.

The successive meshes generated by recursive connection are embedded, in the sense that we can obtain any mesh from a coarser approximation by simply splitting some of the triangles, which leads to the following definition:

Definition 3.1 (Embedding)
Given M_i and M_j, we say that M_j is embedded in M_i iff $|M_j| < |M_i|$ and $M_j \subset M_i$. Note that when $M_i \equiv M_0$, the inclusion is not necessary.

We need now to explain how the operator $\subset$ can be used to compare two meshes: In the case of a matrix of amplitudes z, the mesh can be simply defined as a set of vertices (x, y, z) and therefore be freed from specifying explicitly the connectivity. An alternate definition is to use only the z coordinate and specify the connectivity. This solution allows to define each vertex with a single floating-point value (i.e. the z coordinate). In this case, the definition is a particular case of [36]. Both approaches lead to the same semantic for M and allow for the comparison of two meshes using the inclusion operator $\subset$. Having a mesh freed of parametric information, i.e. the coordinates (x, y) of the vertices, leads to more efficient storage.

Considering a set of N vertices given as a matrix of amplitudes z and $\sqrt{N} = 2^d + 1$, the 4-8 connection induces a natural hierarchy on the vertices in M. We illustrate it by splitting the regular grid Ω into the so-called *quincunx* and *Cartesian* grids (Figures 3.2a and 3.2b). The sets are denoted by Q and C respectively. Since two connection steps are necessary for the mesh to connect a uniform matrix, we have $d = \frac{l}{2}$ with l even. The total size of each grid after $d \geq 0$ connection steps is respectively given by

$$|Q|^d = \frac{1}{6}4^{d+1} + 2^{d+1} - \frac{8}{3} \quad \text{and} \quad |C|^d = \frac{1}{12}4^{d+1} - \frac{1}{3}. \tag{3.1}$$

In Figures 3.2a and 3.2b, the index next to a vertex in the grid indicates at which connection step this vertex is connected. Note that the four vertices with $l = 0$, forming the two initial triangles (Figure 3.1a), are actually not connected during the recursive connection since they are used to construct the initial base mesh. Thus we have $|M|^d = |Q|^d + |C|^d + 4$, where d denotes the number of connection steps. Finally, note that the quincunx grid is just a rotation of the Cartesian one by $\frac{\pi}{4}$ and their superposition results in the uniform matrix Ω (Figure 3.2c).

3.2.2 Arbitrary topologies and subdivision surfaces

The construction presented in the previous section is useful to create representations of parametric surfaces such as terrains. To represent more involved topologies, two approaches are possible: the connection scheme depicted in Figures 3.1a-e may be used to connect vertices parametrized on a regular polyhedron. A regular polyhedron has equal faces whereas its vertices *are all surrounded alike*. Call p the number of vertices connecting each face and q the number of faces surrounding each vertex, then only 5 possible couples (p, q) exist. This set of polyhedra is known as *platonic solids*. Only 3 of them have triangular faces and are depicted in Figure 3.3. For each of these polyhedra, 4-8 connection can be applied. A straightforward application is the representation of spherical datasets since the subdivisions of a triangular platonic solid can be used to obtain a tessellation of the sphere.

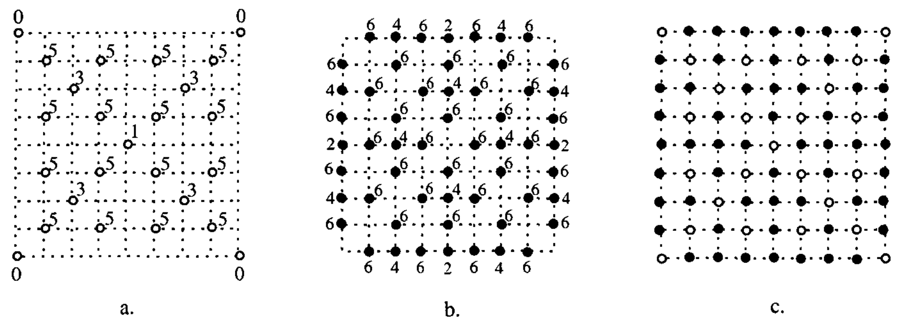


Figure 3.2: Vertex hierarchy: (a) Cartesian grid. (b) Quincunx grid. (c) Superposition of both grids. In (a) and (b) the index next to each vertex corresponds to the connection step at which the vertex is connected.

Subdivision surfaces are used to generate 4-8 meshes with arbitrary topology [87]. A coarse *control mesh* composed of a small set of triangulated quads (as in Figure 3.1a) fixes the topology and is used as an initial mesh. The vertices connecting the quad forming the control mesh are often called *extraordinary vertices* since their valence might be arbitrary. Subdivision rules are used to create new vertices connected on each quad, as in Figures 3.1b-e.

3.2.3 Constraints when simplifying 4-8 meshes

The iterative procedure used to connect the vertices imposes *hierarchical constraints* over the set of vertices. The hierarchical structure is fixed by the connection step l assigned to the vertices. When computing an adaptive representation of the mesh (e.g using decimation or refinement), the hierarchy between vertices must be preserved.

For example, consider the root vertex connected in Figure 3.1b. A decimation *perserving the hierarchy* operates as follows: When this vertex is decimated (e.g. in the mesh of Figure 3.1e), the edge orginally split by the vertex (the diagonal in Figure 3.1a) must be recovered. Consequently, all the vertices in the mesh are also decimated. Call v a vertex, then M_v denotes the set of vertices that must be removed jointly in order to recover the original edge and thus preserve the hierarchy. We call the set M_v *merging domain*. A merging domain is attached to each vertex in the mesh. For the vertices v connected at the step depicted in Figure 3.1e, $M_v = \{v\}$ since it suffices to remove v to recover the corresponding edge in Figure 3.1d. We refer to such decimation as a *restricted* case of decimation (as used in [19]). In contrast, a general case decimation refers to the removal of an arbitrary vertex (i.e. with $|M_v| > 1$).

We look now at the hierarchical constraints incurred when decimating and inserting a vertex

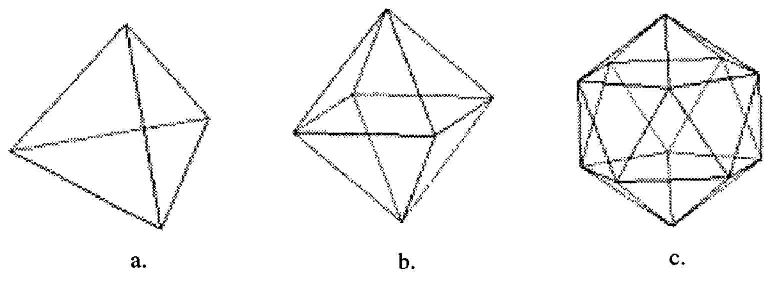


Figure 3.3: The three platonic solids having triangular faces: (a) the tetrahedron, (b) the octahedron, (c) the icosahedron.

in more detail. Consider the example in Figure 3.4a, and assume that v is connected at step l. Then the vertices in M_v connected at steps $l + 1$, $l + 2$ and $l + 3$ are labeled with the index 1, 2 and 3, respectively. The set M_v is decimated as follows: First, v is decimated. Then we remove the four vertices on the edges of the triangulated quad split by v (vertices with index 1). These vertices are called *descendants* of v, and we explain how to find them in Section 3.3.2. Then for each of these vertices, we decimate their four descendants (vertices with index 2). Note that the descendants split the four quads at the next level surrounding each edge of the previous quad. We repeat the procedure until the descendants connected at the last step are reached (step $l + 3$ in the example). The resulting mesh is shown in Figure 3.4b. The set of triangles tiling the merging domain in the figure is called *support*.

We also attach to each vertex v a *splitting domain*, denoted by S_v. The domain contains the vertices to insert jointly to v in order to preserve the hierarchy. Figure 3.4c shows an example of insertion: The vertex v lies at a location connected at step $l = 5$. An insertion preserving the hierarchy operates as follows: ·First, we connect v to its *parents*[2] at step $l = 4$. Then, each of these vertices is connected to its parents at step $l = 3$ and so on. We repeat until no more vertices need to be inserted. At most this bottom-up traversal is stopped at the root vertex. The underlying grid in the figure represents the parametrization of a 9×9 matrix of amplitudes.

Preserving the hierarchy has two consequences for simplified meshes: First, it ensures that successive decimations/insertions yield a set of embedded meshes. For example, consider a coarse mesh obtained after *a series of decimations*. If the series of meshes is embedded (Definition 3.1), it is always possible to reconstruct a mesh from a coarser version *only* by splitting a set of triangles. Second, the resulting mesh is *conforming* [15], i.e. no triangle has a vertex of another triangle in the interior of one of its edges. This condition must be fulfilled in order to

[2]We explain how to find the parents in Section 3.3.2.

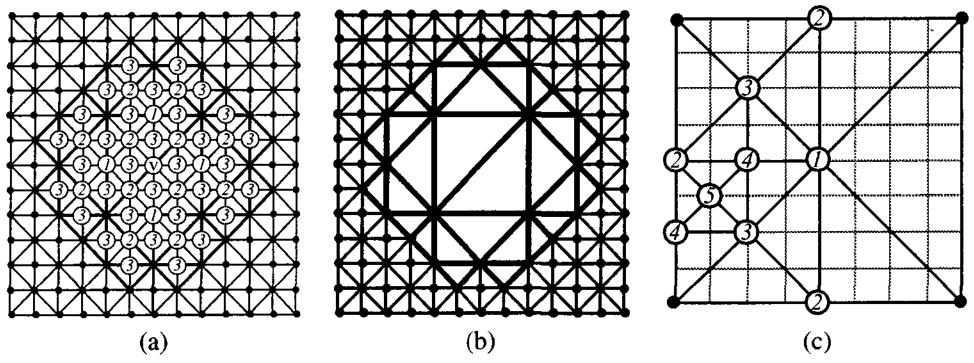


Figure 3.4: Mesh operations: (a) The white vertices represent the set M_v. (b) Support of the merging domain (set of triangles tiling the domain after that M_v is decimated). (c) Insertion of a vertex v at a location connected at step $l = 5$. The white vertices form the splitting domain and the numbers correspond to the connection steps. The underlying grid depicts the parametrization of a 9×9 matrix of amplitudes.

render the surface without cracks, i.e. to avoid shape discontinuities.

3.3 Analysis of simplification operations

3.3.1 Decimation and insertion of a vertex

We first evaluate the cost of decimating a vertex. To do so, we compute the number of triangles connected with at least one vertex in M_v. This number is linearly proportional to the number of vertices in M_v (see below). We address two cases: (1) before the decimation of M_v (e.g. Figure 3.4a), and (2) after the decimation of all vertices in M_v. In the first case, The number of triangles is denoted by $|M_v|_\triangle$. In the second case, the empty set M_v is denoted by $\breve{M}_v$ and the number of triangles is denoted by $|\breve{M}_v|_\triangle$. Note that $|\breve{M}_v|_\triangle$ counts the number of triangles tiling the support of the domain (e.g. Figure 3.4b).

Assume that M_0 contains N vertices and $\sqrt{N} = 2^d + 1$. Thus, the connection of a matrix of amplitudes yields a mesh with $n = 2 \cdot 4^d$ triangles, furthermore N and n are linked by

$$n = (N - 1)(\frac{1}{2} + \frac{1}{2^d})^{-1}. \tag{3.2}$$

Both $|M_v|_\triangle$ and $|\breve{M}_v|_\triangle$ are functions of the size of the mesh n and the connection step l of the vertex. Proposition 3.1 then gives the sizes $|M_v|_\triangle(l,n)$ and $|\check{M}_v|_\triangle(l,n)$, i.e. as functions of n and l. These functions return the sizes of a "fully expanded" merging domain, i.e the mesh boundaries are ignored. Hence the largest overestimate is obtained for the root vertex, e.g. $|M_v|_\triangle(1,n) > n$. The sizes are correct for vertices close to the center of the mesh and having a sufficiently large l. The proofs for Proposition 3.1 are given in Appendix 3.A.1.

Proposition 3.1 (Size of the merging domain) *Consider a uniform 4-8 mesh containing $n = 2 \cdot 4^d$ triangles with $d > 0$. The number of triangles $|M_v|_\triangle(l,n)$ for a vertex v connected at step $1 \geq l \geq 2d$ is given by*

$$|M_v|_\triangle(l,n) = \begin{cases} (2\log_4 n - l)2^{1-l}n, & l > 2d-4, \\ 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3}c_1^{-2}) & \\ +\frac{8}{3}c_2^{-1}c_1^2 - 16 \cdot c_1), & l \leq 2d-4, \end{cases} \tag{3.3}$$

where $c_1(l,n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$ and $c_2(l,n) = \frac{2^{-l}n}{16}$. The number of triangles $|\check{M}_v|_\triangle(l,n)$ is given by

$$|\check{M}_v|_\triangle(l,n) = \begin{cases} 2^{1-l}n - 2, & l > 2d-4, \\ 32(2^{-(l+4)}n \cdot c_1^{-1} + \frac{3}{2} \cdot c_1) & \\ -16(\log_4 n - \frac{l}{2}) - 18, & l \leq 2d-4, \end{cases} \tag{3.4}$$

with $c_1(l,n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$.

In Figure 3.5a, we depict $|M_v|_\triangle$ (middle curve) and $|\check{M}_v|_\triangle$ (bottom curve) as a function of the connection step l posing $n = 2 \cdot 4^{20}$. The top constant value represents the mesh size n. Recall that since the mesh is bounded, the maximum value at $l = 1$ is greater than n, as shown in the latter figure. Figure 3.5b shows the asymptotic behavior of $|M_v|_\triangle$ and $|\check{M}_v|_\triangle$ (posing l=1) for $n \to \infty$. The top linear curve represents the mesh size, whereas the middle and the bottom curves are given respectively by $|M_v|_\triangle$ and $|\check{M}_v|_\triangle$. We give now one more result necessary to derive the asymptotic behavior of $|M_v|_\triangle$ and $|\check{M}_v|_\triangle$ theoretically.

The following properties derive from the hierarchical connection:

$$|M_v|_\triangle(2^{i+1}-1,n) = |M_v|_\triangle(1,\frac{n}{4^i}), \tag{3.5}$$

$$|M_v|_\triangle(2^{i+1},n) = |M_v|_\triangle(2,\frac{n}{4^i}), \tag{3.6}$$

$$|\check{M}_v|_\triangle(2^{i+1}-1,n) = |\check{M}_v|_\triangle(1,\frac{n}{4^i}), \tag{3.7}$$

$$|\check{M}_v|_\triangle(2^{i+1},n) = |\check{M}_v|_\triangle(2,\frac{n}{4^i}), \tag{3.8}$$

where (3.5) and (3.7) hold for all vertices of the Cartesian grid (i.e. l is odd) and (3.6) and (3.8) holds for all vertices of the quincunx grid (i.e. l is even).

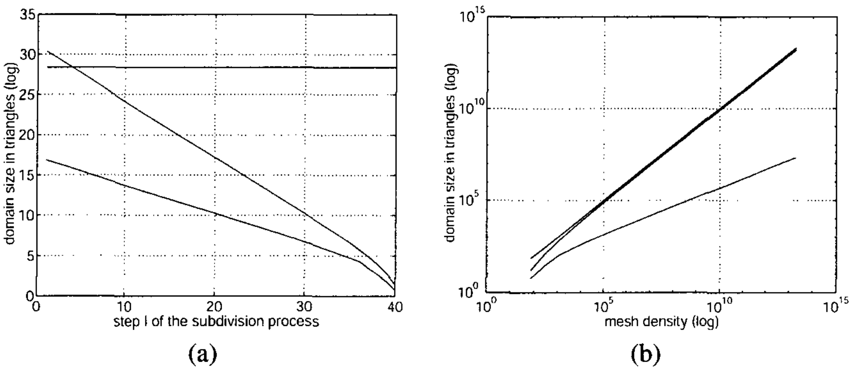


Figure 3.5: Asymptotic behavior of the merging domain: (a) Merging domain sizes as a function of l ($n = 2 \cdot 4^{20}$). The top constant value depicts the mesh size. The middle and bottom curves represent $|M_v|_\triangle$ and $|\check{M}_v|_\triangle$ respectively. (b) Merging domain size as a function of n ($l = 1$). The top linear curve represents the mesh density n, whereas the middle and the bottom curves are given for $|M_v|_\triangle$ and $|\check{M}_v|_\triangle$ respectively.

We derive now the asymptotic behavior of $|M_v|_\triangle(l, n)$ and $|\check{M}_v|_\triangle(l, n)$ or equivalently their size when $n \to \infty$ (Figure 3.5b). We first compute the size posing $l = 1$. We use the short-cuts $|M_v|_\triangle(n)$ and $|\check{M}_v|_\triangle(n)$ to denote respectively $|M_v|_\triangle(1, n)$ and $|\check{M}_v|_\triangle(1, n)$. We have

$$|M_v|_\triangle(n) = \begin{cases} (2\log_4 n - 1)n, & n < 128, \\ 7n + \frac{1024}{3}c_1^2 - 64c_1 - \frac{3}{2}c_1^{-1} + \frac{1}{6}c_1^{-2}, & n \geq 128, \end{cases} \tag{3.9}$$

and

$$|\check{M}_v|_\triangle(n) = \begin{cases} 2^{1-l}n - 2, & n < 128, \\ n \cdot c_1^{-1} + 48 \cdot c_1 - 16\log_4 n - 26, & n \geq 128, \end{cases} \tag{3.10}$$

with $c_1(n) = 2^{\lfloor \log_4 n - \frac{5}{2} \rfloor}$. We need now to bound the term $c_1(n)$, thus

$$\frac{1}{8}\sqrt{\frac{n}{2}} \leq c_1(n) \leq \frac{1}{4}\sqrt{\frac{n}{2}}. \tag{3.11}$$

For (3.9) we have that $c_1(1, n) \in \Theta(\sqrt{n})$ and $c_2(1, n) \in \Theta(n)$, hence

$$|M_v|_\triangle(n) \in \Theta(n), \tag{3.12}$$

Using (3.11), we can now lower bound (3.10) to obtain:

$$\begin{aligned} |\check{M}_v|_\triangle(n) &\geq 4\sqrt{n} + 3\sqrt{2}\sqrt{n} - 16\log_4 n - 26, \\ &\in \Omega(\sqrt{n}). \end{aligned} \tag{3.13}$$

Similarly, we upper bound (3.10):

$$\begin{aligned} |\check{M}_v|_\triangle(n) &\leq 2^{\frac{7}{2}}\sqrt{n} + 6\sqrt{2}\sqrt{n} - 16\log_4 n - 26, \\ &\in O(\sqrt{n}). \end{aligned} \tag{3.14}$$

Using (3.13) and (3.14) we have that

$$|\check{M}_v|_\triangle(n) \in \Theta(\sqrt{n}) \tag{3.15}$$

Both results (3.12) and (3.15) are confirmed by Figure 3.5b. We compute now the complexities on average. To do so, we proceed as follows:

1. We compute the merging domain size for each l and $n \to \infty$.
2. We calculate their weighted sum using the quantity of vertices (3.1) connected at each connection step l.
3. The sum is averaged by the total number of triangles n.

For the first task, we simply use (3.5) and (3.6) which allows to restrict us to two cases, namely $l = 1$ (Cartesian) and $l = 2$ (quincunx) and we vary the density n of the mesh instead. This twist allows us to reuse the results (3.12) and (3.15). For the second task, we weight the merging domain sizes using the approximation 4^i for the vertices quantities, with $i = 1 \dots d$ where is $d = \log_4(n) - \frac{1}{2}$. For the third task, using (3.2) we have that

$$\lim_{d\to\infty} \frac{n}{N} = \lim_{d\to\infty} \frac{(N-1)(\frac{1}{2}+\frac{1}{2^d})^{-1}}{N} = 2. \tag{3.16}$$

Therefore $\frac{n}{2}$ is a good asymptotic value for the number of vertices in the mesh. For $E[|M_v|_\triangle(n)]$ we have

$$\begin{aligned} E[|M_v|_\triangle(n)] &= \frac{2}{n}\left(\sum_{i=0}^{d-1} 4^i m(1, \frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^i m(2, \frac{n}{4^i})\right), \\ &= \frac{2}{n}\left(\sum_{i=0}^{d-1} 4^i \Theta(\frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^i \Theta(\frac{n}{4^i})\right), \\ &= \frac{2}{n}\left(\sum_{i=0}^{d-1} 4^i b_1 \cdot \frac{n}{4^i} + \sum_{i=0}^{d-1} 4^i b_2 \cdot \frac{n}{4^i}\right), \\ &= 2(b_1 + b_2)(d-1), \\ &= 2(b_1 + b_2)(\log_4 n - \frac{3}{2}) \end{aligned} \tag{3.17}$$

which proves that

$$E[|M_v|_\triangle(n)] \in \Theta(\log n) \tag{3.18}$$

Similarly, for $E[|\breve{M}_v|_\triangle(n)]$ we have

$$\begin{aligned} E[|\breve{M}_v|_\triangle(n)] &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^i m(1, \frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^i m(2, \frac{n}{4^i})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^i \Theta(\sqrt{\frac{n}{4^i}}) + \sum_{i=0}^{d-1} 4^i \Theta(\sqrt{\frac{n}{4^i}})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^i b_1 \cdot \sqrt{\frac{n}{4^i}} + \sum_{i=0}^{d-1} 4^i b_2 \cdot \sqrt{\frac{n}{4^i}}), \\ &= \frac{2}{n}\sqrt{n}(b_1 + b_2)[\underbrace{\sum_{i=0}^{d-1} 2^i}_{(\star)}], \end{aligned} \tag{3.19}$$

Finally, we compute the order of the term $(\star)$ by replacing d

$$\sum_{i=0}^{d-1} 2^i = \sqrt{\frac{n}{2}} - 1 \quad \in \Theta(\sqrt{n}), \tag{3.20}$$

yielding

$$E[|\breve{M}_v|_\triangle(n)] \in \frac{\Theta(\sqrt{n})\Theta(\sqrt{n})}{n} \in \Theta(c). \tag{3.21}$$

which proves that $|\breve{M}_v|_\triangle(n)$ has constant size on average. The following theorem summarizes our result on the asymptotic size of the merging domains:

Theorem 3.1 (Asymptotic sizes of the merging domain)
Consider a 4-8 mesh containing $n = 2 \cdot 4^d$ triangles with $d > 0$, the density and support size in triangles of the merging domain are given by

$$\begin{aligned} |M_v|_\triangle(n) &\in \Theta(n), \\ |\breve{M}_v|_\triangle(n) &\in \Theta(\sqrt{n}), \end{aligned} \tag{3.22}$$

whereas, on average, we have,

$$\begin{aligned} E[|M_v|_\triangle(n)] &\in \Theta(\log n), \\ E[|\breve{M}_v|_\triangle(n)] &\in \Theta(c). \end{aligned} \tag{3.23}$$

We compute now the cost for inserting a vertex. To do so, we calculate the number of triangles $|S_v|_\triangle$ connected with at least one vertex in S_v. Finding S_v only requires a bottom-up traversal of the mesh structure (exactly how this traveral is performed will be explained in the next section).

Moreover, each vertex splits two triangles in the mesh (Figure 3.4c). Therefore, for the vertices connected at the last step $l = 2d$, we have

$$|S_v|_\triangle \in \Theta(\log n). \tag{3.24}$$

The function $|S_v|_\triangle$ increases linearly with the connection step, and $|S_v|_\triangle$ is minimum for the root vertex. Therefore, averaging $|S_v|_\triangle$ over all vertices yields again

$$E[|S_v|_\triangle] \in \Theta(\log n), \tag{3.25}$$

3.3.2 Sets of ancestor vertices

Consider a decimation algorithm: We denote by A_{M_v} the set of vertices whose error is modified after decimating a vertex v, i.e. decimating M_v. Similarly, consider an insertion algorithm: We denote by A_{S_v} the set of vertices whose error is modified after inserting a vertex v, i.e. inserting S_v. Both vertices in A_{M_v} and A_{S_v} are called *ancestors*. The set A_{M_v} refers to vertices not yet decimated, whereas A_{S_v} refers to vertices not yet inserted.

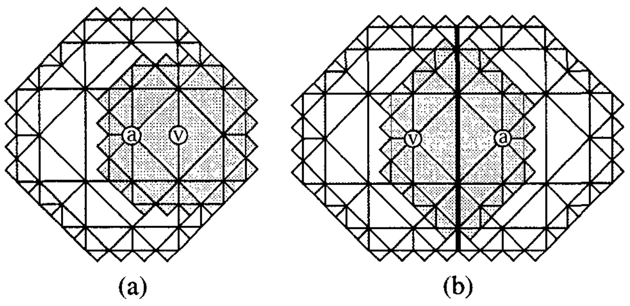


Figure 3.6: Visual representation of the ancestors of M_v: (a) $M_v \subset M_a$. (b) Overlap between M_v and M_a. The dark region depicts the domains' overlap, and the thick line is the intersection boundary.

We give now some insights on the asymptotic behavior of $|A_{M_v}|$ when $n \to \infty$. An intuitive observation is that $|A_{M_v}|$ can only have polynomial size if the merging domain remains local around v in a mesh of increasing size. Figure 3.7 represents the merging domain M_v by its support. In a mesh of infinite density (i.e. $n \to \infty$), the domain converges to the irregular octagon represented by the dark shape in the figure. Proposition 3.2 states that M_v is spatially bounded when $n \to \infty$, which implies that $|A_{M_v}|$ has polynomial size since there is a finite number of merging domains intersecting and containing M_v. The proof is given in Appendix 3.A.2.

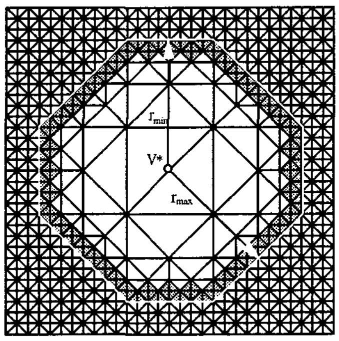


Figure 3.7: Limit surface of the support of vertex v: the white region represents the support for the actual mesh density, whereas the dark region shows the limit surface in a mesh of infinite density.

Proposition 3.2 (Limit surface of the merging domain) *The support of M_v in a mesh of increasing density is bounded by the irregular octagon with dimensions $r_{\max} = \sqrt{2}$ and $r_{\min} = \frac{3}{2}$ in Figure 3.7.*

We explain first how to find the ancestor sets A_{M_v}: We are looking for vertices a such that $M_v \subset M_a$ (Figure 3.6a), and for vertices a whose domain M_a partially overlaps M_v (Figure 3.6b). In the latter figures, we depict the merging domains using their support. The decimation of M_v has removed vertices in the merging domain of both types of vertices a as defined above.

An important property of 4-8 meshes is obtained by construction: When the mesh is subdivided, the merging domain of a vertex v is embedded in at most two merging domains of vertices (call the vertices a_1 and a_2) connected at the previous step. These vertices are the *parents* of v. Each vertex has two parents at the previous step, except for the border vertices, which have only one parent. Figures 3.8a-d depict four connection steps. The root vertex (Figure 3.8a) has no parents by definition. For steps $l > 1$ (Figures 3.8b-d), an arrow links each vertex to its parents (at the previous step l). Symmetrically reversing the arrows would link a vertex to its *descendants*. To find a chain of ancestors, denoted by A_v, for any vertex v, the arrows linking v to its parents are recursively followed until the root vertex is reached. This results in a bottom-up traversal of the mesh. For example, in Figure 3.9a we depict the chain of ancestors $A_v = \{a_i\}$, $i = 1 : 10$ of vertex v. In the example, a_{10} is the root vertex v_0. The ancestor with the smallest connection step is always the root vertex v_0. Hence for any vertex v

$$\forall a_i \in A_v, M_v \subset M_{a_i} \subset M_{v_0} \tag{3.26}$$

Moreover, $\forall i, a_i \notin M_v$. Note that the set S_v (e.g. Figure 3.4c) is found using the ancestor chain of v. However, the set S_v is composed of vertices not yet inserted, hence recursions (e.g. as shown in Figure 3.9a) are stopped when a vertex already in the mesh is met.

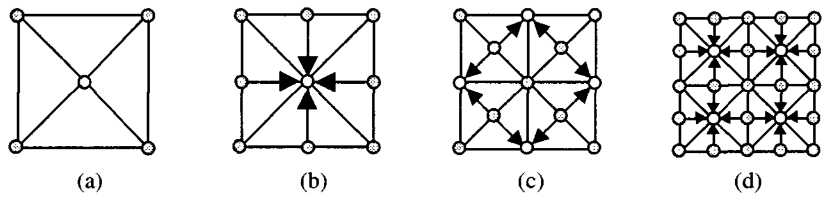


Figure 3.8: Finding the parent vertices: In each figure, the parents are represented in white and an arrow points from each vertex to its parents. (a) The root vertex has no parents by definition. Parents of the vertices inserted at steps (b) $l = 1$, (c) $l = 2$ and (d) $l = 3$.

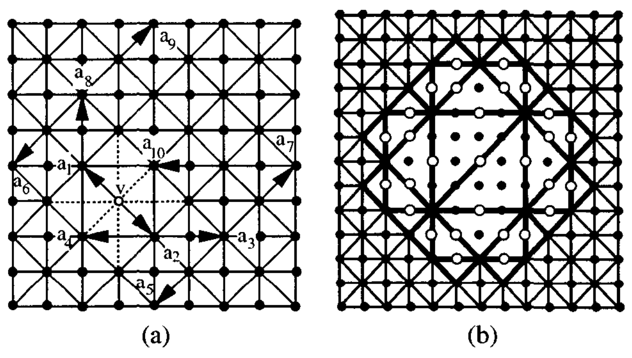


Figure 3.9: Ancestor vertices: (a) The chain of ancestors a_i built from v by recursively finding its parents towards the root vertex. Note that $a_{10} = v_0$. (b) The white vertices in M_v are the only ones with one parent not in M_v.

Construction of a chain of ancestor A_v We give below a recursive algorithm to find the chain of ancestor A_v of a vertex v:

ALGORITHM

```
$A_v \longleftarrow \emptyset$

function CONSTRUCT_SET( VERTEX $v$ )

1 if $v \neq root$
    2 FIND $a_1$ SUCH THAT $M_v \subset M_{a_1}$                see Figure 3.9a
    3 if $a_1 \notin A_v$
        4 $A_v \oplus a_1$                                      $\oplus$ denotes the insertion in the set
        5 CONSTRUCT_SET($a_1$)
    6 if $v$ IS NOT ON THE MESH BOUNDARY
        7 FIND $a_2$ SUCH THAT $M_v \subset M_{a_2}$            see Figure 3.9a
        8 if $a_2 \notin A_v$
            9 $A_v \oplus a_2$
            10 CONSTRUCT_SET($a_2$)
11 end
```

Following the above discussion, the ancestors a of M_v such that $M_a \subset M_v$ are simply found by building a chain of ancestor starting at v. How can we find ancestors when M_v and M_a partially overlap? These ancestors are found by building ancestor chains from a selected set of vertices in M_v. Again, denote by a_1, a_2 the parents of a vertex (found by following the arrows in Figures 3.8a-d). For some vertices in M_v, $a_1, a_2 \in M_v$. Therefore, these vertices must be avoided. Only a small set of vertices in M_v have one parent which does not belong to M_v. These vertices are depicted in white in Figure 3.9b. There is exactly one such vertex per triangle tiling the support of the merging domain. Therefore, with (3.21) we know that, on average, we have a constant number of such vertices. Then, the ancestors a are found by building an ancestor chain from these particular vertices, starting at the parent not belonging to M_v.

A bottom-up traversal of the mesh to find ancestors requires $\Theta(\log n)$ steps. Hence, with (3.21) we have, on average, that

$$A_{M_v} \in \Theta(c) \cdot \Theta(\log n) \in \Theta(\log n). \tag{3.27}$$

We explain now how to find A_{S_v}: A property of the vertices in M_v is

$$\forall w \in M_v, v \in A_w, \tag{3.28}$$

i.e. all the vertices $w \in M_v$ have v as an ancestor. For A_{S_v} we are looking for the vertices a whose splitting domain (hence error) has changed after inserting v, i.e. at least one vertex was inserted in S_a. More precisely, the vertices such that $\exists w \in S_v, w \in S_a$. Therefore, we have to find a subset of vertices w_i in A_v (the ancestor chain built from v) with the smallest connection step, such that no pairs w_i, w_{i+1} verifies $M_{w_i} \subset M_{w_{i+1}}$. Otherwise, the set $\{w_i\}$ is redundant. Call these vertices $\min_l(A_v)$, then following (3.28), we have

$$A_{S_v} = \bigcup_{w \in \min_l(A_v)} M_w. \tag{3.29}$$

The latter result shows an interesting duality between S_v and M_v. Using (3.23), we can conclude that, on average,

$$A_{S_v} \in \Theta(\log n). \tag{3.30}$$

3.4 Merging domain intersections

In the following sections, we propose a method for finding merging domain intersections. Recall that two types of ancestors exist for M_v: the vertices a such that $M_v \subset M_a$ and the vertices a whose domain M_a partially overlaps M_v. When $M_v \subset M_a$, then $M_v \cap M_a = M_v$. Therefore, we are interested in finding the intersection in the second case. We proceed in two steps: First, we compute the size of an intersection. The metric used to compute the size is defined in the next section. Second, we provide an algorithm that can be used to find $M_v \cap M_a$ for all ancestors a whose domain partially overlaps M_v.

3.4.1 How to describe an intersection

We describe the intersection between two merging domains as *the union of a set of (smaller) merging domains.* Using merging domains as building blocks provides a compact, efficient description for intersections. Finding the vertices in M_v only requires searching around v using a single pattern, whereas finding $M_v \cap M_a$ is difficult due to the multiplicity of cases: Just consider all the possible locations for neighboring ancestors a. Hence, Figure 3.6b is just a particular example of arrangement for $M_v \cap M_a$. Following (3.27), in total we have $\Theta(\log n)$ such arrangements.

Consider the following example: In Figure 3.10b, the intersection $M_v \cap M_a$ is the single domain M_w. In general, more than one domain is needed to represent the intersection. Consider then $M_v \cap M_a$ in Figure 3.10c: In this case, the intersection is

$$M_{w_1} \cup M_{w_2} \cup M_{w_3}, \tag{3.31}$$

which makes sense, since the vertices contained in the intersection belong to the domains $M_{w_i}, i = 1, 2, 3$.

We would like to represent the intersection as an *exclusive* set of vertices, i.e. as in (3.31). Consider $M_v \cap M_a$ in Figure 3.10d: The domain M_{w_3} overlaps with the domains M_{w_2} and M_{w_4}. We write the intersection as

$$\bigcup_{i=1}^{5} M_{w_i} = \bigoplus_{i=1}^{5} M_{w_i} \setminus D, \tag{3.32}$$

where the operator $\bigoplus$ "gathers" the vertices in the sets M_{w_i}, and D denotes the set of vertices to remove in order to obtain an exclusive set – in this case, the vertices in $M_{w_3} \cap M_{w_2}$ and $M_{w_3} \cap M_{w_4}$. To minimize the number of terms in the union (3.32), the domains M_{w_i} should be as large a possible (e.g. as depicted in Figure 3.10d). Finally, the set D in (3.32) is also expressed as a union of smaller domains. Therefore, computing this term again involves removing redundant vertices. This suggests that finding an intersection often requires recursively adding ($\oplus$) and subtracting ($\setminus$) domains (inclusion exclusion principle).

We address the problem as follows: We identify a worst case, i.e. the pair of neighbor vertices v and a with the largest intersection. Then, we propose a model to compute the size of the intersection (Section 3.4.2 and 3.4.3). The size is given in terms of domains to add or subtract, e.g. as in (3.32), in order to obtain an exclusive set of vertices. Finally, we provide an algorithm for computing all possible intersections of a merging domain M_v with its neighbors a in A_{M_v} (Section 3.4.4).

3.4.2 Modeling of the intersection between a pair of merging domains

Consider two vertices v and a arranged as in Figure 3.10a. The intersection size is maximum between domains of central vertices in two horizontal (or vertical) adjacent quads (Figure 3.10a). Figures 3.10b to 3.10d depict the union between two domains[3] attached to vertices connected respectively at step $2d-1$, $2d-3$ and $2d-5$ for a mesh of size $n = 2 \cdot 4^d$ (recall that the subdivision steps l range between 1 and $2d$). These vertices are located at the center of a quad. The intersection between the domains is shaded. We denote by I_{2d-1}, I_{2d-3} and I_{2d-5} these unions, hence

$$\begin{aligned} I_{2d-1} &= (M_v \oplus M_a) \setminus M_{w_1}, \text{with } v \text{ and } a \text{ as in Figure 3.10b,} \\ I_{2d-3} &= (M_v \oplus M_a) \setminus (\oplus_{i=1}^{3} M_{w_i}), \text{with } v \text{ and } a \text{ as in Figure 3.10c.} \end{aligned} \tag{3.33}$$

Assume that $C()$ is an operator measuring the cost to find I_{2d-j}, $j \geq 1$, as defined in the previous section. Then, we have that $C(I_{2d-1}) = 2$ and $C(I_{2d-3}) = 4$. To verify this, simply count the number of times an operator $\oplus$ or $\setminus$ is used in the above equations.

Finding I_{2d-5} requires a little more work. Call *basic domains* the domains forming an intersection in I_{2d-j}. For example, I_{2d-1}, I_{2d-3} and I_{2d-5} have one, three and five basic domains,

[3]In the figures, we choose to depict M_a as triangulated to explicitly show the density of triangles needed for the intersection.

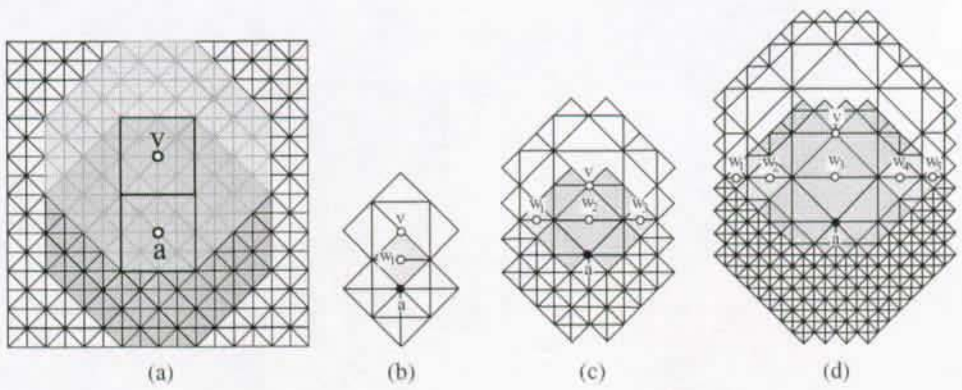


Figure 3.10: The intersection between merging domains in vertical position: (a) The intersection is maximum for direct vertical (as depicted) and horizontal neighbors. The shaded parts in (b) I_{2d-1}, (c) I_{2d-3} and (d) I_{2d-5} depict the intersections between the domains M_v and M_a.

respectively. Then, figure 3.11a depicts the intersection in I_{2d-5} and a decomposition into a set of *basic domains* M_{w_i}, $i = 1:5$ is shown in Figure 3.11b. Unlike I_{2d-1} and I_{2d-3}, some of the basic domains intersect and the left part in Figure 3.11b shows that $M_{w_2} \cap M_{w_3}$ is an instance of I_{2d-1}. Symmetrically, the same observation can be made for $M_{w_3} \cap M_{w_4}$. Therefore, to find I_{2d-5}, we must first deal with the embedded I_{2d-1}'s. Hence, I_{2d-5} can be written as

$$I_{2d-5} = \underbrace{(M_v \oplus M_a)}_{2} \setminus \underbrace{(\oplus_{i=1}^{5} M_{w_i}}_{4} \underbrace{\setminus \overbrace{I_{2d-1}}^{left} \setminus \overbrace{I_{2d-1}}^{right})}_{2(C(I_{2d-1})+1)}, \tag{3.34}$$

where "left" and "right" above the equation stand for the left and right I_{2d-1}'s in Figure 3.11b. The costs of the individual part of (3.34) are given underneath the equation. How is computed the cost $C(I_{2d-5})$ in this case? First, we account for the cost of each I_{2d-1} and the cost to substract them from $\oplus_{i=1}^{5} M_{w_i}$. Then, we add the cost for adding the five basic domains forming the intersection and the cost for substracting them to $M_v \oplus M_a$. We called this latter part of the cost *basic cost* because it does not account for embedded intersections. Hence, the total cost to compute (3.34) is

$$C(I_{2d-5}) = \underbrace{6}_{\text{basic cost}} + 2 \cdot (C(I_{2d-1}) + 1) = 6 + 2 \cdot (2 + 1) = 12. \tag{3.35}$$

More generally, finding I_{2d-j}'s with $j \geq 5$ always involves dealing with smaller embedded I_{2d-j}'s. The tree in Figure 3.12 efficiently models the problem: Each level, as well as each

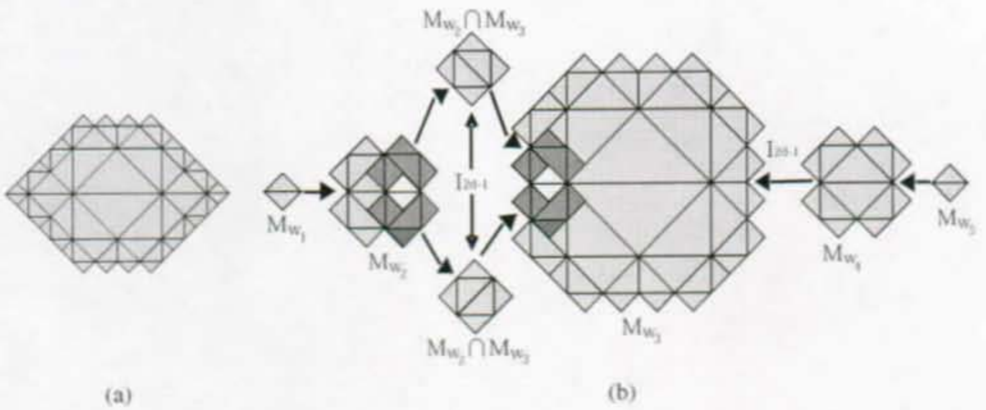


Figure 3.11: Decomposition of the intersection in I_{2d-5}: (a) The intersection in I_{2d-5}. (b) Decomposition of the intersection into a set of *basic domains* M_{w_i}, $i = 1 : 5$. The embedded intersection $M_{v_2} \cap M_{v_3}$ is further split. This intersection is an instance of I_{2d-1} (Figure 3.10b).

node, represents an instance of I_{2d-j}. For example, I_{2d-5} is represented by the first level in the tree. The two nodes at this level depict the symmetrical embedding of I_{2d-1}'s as represented in Figure 3.11b. Hence, the tree is recursive: Consider for example I_{2d-9}, which contains two instances of I_{2d-5}. Then, each instance embedds I_{2d-1}'s and is represented by the first level of the tree.

Only I_{2d-1} and I_{2d-3} do not contain embedded intersections to resolve. We can obtain a nonrecursive formulation of the tree as follows: First, we replace each node representing an instance of $I_{2d-j}, j \geq 5$, by the level representing its embedded instances I_{2d-1} and I_{2d-3}. We call $T_{1,3}$ the resulting tree. The right half of $T_{1,3}$ after substitution is shown in Figure 3.13a. The left half is a vertically mirrored version.

Then, we need a new tree $T_{5,7}$ to represent pairs of instances I_{2d-5} and I_{2d-7}, embedded in $I_{2d-j}, j \geq 9$ (Figure 3.13b). Also, we need a tree $T_{9,11}$ to account for pairs I_{2d-9} and I_{2d-11} in $I_{2d-j}, j \geq 13$, etc... More generally, we need a set of trees $T_{2d-j,2d-j-2}$ to represent instances of pairs I_{2d-j} and I_{2d-j-2}, with $j \geq 2d - j + 2$. Therefore the recursive tree model in Figure 3.12 is replaced by a set of trees.

We give now an example: How can we find $C(I_{2d-9})$ with our nonrecursive set of trees? Computing $C(I_{2d-9})$ involves two trees: First, we account for the set of embedded instances I_{2d-1} and I_{2d-3} in $T_{1,3}$ (third level in the tree of Figure 3.13a). Then, we account for the

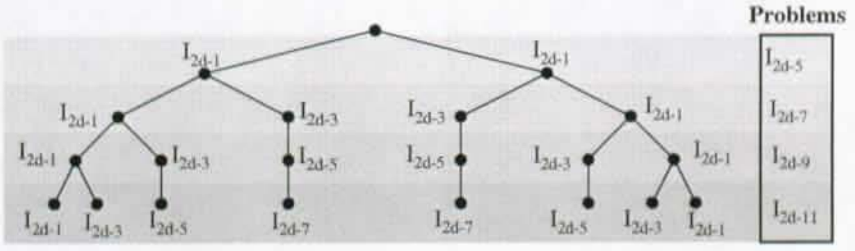


Figure 3.12: Embedding of intersection problems using a recursive tree: Each level of the tree, as well as each node, represents an instance of embedded intersection in I_{2d-j}, $j \geq 5$.

embedded instances of I_{2d-5} in $T_{5,7}$ (first level in the tree of Figure 3.13b). Finally, the basic domains forming the intersection in I_{2d-9} are taken into account.

3.4.3 Computational cost

The advantage of the nonrecursive formulation is that the appearance pattern of any pair I_{2d-j}, I_{2d-j-2} is represented by a single generic tree $T_{2d-j,2d-j-2}$. It suffices then to study this tree in order to evaluate the asymptotical cost for finding I_{2d-j}.

Let us denote by Φ and Ψ the *number of pairs* of problems I_{2d-j} and I_{2d-j-2}, respectively. The repetition pattern is given by the following system of recurrence equations

$$\begin{aligned} \Phi(k) &= \Phi(k-1) + 2\Psi(k-1), \\ \Psi(k) &= \Phi(k-1), \\ &\quad k \geq 0, \Phi(0) = 1, \Psi(0) = 0. \end{aligned} \tag{3.36}$$

The case $k = 0$ corresponds to the first level in the tree, where we have a single pair of subproblems I_{2d-j} (Figures 3.13a-b), therefore $\Phi(0) = 1$ and $\Psi(0) = 0$. The system (3.36) has the solution

$$\begin{aligned} \Phi(k) &= \frac{1}{3}(-1)^k + \frac{2}{3}2^k, \\ \Psi(k) &= -\frac{1}{3}(-1)^k + \frac{1}{3}2^k, \end{aligned} \tag{3.37}$$

where $k \geq 0$.

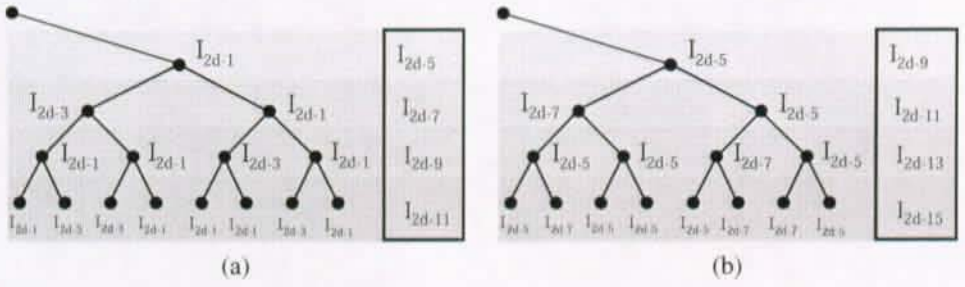


Figure 3.13: Trees for the nonrecursive model: (a) The tree $T_{1,3}$ in the figure is obtained by replacing the embedded I_{2d-i}'s, $i \geq 5$ in Figure 3.12 by their associated levels. (b) The tree $T_{5,7}$ in the figure models the occurrences of embedded I_{2d-5} and I_{2d-7} in I_{2d-i}'s, $i \geq 9$.

We compute the cost $C(I_{2d-j})$ as follows: First, we rename each cost $C(I_{2d-j})$ by $C(I_i)$, where $i = \lfloor j/2 \rfloor$. This allows for computing the costs as a single-parameter function and simplifies our computation. Then, we weight the number of pairs $\Phi(k), \Psi(k)$ of problems I_{2d-j} and I_{2d-j-2} with their basic costs, since the nonrecursive model let us use a summation across a set of trees in order to account for embedded intersections. In general, the basic cost for I_{2d-j} is $j+1$, or equivalently $2i+2$ in our single-parameter cost function. The first problem involving two trees is $C(I_{2d-7})$, i.e. $C(I_3)$. Therefore the cost for $i > 2$ is

$$C(I_i) = 2i + 2 + 2\sum_{j=1}^{p}(4j-1)\Phi(i-2j) + 2\sum_{j=1}^{q}(4j+1)\Psi(i-2j), \tag{3.38}$$

where $p = i - 1 - \lfloor \frac{i-1}{2} \rfloor$ and $q = \lfloor \frac{i-1}{2} \rfloor$. To obtain the asymptotic behavior, we sum the equation, leading to

$$\begin{aligned} C(I_i) =& 2i + \frac{90}{27}2^i + 1 - \frac{2^i}{4^{p+1}}[\frac{64}{9}(p+1) + \frac{16}{27}] - \frac{2^i}{4^{q+1}}[\frac{32}{9}(q+1) + \frac{56}{27}] + \\ & \frac{4}{3}(-1)^i[(p+1)^2 + 1 - (q+1)^2 - \frac{3}{2}(p+1) + \frac{q+1}{2}]. \end{aligned} \tag{3.39}$$

A quick analysis is performed by observing the magnitude of each term:

$$i, p, q \in \Theta(\log n), \quad 2^i \in \Theta(n), 4^{-p}, \quad 4^{-q} \in \Theta(\tfrac{1}{n}). \tag{3.40}$$

Therefore,

$$C(I_i) \in \Theta(n), \tag{3.41}$$

since 2^i is the dominant term in (3.39). As for (3.3) or (3.4), the cost $C(I_i)$ decreases exponentially when i increases. Hence averaging (3.41) over all vertices yields

$$E[C(I_i)] \in \Theta(\log n). \tag{3.42}$$

3.4.4 Algorithm computing all intersections

In this section we provide an algorithm to compute in the sets D in (3.32) between M_v and *all* the ancestors in A_{M_v} based on an inclusion-exclusion technique.

To illustrate the algorithm with a simple example, we compute the term D in (3.32). Recall the decomposition in Figure 3.11b. Then, $D = (M_{w_2} \cap M_{w_3}) \oplus (M_{w_3} \cap M_{w_4})$. We restrict our example to computing the first term of D. The intersection $M_{w_2} \cap M_{w_3}$ is shown in Figure 3.10b, hence in our example $M_{w_2} \cap M_{w_3} = M_a \cup M_v$. Assume that a and v are connected at step l. Then, w in Figure 3.10b is connected at level $l+1$ and $A_w = \{a, v, \ldots\}$, where the dots suggest addtionnal vertices. The algorithm iteratively decimates vertices starting at the ones with the largest connection step ($l+1$ in our example). Each vertex is removed from all the merging domains of its ancestors. Assume that D gathers the removed vertices forming $M_{w_2} \cap M_{w_3}$, then the algorithm proceeds as follows: First, $\forall a \in A_w$, decimate w from M_a, i.e. $M_a \setminus \{w\}$. The same is done for all other vertices connected at step $l+1$ in M_a and M_v (see the dots in D below). Then after the first step, we have

$$D = \{w, \ldots\}, \quad M_a = \{a\}, \quad M_v = \{v\}. \tag{3.43}$$

At the second step, the vertices connected at step l are considered. Hence, the vertices a and v are decimated and $D = \{w, a, v, \ldots\}$. The set D contains only one w and an exclusive set is obtained.

The algorithm below computes all intersections between M_v and M_a, with v connected at step l and $a \in A_{M_v}$. As suggested before, our algorithm finds the intersection by decimating M_v, although this is not mandatory to implement the algorithm. At each ancestor $a \in A_{M_v}$, a set D_a gathers the vertices in the intersection between M_v and M_a. Note that the decimation must be performed step-wise and starts at vertices with the largest step l.

ALGORITHM

```
1 for ALL VERTICES $w$ CONNECTED AT STEP $2d \ldots l$
    2 for ALL $a \in A_w$
        3 $M_a \setminus w$
    4 if $a \notin M_v$ then $D_a \oplus w$
  5 end
6 end
```

Since M_v contains $\Theta(\log n)$ vertices on average and $\Theta(\log n)$ operations are required to find the ancestor chain A_v, the cost of the algorithm is $\Theta(\log^2 n)$.

3.5 Application: cost of algorithms using local error

Consider an algorithm using general decimation whose input is a dense 4-8 mesh of vertices in $\mathbb{R}^3$. A progressive representation is computed using iterated decimation. An error in l_2 norm is computed for each vertex v as the sum of the squared differences between the vertices in M_v and their projection in the domain's support averaged by $|M_v|_\triangle$. Then, at each step we need $\Theta(\log n)$ operations (3.23) to decimate the vertices, and $\Theta(\log n)$ operations (3.27) to find the ancestors. For each ancestor, $\Theta(\log n)$ vertex errors (3.23) have to be locally recomputed, hence the cost for updating all errors is $\Theta(\log^2 n)$. On average, the algorithm requires $n/\Theta(\log n)$ steps to fully decompose the mesh; therefore, the minimal cost is $\Theta(n \log n)$.

Now consider an algorithm using general insertion. The input mesh has minimal resolution (e.g. Figure 3.1a) and is iteratively refined using vertex insertion. Then, at each step we need $\Theta(\log n)$ operations (3.24) to insert the vertices, and $\Theta(\log n)$ operations (3.30) to find the ancestors. Again, for each ancestor, $\Theta(\log n)$ vertex errors (3.24) have to be locally recomputed, hence the cost for updating all errors is $\Theta(\log^2 n)$. On average, the algorithm requires $n/\Theta(\log n)$ steps on to fully refine the mesh, therefore the minimal cost is again $\Theta(n \log n)$. We conclude with the following proposition:

Proposition 3.3 *On average, an algorithm based on local error (evaluated over the vertex domains) and using general decimation or insertion requires $\Theta(n \log n)$ operations to fully decompose or refine a 4-8 mesh with n triangles.*

3.6 Summary

We presented several fundamental results in computational complexity when processing 4-8 meshes. We have shown that $\Theta(\log n)$ operations are necessary to decimate or insert a vertex in the mesh while preserving the hierarchy over the vertex set. These operations yield a conforming mesh, hence the represented surface can be rendered without shape discontinuity. We have shown how to efficiently update the vertex errors when decimating or inserting vertices. More precisely, the latter operations change the errors at $\Theta(\log n)$ vertices, and on average, $\Theta(\log^2 n)$ operations are needed to update them. Since $n/\Theta(\log n)$ steps are necessary to decompose or refine a mesh of n triangles, the total cost of the algorithm is $\Theta(n \log n)$.

We addressed the problem of finding merging domain intersections and provided a model for obtaining a closed form for the computational cost of this operation. More precisely, we have shown that $\Theta(\log n)$ operations are required to compute an intersection and that all intersections between the merging domain of a vertex and the domain of its ancestors can be found in $\Theta(\log^2 n)$ operations.

Our results can be advantageously used to implement efficient algorithms. Moreover their analysis is simplified because the computational complexity mainly depends on the operations analyzed in this chapter. We will show in Chapter 5 that, using merging domain intersections, we obtain an algorithm to compute progressive representations of 4-8 meshes using general decimation and global error. This algorithm is a generalization to meshes of the algorithm `coptimal` (optimal tree-constrained approximation of polylines, see Section 2.6).

Appendix 3.A Proofs

In this section, we give the proofs for the properties in this chapter.

3.A.1 Proof of Proposition 3.1

We compute the number of triangles $|M_v|_\triangle$ and $|\breve{M}_v|_\triangle$ as follows: We construct a dual representation of the support using a tree structure. Hence, each triangle corresponds to a node and the tree expands towards the boundaries of the support (Figure 3.14a).

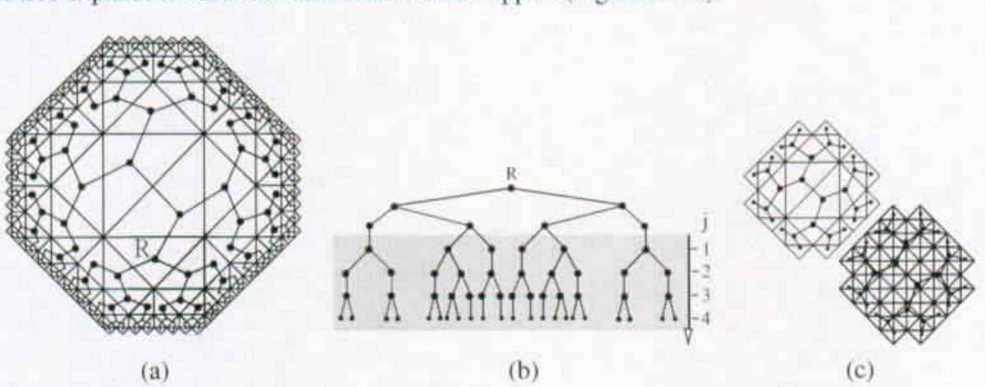


Figure 3.14: Computing $|M_v|_\triangle$ and $|\breve{M}_v|_\triangle$: (a) A dual representation of the support is constructed using a tree structure. (b) Tree structure expanding towards the boundaries. The node labeled "R" corresponds to the node in part (a). (c) The top part depicts a support and the bottom part is its triangulated counterpart. The dual tree is weighted using the number of triangles embedded in the triangles corresponding to the nodes.

Using the dual representation for the support, $|\breve{M}_v|_\triangle$ is found by summing the tree nodes; $|M_v|_\triangle$ is a weighted version of the sum. At the center of the support, the tree is balanced, i.e. each node has two children. However, the tree becomes unbalanced towards the boundaries. Figure 3.14b depicts the tree at the bottom part of the support (note that the same tree expands towards the other cardinal directions). The label "R" in Figure 3.14a and 3.14b points out the sibling nodes. The shaded region in Figure 3.14b shows the unbalanced part of the tree.

We count the tree nodes as follows: We compute two sums, one for the balanced part and one for the unbalanced part. Assume that i counts the tree levels and denote by $|\breve{M}_v|_{\triangle\mathbf{b}}(i)$ the number of triangles in the balanced part, then

$$|\breve{M}_v|_{\triangle\mathbf{b}}(i) = \sum_{k=1}^{i} 2^k = 2^{i+1} - 2. \qquad i \leq 4. \tag{3.44}$$

Assume now that j counts the unbalanced levels (vertical axis in Figure 3.14b), i.e. $j = i - 4$. Then for the unbalanced part, we use the following observation: For j odd 2^j nodes have two children and $2^{j+1} - 2$ nodes have one child. Moreover for j even, 2^{j+1} nodes have two children and $2^{j+1} - 2$ nodes have one child. Then, the sum of nodes for the unbalanced part is again split into two sums: one over odd j and one over even j. For any j, we have $j - \lfloor j/2 \rfloor$ odd indices and $\lfloor j/2 \rfloor$ even indices. We denote by $|\breve{M}_v|_{\triangle\mathbf{u}}(j)$ the the number of nodes in the unbalanced part, then for $j \geq 1$

$$\begin{aligned} |\breve{M}_v|_{\triangle\mathbf{u}}(j) &= 4(\sum_{k=1}^{j-\lfloor j/2 \rfloor} (2^{k+2} - 2) + \sum_{k=1}^{\lfloor j/2 \rfloor} (2^{k+2} + 2^{k+1} - 2)), \\ &= 32(2^{j-\lfloor \frac{j}{2} \rfloor} + 3 \cdot 2^{\lfloor \frac{j}{2} \rfloor - 1}) - 8j - 80. \end{aligned} \tag{3.45}$$

Hence, for $i > 4$

$$\begin{aligned} |\breve{M}_v|_{\triangle}(i) &= |\breve{M}_v|_{\triangle\mathbf{b}}(4) + |\breve{M}_v|_{\triangle\mathbf{u}}(i-4), \\ &= 32(2^{i-\lfloor \frac{i-4}{2} \rfloor - 4} + 3 \cdot 2^{\lfloor \frac{i-4}{2} \rfloor - 1}) - 8i - 18. \end{aligned} \tag{3.46}$$

We now compute $|M_v|_{\triangle}$ using a weighted version of the sums (3.44) and (3.45). Each tree node is weighted using the number of triangles embedded in the triangle represented by the node. This number is a function of the total number of tree levels i and the level k of the tree node. More precisely, the weight is given by $w_k = 2^{i-k+1}$. Hence, the weighted sum of (3.44) yields

$$|M_v|_{\triangle\mathbf{b}}(i) = \sum_{k=1}^{i} 2^{i-k+1} 2^k = \sum_{k=1}^{i} 2^{i+1} = i \cdot 2^{i+1}. \qquad i \leq 4. \tag{3.47}$$

Since we used two sums for the unbalanced part, we use $w_k^0 = 2^{j-2k+1}$ for the sum over even j's and $w_k^1 = 2^{j-2k+2}$ for the sum over odd j's. Hence,

$$\begin{aligned} |M_v|_{\triangle\mathbf{u}}(j) &= 4(\sum_{k=1}^{j-\lfloor j/2 \rfloor} (2^{k+2} - 2)2^{j-2k+2} + \sum_{k=1}^{\lfloor j/2 \rfloor} (2^{k+2} + 2^{k+1} - 2)2^{j-2k+1}), \\ &= 24 \cdot 2^j + \frac{8}{3} 2^{2\lfloor \frac{j}{2} \rfloor - j} - 16 \cdot 2^{\lfloor \frac{j}{2} \rfloor} - 12 \cdot 2^{j - \lfloor \frac{j}{2} \rfloor} + \frac{4}{3} 2^{j - 2\lfloor \frac{j}{2} \rfloor}. \end{aligned} \tag{3.48}$$

Hence, for $i > 4$

$$\begin{aligned} |M_v|_{\triangle}(i) =& |M_v|_{\triangle\mathbf{b}}(4) + |M_v|_{\triangle\mathbf{u}}(i-4), \\ =& 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3} c_1^{-2}) \\ &+ \frac{8}{3} c_2^{-1} c_1^2 - 16 \cdot c_1), \end{aligned} \tag{3.49}$$

where $c_1 = 2^{\lfloor \frac{i-4}{2} \rfloor}$ and $c_2 = 2^{i-4}$. To conclude, we need to express (3.46) and (3.49) in terms of the total number of triangles n and the connection step l. The parameter i is linked to n and l as $2^i = n \cdot 2^l$, or equivalently $n = 4^{(i+l)/2}$. For example, a triplet i, n, l is found as follows: In a uniform mesh of $n = 8$ triangles, $i = 2$ tree levels are needed to compute $|M_v|_\triangle(l = 1, n = 8)$ or $|\breve{M}_v|_\triangle(l = 1, n = 8)$. Hence, we replace $i = 2 \cdot \log_4 n - l$ in (3.44), (3.46), (3.47) and (3.49), Therefore, for $|M_v|_\triangle(i)$ we have

$$|M_v|_\triangle(l,n) = \begin{cases} (2\log_4 n - l)2^{1-l}n, & l > 2d-4, \\ 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3}c_1^{-2}) & \\ +\frac{8}{3}c_2^{-1}c_1^2 - 16 \cdot c_1), & l \le 2d-4, \end{cases} \tag{3.50}$$

with $c_1(l,n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$, $c_2(l,n) = \frac{2^{-l}n}{16}$, $1 \ge l \ge 2d$ and $n = 2 \cdot 4^d$.

whereas for $|\breve{M}_v|_\triangle(i)$, we have

$$|\breve{M}_v|_\triangle(l,n) = \begin{cases} 2^{1-l}n - 2, & l > 2\log_4 \frac{n}{2} - 4, \\ 32(2^{-(l+4)}n \cdot c_1^{-1} + \frac{3}{2} \cdot c_1) & \\ -16(\log_4 n - \frac{l}{2}) - 18, & l \le 2\log_4 \frac{n}{2} - 4. \end{cases} \tag{3.51}$$

and $c_1(l,n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$, $1 \ge l \ge 2d$ and $n = 2 \cdot 4^d$. □

3.A.2 Proof of Proposition 3.2

The support of M_{v^*} in Figure 3.7 grows like a geometric series along the radii $r_{\min}$ and $r_{\max}$. Count i each element of such series, then the triangles at step i are twice smaller than the triangles at the preceding step $i-1$. Series $r_{\min}$ and $r_{\max}$ have common ratio $\frac{1}{2^i}$ and $\frac{1}{\sqrt{2}}$ respectively. Therefore the dimension of the octagon are given by

$$r_{\min} = \lim_{n\to\infty} \frac{1}{2} + \sum_{i=1}^{n} \frac{1}{2^i} = \frac{3}{2} \tag{3.52}$$

and

$$r_{\max} = \lim_{n\to\infty} \frac{1}{\sqrt{2}} + \sum_{i=1}^{n} \frac{1}{2^{i+\frac{1}{2}}} = \sqrt{2} \tag{3.53}$$

□

Chapter 4

Quadtree data structure for efficient storage and access of 4-8 meshes

4.1 Introduction

4.1.1 Design of an efficient data structure

This chapter addresses the design of an efficient quadtree data structure for 4-8 meshes. We have a twofold objective, as explained below.

Efficient navigation First, we want the quadtree to provide mechanisms to implement at best the operations described in Chapter 3. Such an aspect eventually translates into the development of a *navigation method.* Recall the decimation of an arbitrary vertex in the dataset (Section 3.2.3): We show that solving this problem efficiently depends on the ability for the quadtree to provide fast node traversal methods. Moreover, important mesh processing steps, such as the determination of the visibility, rely on quick retrieval of a vertex subset.

Compact storage Second, we want the quadtree to provide *compact storage of the dataset,* rather than primarily only keep track of the connectivity of the simplified mesh as in previous works [84, 42, 19, 58, 71]. We want to take advantage of the regular connectivity provided by the subdivision scheme to describe the mesh connectivity with a minimum of bits. The data structure should also be easily extendable to arbitrary topology models, e.g. subdivision surfaces build using 4-8 subdivision.

4.1.2 Contribution and plan

We propose a new quadtree data structure called *semi-linear quadtree*, which aims at storing 4-8 meshes. The tree stores jointly individual nodes and *subtree levels*, where a subtree level is represented as a continuous array in memory. We propose a location code space to identify both nodes and subtree levels. The semi-linear quadtree construction contributes in two areas: *neighbor-finding techniques* as used in spatial databases, and *processing and storage of 4-8 meshes* for computer graphics. Our contributions are summarized below:

Neighbor-finding techniques Instead of giving an algorithm to find the index of an adjacent node as done in all previous works [29, 81, 75], we give a *closed form for all index differences in the quadtree*, hence

- Our neighbor-finding technique outperforms previous methods [75, 80, 29] in terms of computational cost.
- The closed form equations generalizes neighbor-finding methods to *traversals*. A traversal computes the index of an arbitrary node in the tree with only the characteristics of the starting node (Section 4.4.4).
- Although a particular node or subtree level is generally accessed in $O(\log m)$, accessing a node within a subtree level has constant cost.

Processing and storage of 4-8 meshes The semi-linear quadtree provides efficient mechanisms to store and process 4-8 meshes. We expose below its improvements over previous implementations [29, 81, 75]:

- No location code is needed to index nodes within a subtree level, leading to more compact storage than previous implementations.
- The semi-linear quadtree dynamically improves *both storage and access cost* while growing by *gathering nodes into subtree levels*, as well as *subtree levels into larger levels*. At one extreme, a dense quadtree can be stored as a single array in memory. In this case, accesses have constant cost and no location code needs to be stored explicitly for the tree.
- We further extend traversals to *traversal paths*. A traversal path is a powerful paradigm *to state computationally optimal implementations of algorithms on the quadtree*, i.e. the algorithm complexity has the same magnitude as the problem size. We give examples of applications in Section 4.7.
- We provide a method to store the connectivity and the geometry of all vertices without any redundancy, thus solving the issues pointed out in [58].
- Our framework extends naturally to spherical datasets using a subdivided octahedron and to models of arbitrary topology, e.g. subdivision surface.

- Finally, large mesh patches can be described with a forest of quadtrees without additional work leading to a framework to process and store arbitrary large meshes.

We start this chapter with a in-depth review of background material and previous investigations. This review is important in order to clarify our contributions. Then, Section 4.3 presents a quadtree for 4-8 meshes called *semi-linear quadtree*. Section 4.4 and 4.5 present respectively a neighbor-finding technique and an efficient storage method for the quadtree. Section 4.6 explains how to adapt the quadtree to meshes having a more involved topology. Finally, Section 4.7 introduces applications for the processing of 4-8 meshes.

4.2 Background material and review

4.2.1 Square quadtree and triangle quadtree

The quadtree is a widely used data structure in computer graphics [84, 42, 58, 71] and image processing [81, 29]. It provides an efficient means to describe hierarchical datasets and recursive subdivision processes. A common utilization of the quadtree is the description of recursive subdivisions of the plane into square regions[1] [81, 29, 75], as shown in Figure 4.1a. This formulation is still the most appropriate to describe 4-8 meshes when starting from an initial square of two triangles (Figure 3.1a).

The appearance of alternate subdivision schemes led to the study of quadtrees describing recursive subdivisions of triangular regions (Figure 4.1b) [56, 24, 32]. The two constructions are named *square quadtree* and *triangle quadtree*, respectively. A triangle quadtree is typically used to describe meshes whose connectivity is based on quaternary subdivision [94, 36, 48].

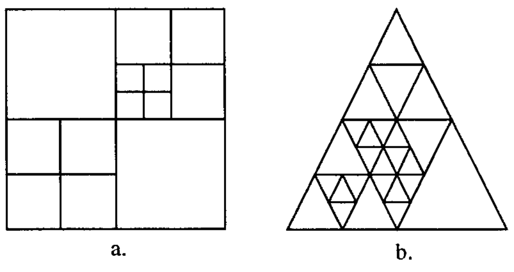


Figure 4.1: Different quadtrees: (a) The square quadtree. (b) The triangle quadtree.

[1]Each subdivision splits a square into four squares of equal size.

4.2.2 Overview of mesh storage using quadtrees

Our aim being to store 4-8 meshes, we focus our review on the square quadtree and simply refer to this particular formulation as quadtree for the rest of this thesis. Each quadtree node represents a triangulated square, as represented in Figure 4.2. Alternatively to common representation [76], we propose to depict a quadtree as shown in the left hand-side of Figure 4.2: We link together only the nodes having a common father and located at the same level. This representation transcribes the spatial nature of the dataset.

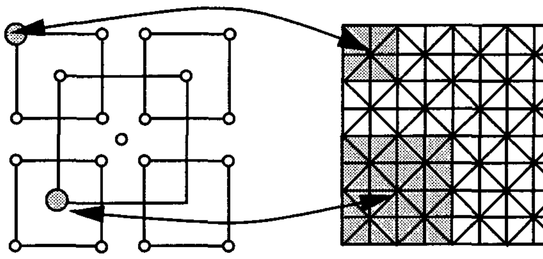


Figure 4.2: Quadtree structure storing a 4-8 mesh: To have a clear representation of the quadtree in the figure, we link together only the nodes having a common father, and located at the same level. The arrows link the nodes to their corresponding regions in the mesh.

Note that, neighbor squares share common vertices on their edges which may induce redundant storage [58, 71]. We will address this issue later in Section 4.5.

4.2.3 Restricted quadtree

Early usages of the quadtree for the display of meshes take us back to Catmull [12] in the 1970's. In order to rasterize a parametric surface, Catmull proposes an algorithm starting with a single quadrilateral approximating the model. Then, the patch is recursively subdivided until each pixel of the display is filled by the color of a single subpatch (see pages 286-288 in [76]). The output is a set of quadrilaterals that is well represented using a quadtree. Unfortunately, the set of adjacent patches are not coplanar in general, which provokes discontinuities or *cracks* on the surface, as depicted in Figure 4.3a.

This problem is addressed by Tamminen *et al.* [84]. Their solution consists in aligning vertices shared by adjacent edges (Figure 4.3b). Later, Von Herzen *et al.* [42] propose to combine a decomposition rule with triangulation in order to solve the problem: Patches are further subdivided such that neighbor squares do not differ from more than one subdivision level

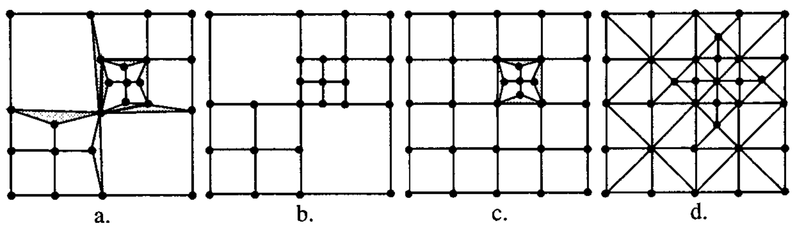


Figure 4.3: Preserving of the continuity of a subdivided surface: (a) A set of quadrilateral is generated to rasterize a parametric surface. (b) Tamminen *et al.* align the vertices shared by the edges of adjacent patches [84], whereas (c) Von Herzen *et al.* [42] further subdivide the patches such that adjacent squares do not differ from more than one subdivision level, and (d) the resulting subdivision is triangulated.

(Figure 4.3c). The resulting set of quadrilaterals is finally triangulated and has 4-8 connectivity (Figure 4.3d). This construction is called *restricted quadtree*.

More recent contributions are due to Lindstrom *et al.* [58] and Pajarola [71]. They both propose the same approximation method based on quadtrees. The mesh is refined by inserting vertices and the surface is retriangulated in order to obtain a restricted quadtree (see Section 3.3.1). Solving the problem of computing a restricted quadtree after inserting a vertex is equivalent to finding its set of ancestors (Section 3.3.2). Albeit Lindstrom *et al.* do not give the complexity of their algorithm, Pajarola claims a computational optimal algorithm running in linear time with the tree size[2], but does not give any proof.

Navigation methods, also referred as *neighbor-finding algorithms* for the quadtree have been a active area of research in the *spatial databases* and *geographical information systems* community. Finding the adjacent squares [75, 81, 29] or triangles [70, 32, 24, 56] (Figures 4.1a-b) is a fundamental operation for a number of algorithms. To implement their decomposition rule, Von Herzen *et al.* [42] take advantage of such techniques. However, neither Lindstrom *et al.* nor Pajarola use neighbor-finding algorithms to carry out their refinement scheme. A possible explanation could be the following: Neighbor-finding methods have been developed for a particular implementation of the quadtree called *linear quadtree*. A linear quadtree is a list of nodes representing *the leaves of the tree*. It is used as an alternative to common pointer-based implementations since in most cases (see [76]) it saves a substantial amount of memory[3].

Each node is assigned a *location code* describing its subdivision path from the root. This

[2]The size is defined as the number of nodes in the tree.

[3]In the pointer-based implementation, each node has to store four pointers.

code is used to define adjacency relationships between nodes. Since our contribution to neighbor-finding techniques is tightly related to the design and processing of location codes, we will give more details in the next paragraph. For the case of 4-8 meshes as used by Lindstrom *et al.* and Pajarola, the connectivity is encoded across levels of the quadtree since the hierarchy of the data structure lodges the recursive subdivision process, i.e. the mesh is locally stored *in a subtree* and a set of leaf nodes is inadequate to characterize the connectivity and the dataset. As used in Von Herzen *et al.* (Figure 4.3d) the linear quadtree formulation somehow "flattens" the multiresolution nature of the mesh. Therefore, previous works on neighbor-finding techniques for linear quadtree [75, 81, 29] are not immediately applicable since location codes constrain to perform operations on leaf nodes.

4.2.4 Linear quadtrees and neighbor finding techniques

In the previous section, we explained that a linear quadtree is represented by a list of leaf nodes, each containing a location code. The location code is stored as an integer and its expression in base 4 reflects the quadrant subdivisions leading to the node. Additionally, the level of the node is stored. The list is finally sorted using the decimal expression of the location code and the level. Operations on the quadtree, for example *neighbor-finding techniques*, are implemented in terms of manipulations of the location code and have been studied by Samet [75], Schrack [81] and Gargantini [29]. Samet and Gargantini adopt the same convention for the assignment of quaternary digits to quadrants, whereas Schrack opts for an alternate one –horizontally mirrored– as depicted in Figure 4.4a. However, both conventions lead to the same code semantic. Figure 4.4b shows a partition and the quaternary location codes corresponding to each node are given. Figure 4.4c gives the equivalent decimal expressions.

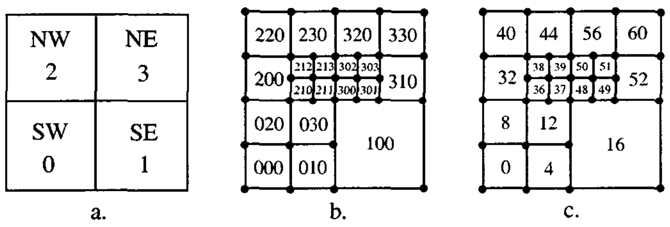


Figure 4.4: Location codes: (a) Orientation corresponding to quaternary digits. (b) Quaternary location codes and (c) their equivalence in decimal notation.

Samet's, Gargantini's and Schrack's neighbor-finding techniques proceed in the following way: First, the quaternary location code of the node (initially stored as an integer) is calculated. The level of the node is used to stop the decoding process. Then, the code of the neighbor in the desired direction is computed (see next paragraph). Finally, the list is traversed to determine the existence or the absence of the neighbor (after converting back the quaternary code). Their approaches differ in the way the location of the neighbor is obtained, i.e. how the code is processed.

Assume that the smallest square in the subdivision belongs to an $2^n \times 2^n$ grid[4] and that m squares are encoded. Samet's and Gargantini's methods operate directly on the quaternary expression. To find a neighbor, the digits are "reflected" in the appropriate direction according to their assignment to quadrants: For example node 300_4 has western neighbor 211_4 (Figure 4.4b). Codes are reflected starting from the least significant digit until a common father (detected by an equal digit) is met. For instance, 300_4 has northern neighbor 302_4 (Figure 4.4b), i.e. only the least significant digit is reflected since a common father is already encountered at the second digit. In conclusion, Samet's and Gargantini's methods have complexity $\Theta(\log m + 3n)$. The factor 3 enters because we need two conversions and n digits reflections.

An important property of the location code is that its binary expression contains the coordinates of the pixel represented by the node [29, 69, 53], referred as *interleaved coordinates* [80]. Schrack's method operates on the interleaved coordinates of the location code. He introduces the concept of *dilated integers* to perform operations directly on the coordinates. In this setting, neighbor directions are expressed using couples of relative increments (for example, the NW neighbor corresponds to the increments $\Delta = (-1, 1)$) and the adjacent node is found using a constant number of operations. However, the total complexity of Schrack's algorithm is $\Theta(\log m + 2n)$ since we still need two conversions.

4.3 A quadtree for 4-8 meshes

4.3.1 Design of the location code

Linear quadtrees are appropriate to describe the recursive subdivisions of a square region [75, 80, 29], however in the case of 4-8 meshes, the recursive construction spreads information (connectivity and vertices) across levels of the tree. Therefore, *subtrees* must be stored leading to a different construction. When used in this context, the location codes proposed by Samet, Gargantini and Schrack overlap, i.e. the same integer appear more that once in the list. However, the list can still be sorted since the level of the node is stored jointly. However, their neighbor-finding techniques are not defined in this case. We use the continuous integer range $0 \ldots \frac{1}{3}(4^d - 1) - 1$ as *location code space*, since a complete (or balanced) quadtree can be implemented as a continuous array of nodes in memory, as stated in the following definition:

Definition 4.1 (quadtree indices as a continuous range of integers)
A quadtree of depth d contains $\sum_{i=0}^{d-1} 4^i$ nodes, with indices ranging from $0, \ldots, \frac{1}{3}(4^d - 1) - 1$. For a tree of depth $d > 0$, each node p is located at level $\lfloor \log_4(3p + 1) \rfloor$. Each node $p > 0$ has

[4] In this case, location codes of n bits are required.

a parent node $\lfloor \frac{p-1}{4} \rfloor$, and each node such that $p < \frac{1}{3}(4^{d-1} - 1) - 1$ (i.e. not corresponding to a leaf) has child nodes $4p + i$, with $i = 1 \ldots 4$.

The properties of the location code construction are summarized below:

Property 4.1 (Location code)

4.1.1 A location code p contains the level of the node and the level is $\lfloor \log_4(3p + 1) \rfloor$.

4.1.2 Each level i of the quadtree 4.1 can be seen as an $2^i \times 2^i$ grid. A location code p at level i contains a local location code $p - \frac{1}{3}(4^i - 1)$ embedding the interleaved coordinates locating the node inside the grid at level i.

4.1.3 Subtree levels are represented by continuous ranges of integers.

A local location code at level i is equivalent to the code used by Samet, Gargantini and Schrack to describe an $2^i \times 2^i$ grid. Therefore, all properties studied in [75, 80, 29, 69, 53] still hold for a local code.

The assignment of quaternary digits to quadrants must be generalized accordingly. Figure 4.5a depicts the spatial organization of a node and its child nodes. We named the generalized digit assignment *z-ordering*. A direct consequence of the assignment is that the index differences between rows and columns at every levels in the quadtree are constant, as depicted by the arrows in Figure 4.5b. The property is also true with the assignment used by Samet, Gargantini and Schrack between neighbor pixels at the same level, i.e. represented by squares of equal size (see Figure 4.4).

4.3.2 Semi-linear quadtree

In this section, we propose a new tree construction called *semi-linear quadtree* (SLQ). The SLQ aims at generalizing the linear quadtree construction presented in [75, 80, 29]. We call it *semi-linear* because both nodes and subtree levels are stored jointly. The location code construction described in the previous section allows for the description of subtrees, which is a requirement to store 4-8 meshes. The generalization comes only with a little increase in storage price (1 quaternary, or 2 bits per location code) since the space spanned by our code is four times larger than the previous implementations [75, 80, 29]. However we must keep in mind that quadtrees are exponential in nature. Therefore even a small increase in code size can result in great differences in storage requirements. This issue is addressed as follows: in the SLQ, *individual nodes* and *subtree levels* are stored jointly in the data structure. Subtree levels can be stored as arrays in memory considering Property 4.1.3 of our location code. This has the following interesting outcomes:

- No location code needs to be stored for the nodes in a subtree level, since it can be found by incrementing (i.e. shifting) the code of the root node.

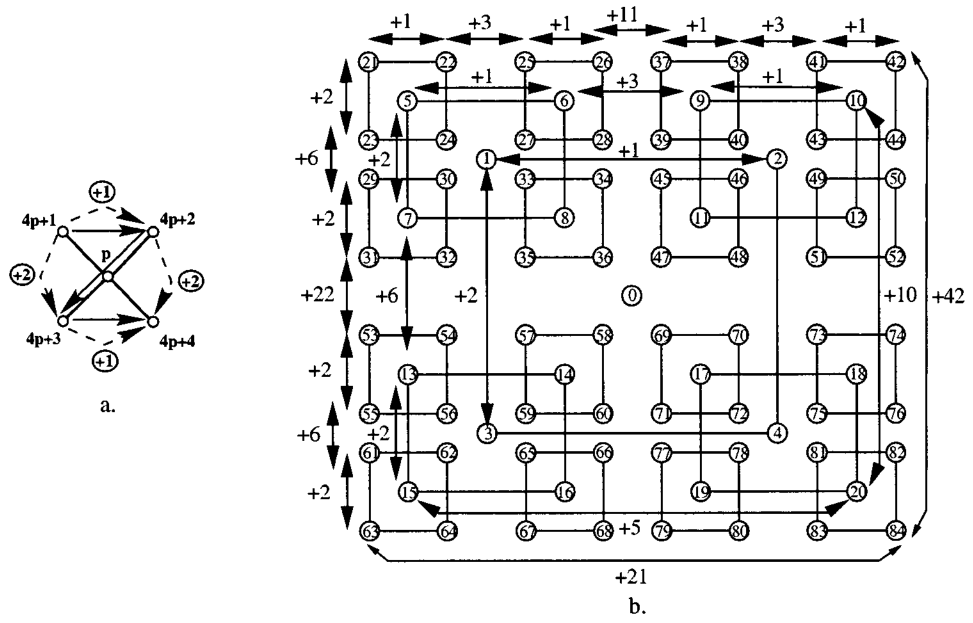


Figure 4.5: Quadtree indexing: (a) The z-ordering is used to assign an index to each child. (c) Spatial index organization using z-ordering for a balanced quadtree of depth 4.

- Nodes within subtree levels are accessed in constant-time.

The SLQ includes a control algorithm grouping nodes and subtree levels when modifying the structure of the tree, i.e. inserting or deleting nodes. At the extreme, a dense quadtree can be stored as a continuous array in memory. In this case, all accesses have constant cost (compared to always $O(\log m)$ for linear quadtree implementations [75, 80, 29]) and no location code is needed to index the nodes. However, we will not describe the control algorithm here; instead we will focus on the development of neighbor-finding techniques for the SLQ.

4.4 Neighbor-finding technique

4.4.1 A closed form for index differences

Recall that a property of location codes is that the index differences, between columns and rows of nodes at the same level, is constant. This fact suggests that a closed form for the index differences can be found. Samet's and Gargantini's reflection rule on quaternary codes and Schrack's operations on the interleaved coordinates compute these index differences to find neighbors. Call now r the *relative level distance* (RLD) between two nodes. The RLD is the distance in terms of quadtree levels between two nodes to their closest common father. The closed form for the index differences is given in Theorem 4.1 in terms of the RLD and the proof is given in Appendix 4.A.

Theorem 4.1 (closed form for the index differences)
The horizontal/vertical differences between the indices of adjacent nodes having a relative distance distance r are constant for a particular column/row. The horizontal differences are

$$\delta_h(r) = \frac{2}{3}4^r + \frac{1}{3}, \tag{4.1}$$

$$\delta_{ht}(r) = \frac{1}{3}4^{r+1} - \frac{1}{3}. \qquad \textit{(toroidal)} \tag{4.2}$$

The vertical differences are

$$\delta_v(r) = 2\delta_h(r) = \frac{1}{3}4^{r+1} + \frac{2}{3}, \tag{4.3}$$

$$\delta_{vt}(r) = 2\delta_{ht}(r) = \frac{2}{3}4^{r+1} - \frac{2}{3}. \qquad \textit{(toroidal)} \tag{4.4}$$

Equations (4.1) and (4.3) in Theorem 4.1 express the vertical and horizontal differences δ between the node indices as a function of r. Additionally, (4.2) and (4.4) give a similar difference for the border nodes with their opposite neighbors (which is equivalent to assume that the quadtree has a toroidal structure). These latter equations are very useful to extend neighbor-finding techniques to forests of quadtrees or to implement more involved topologies (Section 4.6). Recall now the reflection rule of Samet and Gargantini (Section 4.2.4). Their algorithms stop as soon as a common father is met, or equivalently when the same quaternary digit is found. The RLD actually gives the number of digits to examine for a pair of nodes, since it measures the distance to their common father. It makes sense then to use the RLD as input value for the closed form. In the next section, we explain how the RLD between two nodes is obtained.

4.4.2 Recurrence equations for the relative level distance

Consider two neighbor nodes p_1 and p_2, then their RLD is the solution to

$$\lfloor\frac{p_1 - 1}{4^r}\rfloor = \lfloor\frac{p_2 - 1}{4^r}\rfloor. \tag{4.5}$$

The recursive nature of the quadtree allows us to derive a set of recurrence equations for the solutions to (4.5). Call $\mathbf{r}^i$ the RLD's for the nodes of level i, then we have

$$\begin{aligned} \mathbf{r}^1 &= \left[0,\ \phi_1,\ 0\right], & \phi_1 &= \left[0\right], \\ \mathbf{r}^i &= \left[i-1,\ \phi_i,\ i-1\right], & \phi_i &= \left[\phi_{i-1},\ i-1,\ \phi_{i-1}\right]. \end{aligned} \tag{4.6}$$

Therefore, (4.1)-(4.4) are evaluated using the vectors $\mathbf{r}^i$. We need then one vectors per level (since the horizontal and vertical distance are proportional) to retrieve all index differences in the quadtree. Call $\Delta_h^i(\mathbf{r}^i)$ and $\Delta_v^i(\mathbf{r}^i)$ the vectors giving the index differences for pairs of horizontal and vertical neighbor nodes, respectively. Then, for $\Delta_h^i(\mathbf{r}^i)$ we have

$$\begin{aligned} \Delta_h^1(\mathbf{r}^1) &= \left[-\delta_{ht}(0),\quad \delta_h(0),\quad -\delta_{ht}(0)\right], & \Phi_1 &= \left[\delta_h(0)\right], \\ \Delta_h^i(\mathbf{r}^i) &= \left[-\delta_{ht}(i-1),\quad \Phi_i,\quad -\delta_{ht}(i-1)\right], & \Phi_i &= \left[\Phi_{i-1},\quad \delta_h(i-1),\quad \Phi_{i-1}\right]. \end{aligned} \tag{4.7}$$

Following Theorem 4.1, for $\Delta_v^i(\mathbf{r}^i)$ we have that

$$\Delta_v^i(\mathbf{r}^i) = 2\Delta_h^i(\mathbf{r}^i), i \geq 0. \tag{4.8}$$

For example, Δ_h^1, Δ_h^2 and Δ_h^3 yield respectively

$$\begin{aligned} \Delta_h^1 &= \left[-1,\quad 1,\quad -1\right], \\ \Delta_h^2 &= \left[-5,\quad 1,\quad 3,\quad 1,\quad -5\right], \\ \Delta_h^3 &= \left[-21,\quad 1,\quad 3,\quad 1,\quad 11,\quad 1,\quad 3,\quad 1,\quad -21\right], \end{aligned} \tag{4.9}$$

whereas for Δ_v^1, Δ_v^2 and Δ_v^3 we have respectively

$$\begin{aligned} \Delta_v^1 &= \left[-2,\quad 2,\quad -2\right], \\ \Delta_v^2 &= \left[-10,\quad 2,\quad 6,\quad 2,\quad -10\right], \\ \Delta_v^3 &= \left[-42,\quad 2,\quad 6,\quad 2,\quad 22,\quad 2,\quad 6,\quad 2,\quad -42\right], \end{aligned} \tag{4.10}$$

as depicted in Figure 4.5b. Finally, note that because the vectors are symmetric for each direction, only half of the elements are actually needed. In the rest of this chapter, we will refer to these vectors simply as Δ-vectors.

4.4.3 Interleaved coordinates

To complete our navigation framework, we need to find a way to access the Δ-vectors. To achieve this, we use the interleaved coordinates contained in the local location code. We proceed in two steps: First, the local location code is computed. Second, we decode its quaternary expression and derive the interleaved coordinates. Call p_s the node at level i for which a neighbor is searched and $[g_h, g_v]$ its interleaved coordinates within the level. Then, the local location code is given by

$$p_s - \frac{1}{3}(4^i - 1). \tag{4.11}$$

Its quaternary expression is denoted by $\mathbf{q}_{p_s} = \{q_{i-1}, \ldots, q_0\}$ and has length i. To obtain the interleaved coordinates, we define a coordinate system for the level and compute the coordinates with a simple sum of 2×2 matrices. Setting the origin to the top leftmost node leads to the following four matrices

$$\mathsf{F}_0 = \mathbf{0}, \quad \mathsf{F}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathsf{F}_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathsf{F}_3 = \mathbf{I}_2, \tag{4.12}$$

where each matrix corresponds to a digit in base 4. Then $[g_h, g_v]$ is given by

$$\begin{bmatrix} g_h \\ g_v \end{bmatrix} = \sum_i \mathsf{F}_{q_i} \begin{bmatrix} 2^i \\ 2^i \end{bmatrix}. \tag{4.13}$$

Example 4.1
Consider node 59, located at level 3 and having local index 38. The local location code is $\mathsf{q} = \{2, 1, 2\}$. Hence,

$$\begin{bmatrix} g_h \\ g_v \end{bmatrix} = \mathsf{F}_2 \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \mathsf{F}_1 \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \mathsf{F}_2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} \tag{4.14}$$

The coordinates g_h and g_v are used to access Δ_h^i and Δ_v^i respectively. Note that Δ_h^i and Δ_v^i have both size $2^i + 1$. Nodes p_s and p_t are further called *the starting node* and *the target node* respectively. The vectors are accessed in the following way:

- Its *western neighbor* is given by

$$p_t = p_s - \Delta_h^i[g_h]. \tag{4.15}$$

- Its *eastern neighbor* is given by

$$p_t = p_s + \Delta_h^i[g_h + 1]. \tag{4.16}$$

- Its *northern neighbor* is given by

$$p_t = p_s - \Delta_v^i[g_v]. \tag{4.17}$$

- Its *southern neighbor* is given by

$$p_t = p_s + \Delta_v^i[g_v + 1]. \tag{4.18}$$

Example 4.2
For example, we now use the Δ-vectors to find the neighbors of 8, thus

- Its *western neighbor* is $p_t = 8 - \Delta_h^2[1] = 8 - 1 = 7$.
- Its *eastern neighbor* is $p_t = 8 + \Delta_h^2[2] = 8 + 3 = 11$.
- Its *northern neighbor* is $p_t = 8 - \Delta_v^2[1] = 8 - 2 = 6$.
- Its *southern neighbor* is $p_t = 8 + \Delta_v^2[2] = 8 + 6 = 14$.

These results can be verified in Figure 4.5b.

Assume a quadtree of m nodes and a location code of length n. First, we need to convert the decimal expression of the node index. Second, we compute the matrix sum to find the interleaved coordinates. We can conclude that our neighbor-finding method has complexity $\Theta(\log m + 2n)$. However, in a dense quadtree the logarithmic term fades away since nodes are accessed in constant time on average. Thus in the latter case, the complexity on average is $\Theta(2n)$ operations.

4.4.4 Traversals and traversal paths

As seen in the previous sections, our solution allows adjacent neighbors to be found using a closed form for the index differences, avoiding to compute the distance each time a neighbor is searched as needed in [75, 80, 29]. In this section, we explain how to go beyond adjacent neighbor-finding schemes and introduce the generalized notions of *traversals* and *traversal paths.*

Traversals Assume again a starting node p_s and a target node p_t. Both can be arbitrary nodes in the quadtree. A *traversal* is achieved if we can find two scalars $\Upsilon(p_s)$ and $\Delta(p_s)$ such that

$$p_t = \Upsilon(p_s) + \Delta(p_s), \tag{4.19}$$

where Υ is a *scaling function* and Δ is an index difference. The key fact is that the values Υ and Δ are only functions of p_s and a geometric relation $\mathsf{G}(p_s, p_t)$ between the two nodes (Figure 4.6a). The geometric relation defines how Υ and Δ are implemented and can be seen as a set of unitary displacements[5] in the tree. The chosen sequence is arbitrary and links p_s and p_t. A traversal is constructed as follows:

1. The index p_s is scaled, i.e. the index of an ancestor or a descendant at the same level of p_t is computed. The scaled node index is $\Upsilon(p_s)$. More details on the general form of the scaling function are given at the end of this paragraph.
2. The shortest path from $\Upsilon(p_s)$ to p_t using vertical and horizontal displacements is used to construct Δ. The index difference Δ is a sum of Δ_v^j and Δ_h^j where j is the level of p_t.

[5]As computed in the previous section. Additionally, moves to father and children nodes are allowed.

Assume that $[g_h, g_v]$ are the interleaved coordinates of p_s. Then, the coordinates of $\Upsilon(p_s)$ are easily obtained by multiplying or performing the integer division by 2 of $[g_h, g_v]$ and adding or subtracting unit increments. Call $[g'_h, g'_v]$ the scaled coordinates, then the coordinates of the subsequent nodes in the traversal are obtained by *incrementing* its components according to the directions. Therefore, the Δ-vectors Δ^j_v and Δ^j_h can be appropriately accessed.

Equations (4.15)-(4.18) implement the relations G_0 (i.e. horizontal) and G_1 (i.e. vertical) depicted in Figure 4.6a (in both directions). In this case, Υ is the identity function. The traversals G_2 and G_3 are implemented using the methodology described above. Thus, for G_2 we have

$$G_2 : \qquad p_t = p_s + \Delta^i_h[g_h + 1] - \Delta^i_v[g_v], \tag{4.20}$$

whereas G_3 is given by

$$G_3 : \qquad p_t = \underbrace{4 \cdot p_s + 2}_{\Upsilon} + \underbrace{\Delta^{i+1}_h[2g_h + 2] - \Delta^{i+1}_v[2g_v]}_{\Delta}. \tag{4.21}$$

Example 4.3
Assume that $p_s = 12$ (at level 2), then G_3 yields $p_t = 24$ (Figures 4.5b). Namely we have $[g_h, g_v] = [3, 1]$ and

$$p_t = 4 \cdot 12 + 2 + \Delta^3_h[8] - \Delta^3_v[2] = 50 - 21 - 6 = 23. \tag{4.22}$$

Assuming that r levels are separating nodes p_s and p_t, then the general form for the scaling function $\Upsilon(p_s)$ is given by the scalar product

$$\Upsilon(p_s) = [4^r 4^{r-1} \ldots 1] \begin{bmatrix} p_s \\ c_1 \\ \vdots \\ c_r \end{bmatrix}, \tag{4.23}$$

where $c_1, \ldots c_r$ is a series of r child nodes' locations, therefore c_r is an integer such that $1 \le c_r \le 4$. For example, in G_3 we have $r = 1$ and $c_1 = 2$ (see Equation (4.21) and Figure 4.6a).

Traversal paths A *traversal path* is defined as a series of visited nodes, as depicted in Figure 4.6b. A traversal path only differs from a traversal in the sense that the indices of the successive nodes between p_s and p_t are computed. A traversal path is a powerful paradigm for implementing algorithms visiting a series of adjacent nodes in the quadtree (see Section 4.7). When constructing traversals in the previous paragraph, we simply used the fact that geometric relations are *additive*. For example, in Figure 4.6a we have

$$G_2 = G_1 \oplus G_0, \tag{4.24}$$

where $\oplus$ denotes the successive application, i.e. composition, of G_0 and G_1. Similarly, G_3 was implemented with a series of relations G_0 and G_1, plus a scaling operation for the initial node.

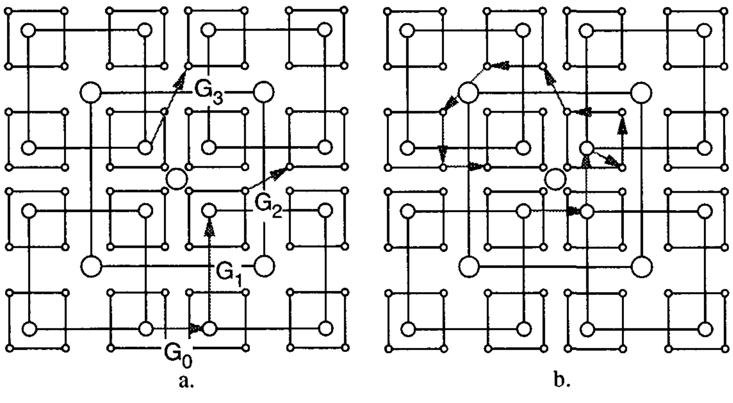


Figure 4.6: Traversals and traversal paths: (a) The geometric relations between pair of nodes represent *traversals* in the tree. (b) A traversal path of length 10.

The advantage of using traversals and traversal paths over previous neighbor-finding methods [75, 80, 29] is their low computational complexity. Their cost is linear with the path length because only the quaternary location code of the initial node p_s in the path must be computed in order to obtain its interleaved coordinates. Each neighbor-finding step costs one addition compared to d operations [75, 29], for a quadtree of depth d. Schrack [80] also finds neighbors at constant cost, but needs (as in [75, 29]) to convert the quaternary digits of the code into a decimal expression expression in order to check availability. This step is not necessary with our method because the *Δ-vectors store decimal values*, allowing us to directly obtain the decimal expression of the neighbor index.

Assuming a location code of length n and a quadtree of m nodes, the expected cost of a sufficiently long traversal or traversal path in a sparse quadtree is $\Theta(l \log m)$. Finally, the use of a SLQ greatly improves access performances, since subtree levels are stored as continuous arrays in memory when the quadtree is dense. In conclusion, the expected cost of a traversal or a traversal path is $\Theta(l)$, whereas previous methods in [75, 29] and [81] yield a cost of $\Theta(l \log m + 3ld)$ and $\Theta(l \log m + 2ld)$, respectively.

4.5 Efficient storage of the mesh

This section describes an efficient method to store 4-8 meshes using the SLQ. For the sake of clarity, we describe the storage for a balanced quadtree, but our method applies to sparse trees as well.

In a 4-8 mesh, each triangulated square can potentially describe the characteristics (i.e. value and connectivity) of five vertices, as depicted in Figure 4.7a. However, since neighbor squares share common vertices on their edges, such a description is highly redundant. Lindstrom *et al.* point out this problem in [58] and their solution requires to duplicate the shared information. In [71], the author seems to store only Cartesian vertices[6] and does not explain how quincunx vertices are handled.

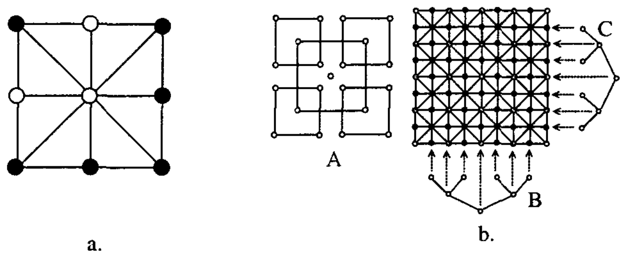


Figure 4.7: Storage of vertex characteristics avoiding redundancy: (a) Only the characteristics of the white vertices are stored in a quadtree node. (b) The full data structure storing the mesh without redundancy.

In order to avoid redundancy, we store only the characteristics of three vertices per node (the central vertex and two adjacent border vertices), for example the white vertices in Figure 4.7a. Therefore each node stores three floating point numbers plus three bits (each bit checks the vertex availability) to completely encode the geometry and the connectivity. The quadtree is sufficient to store all vertices characteristics but the ones on the eastern and southern borders. It suffices to use two additional binary trees to store the remaining vertices, as depicted in Figure 4.7b. Therefore, each node in the binary trees stores one floating point number plus one bit.

To access the information stored in the binary trees, we need to find the indices of the binary nodes given the characteristics of their sibling in the quadtree: Consider a quadtree node p at level i located at the eastern border, and having local interleaved coordinates $[g_h, g_v]$. The index

[6]See Section 3.2.1.

of its sibling binary node is given by

$$2^i - 1 + g_v. \tag{4.25}$$

Consider now a node p located at the southern border, then the index in this case is

$$2^i - 1 + g_h. \tag{4.26}$$

Example 4.4
For example, node 8 (level 2) has coordinates $[3, 1]$. Therefore, its sibling binary node has index 4 in tree C (Figure 4.7b). Similarly, node 16 has coordinates $[3, 3]$ and its sibling binary node has index 6 in tree B (Figure 4.7b).

4.6 Further constructions

Quadtrees describing more involved topologies than surfaces are very interesting in computer graphics. In this section, we give a construction for the octahedron. Then, we describe how we handle models with arbitrary topologies. Finally, we explain how we store large mesh patches using a forest of quadtrees.

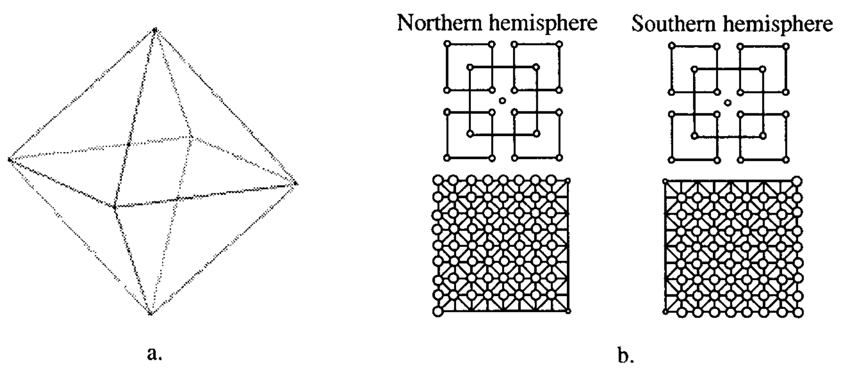


Figure 4.8: Octahedron: (a) The octahedron is handled seamlessly in our framework. (b) Each hemisphere is represented by a separate mesh, and the octahedron is obtained by merging them one on top of the other using the border vertices. The vertices stored for each hemisphere are represented with white dots.

Octahedron The storage of tessellations of the sphere is important to manage spherical datasets. For this purpose, triangle quadtrees have been developed for example in [56, 24, 32]. The management of spherical data is also achieved with square quadtrees such as the SLQ. A regular tessellation of the sphere is usually obtained by recursive subdivision of a platonic polyhedron having triangular faces. There are three such polyhedra: The tetrahedron (4 faces), the octahedron (8 faces) and the icosahedron (20 faces). In previous work [56, 24, 32], polyhedra are stored using a forest of quadtrees, where one quadtree is assigned per facet. The SLQ construction for surfaces is easily generalized to the octahedron (Figure 4.8a): Using two quadtrees, one per hemisphere of the octahedron, we obtain a construction having exactly the same traversal properties as surfaces. Moreover, no binary tree is required (Section 4.5) since two quadtrees suffice to store the information of all vertices without redundancy. In Figure 4.8b, we depict the vertices stored in each hemisphere quadtree. To traverse hemispheres, we use Equations (4.2) and (4.4).

Arbitrary topologies Storing arbitrary topology models using a set of quadtrees is equivalent to the problem of finding a parameterization for the mesh. The parameterization of arbitrary models can be performed using remeshing techniques [94].

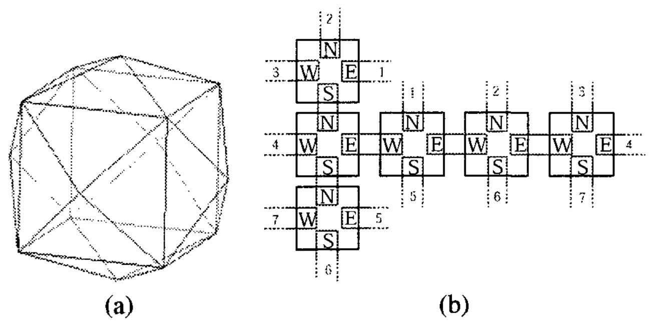


Figure 4.9: Tetrakishexahedron: (a) The tetrakishexahedron is a simple example of complex topology requiring side-information in order to traverse the faces. (b) The adjacency relations between the faces of the cube underlying the tetrakishexahedron. Each couple of integer denotes an adjacency pair.

A model with arbitrary topology can be stored using a forest of SLQ's. Each quadtree is assigned to a mesh patch and describes *a region of the model*, such as one quadtree is assigned per hemisphere in the case of the octahedron. However, the coherence of the forest must be maintained with additional side information, i.e. how adjacent patches are cleaved. We must know the cleaving of the patches in order to implement traversals between them. A simple example is given by the *tetrakishexahedron* depicted in Figure 4.9a. This polyhedron is simply the result of putting a 4-8 mesh on each face of a cube. The coherence is maintained as follows: For each adjacent face, their geometric orientation is recorded with a couple of cardinal points, for example (N,E), (E,E), etc...

Using (4.2) and (4.4) –the toroidal equations in Theorem 4.1–, the navigation between the adjacencies (E,W),(W,E),(N,S) and (S,N) is achieved. Consider now the pair (E,N), since the adjacency (W,E) is included in our framework, we can implement (E,N) by rotating the interleaved coordinates of the eastern border node by $\frac{\pi}{2}$. Then the index corresponding to the rotated coordinates is computed by inverting (4.14). We can now compute the difference between the index of the previous node in the traversal (or traversal path) and this index and continue the browsing using the rotated coordinates. Figure 4.9b depicts the adjacencies in the case of the tetrakishexahedron. In the figure, each couple of integer denotes an adjacency pair. Among the 12 adjacencies (i.e. edges) of the underlying cube, only 6 of them need a rotation before performing a traversal.

Handling large surface patches In order to process very large meshes, a patch is subdivided into a group of smaller patches. Therefore, a single patch is managed with a forest of quadtrees. In this case, no side-information is needed, unlike in the case of the tetrakishexahedron since the set of patches can be cleaved in a way such that we use only pair of adjacencies comprised in our framework. Figure 4.10 depicts traversals between components of the forest.

The support of *mesh partitioning* in our framework allows for *block-based approximation* of

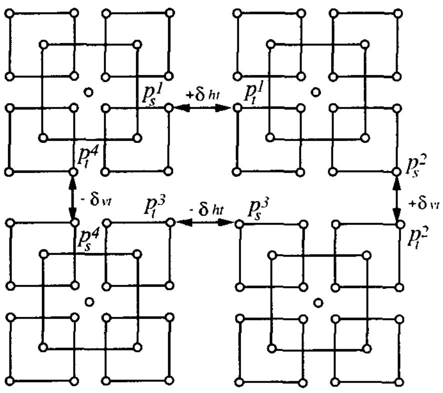


Figure 4.10: Data structure scalability to the navigation of a forest of quadtrees: The starting nodes are p_s^i and the target nodes are p_t^i, $i = 1 \ldots 4$. The index difference δ is the value to add to p_s^i to obtain p_t^i, i.e. $p_t^i = p_s^i + \delta$.

meshes as in [44], and meshes of arbitrary size can be handled.

4.7 Algorithms for 4-8 meshes

In this section, we present three algorithms for 4-8 meshes. We briefly explain how computationally optimal implementations can be obtained on the quadtree using the traversal path paradigm.

Restricted quadtree Recall the problem of computing a restricted quadtree after decimating an arbitrary vertex in the mesh (Section 3.2.3). Figure 4.11a unveils the quadtree structure storing the support of the merging domain of Figure 3.7: Intuitively, the mesh layer has been replaced by the quadtree used to store it. The dark region depicts the nodes traversed during the decimation. The arrows represent a proportion of the extensive traversal required in the quadtree to visit the vertices in the domain. According to Theorem 3.1, the cost of the computationally optimal algorithm is $\Theta(\log n)$. Using a traversal path, it is easy now to construct an algorithm to solve the problem exactly in this complexity.

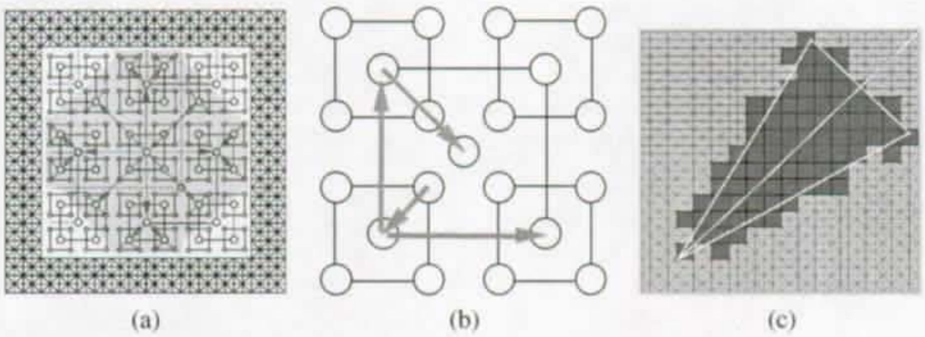


Figure 4.11: Applications on the quadtree: (a) Traversal of the merging domain when decimating an arbitrary vertex. (b) Finding an ancestor path, or computing the splitting domain of an arbitrary vertex. (c) Quick selection of the nodes containing data lying in the view frustum.

Ancestor path A computationally optimal implementation of the algorithm to find an ancestor path in Section 3.3.2 is also obtained using a traversal. Figure 4.11b depicts the path in the quadtree corresponding to Figure 3.9a. The problem of computing a restricted quadtree after inserting a vertex requires the same traversal path [58, 71]. The computationally optimal algorithm has complexity $\Theta(\log n)$ (Section 3.3.2).

Author	*Neighbor-finding complexity*		*Traversal/Traversal path of length l*	
	sparse tree	*dense tree*	*sparse tree*	*dense tree*
Our technique	$\Theta(\log m + 2n)$	$\Theta(2n)$	$\Theta(l \log m)$	$\Theta(l)$
Samet [75]	$\Theta(\log m + 3n)$	$\Theta(\log m + 3n)$	$\Theta(l \log m + 3ln)$	$\Theta(l \log m + 3ln)$
Gargantini [75]	$\Theta(\log m + 3n)$	$\Theta(\log m + 3n)$	$\Theta(l \log m + 3ln)$	$\Theta(l \log m + 3ln)$
Schrack [75]	$\Theta(\log m + 2n)$	$\Theta(\log m + 2n)$	$\Theta(l \log m + 2ln)$	$\Theta(l \log m + 2ln)$

Table 4.1: Summary of properties for the semi-linear quadtree. We assume location codes of length n and that m nodes are stored. The computational complexity to find a neighbor accounts for all the operations to obtain the node, i.e. conversion of the decimal expression of the location code, operations to find the neighbor location code, conversion of the quaternary digit expression and node retrieval in the quadtree. We assume traversals and traversal paths of length $l >> 1$. Finally, all complexities are given in expectation.

Visibility When processing large meshes, algorithms performing a quick selection of a subset of the vertices dramatically improve performances. Consider the problem of retrieving the visible part of a mesh: A simple algorithm aims at *rastering the quadtree structure*, as depicted in Figure 4.11c. In this figure, the view frustum is projected on the surface, as well as the direction of view. The direction of view and its perpendicular vector on the surface are used to rasterize the frustum. Using a traversal path, the algorithm is implemented with a computational complexity linear with the content of the field of view. Thus, the algorithm is computationally optimal.

4.8 Summary

In this chapter, we presented a framework to store and efficiently process 4-8 meshes stored in a quadtree data structure. We presented a new quadtree construction called *semi-linear quadtree*, which improves the linear quadtree in [75, 81, 29] both in storage and access performances. The best performances are obtained with a dense quadtree since few side information (i.e. *location codes*) is needed and constant-time access to the nodes is achieved on average. In this chapter, we presented the following results:

- A neighbor-finding method for the semi-linear quadtree outperforming all previous techniques in terms of computational complexity [75, 81, 29].
- A generalization of the concept of neighbor-finding to *traversal* and *traversal paths*.
- A technique to store and efficiently browse spherical datasets using a subdivided octahedron.

- ▶ A construction using a forest of semi-linear quadtrees to store and browse meshes with arbitrary topology, e.g. subdivision surfaces using 4-8 subdivision.

Also, our framework allows the processing of large meshes using *partitioning*. In Table 4.1, We compare our results to the main contributions in the field [75, 81, 29]. Finally, we explained how to use our results to construct computationally optimal algorithms on quadtrees. In particular, we gave computationally optimal implementations for two algorithms studied in Chapter 3: the computation of merging domains (Section 3.3.1) and the construction of ancestor paths (Section 3.3.2). We use these implementations in the surface simplification algorithm presented in Chapter 5.

Appendix 4.A Proofs

4.A.1 Proof of Theorem 4.1

To find the distance between nodes in closed-form, we need to write the general expression of pairs of nodes' indices for which the index differences are searched. Figure 4.12 illustrates the technique to find the distance between two horizontal neighbor nodes: At the tree level 1, nodes $4p + 1$ and $4p + 2$ are two horizontal neighbors. At level 2, the neighbors are respectively the nodes with indices $4(4p + 1) + 2$ and $4(4p + 2) + 1$. For an arbitrary level $r + 1$, the indices, denoted respectively by p_w and p_e (where w and e stand for *west* and *east*), are given by the recurrence equations

$$\begin{aligned} p_w(0) &= 4p + 1, \\ p_w(r) &= 4p_w(r-1) + 2, r > 0, \end{aligned} \tag{4.27}$$

and

$$\begin{aligned} p_e(0) &= 4p + 2, \\ p_e(r) &= 4p_e(r-1) + 1, r > 0, \end{aligned} \tag{4.28}$$

which have the solutions, respectively,

$$p_w(r) = 4^{r+1}p + \frac{5}{3}4^r - \frac{2}{3}, \tag{4.29}$$

$$p_e(r) = 4^{r+1}p + \frac{7}{3}4^r - \frac{1}{3}. \tag{4.30}$$

As an example, note that when using the scalar product notation in (4.23), (4.29) corresponds to the scaling function

$$p_w(r) = \left[4^{r+1}4^r \ldots 1\right] \begin{bmatrix} p \\ 1 \\ 2 \\ \vdots \\ 2 \end{bmatrix}. \tag{4.31}$$

In other words, the index $p_w(r)$ is found by traversing one child node at location 1, then $r - 1$ child nodes at location 2. The distance in closed-form is obtained by computing $p_e(r) - p_w(r)$, i.e.

$$\delta_h(r) = \frac{2}{3}4^r + \frac{1}{3}. \tag{4.32}$$

We can verify that $\delta_h(1) = 1$, $\delta_h(2) = 3$, $\delta_h(3) = 11$, etc... as shown in Figure 4.5.

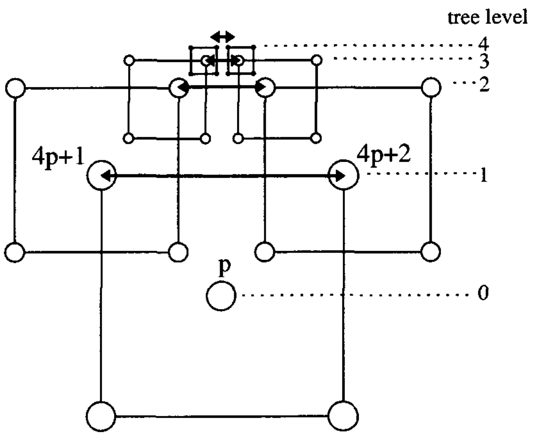


Figure 4.12: Illustration of the technique to find the index differences in closed-form. In this example, we show how to find the horizontal distance between two arbitrary nodes.

To find δ_{ht}, the same technique is used with the recurrence equations:

$$\begin{aligned} p_w(0) &= 4p+1, \\ p_w(r) &= 4p_w(r-1)+1, r>0, \end{aligned} \tag{4.33}$$

and

$$\begin{aligned} p_e(0) &= 4p+2, \\ p_e(r) &= 4p_e(r-1)+2, r>0, \end{aligned} \tag{4.34}$$

which have the solutions, respectively,

$$p_w(r) = 4^{r+1}p + \frac{4}{3}4^r - \frac{1}{3}, \tag{4.35}$$

$$p_e(r) = 4^{r+1}p + \frac{8}{3}4^r - \frac{2}{3}. \tag{4.36}$$

The distance in closed-form is obtained by computing $p_e(r) - p_w(r)$, i.e.

$$\delta_{ht}(r) = \frac{1}{3}4^{r+1} - \frac{1}{3}. \tag{4.37}$$

We can verify that $\delta_{ht}(0) = 1$, $\delta_{ht}(1) = 5$, $\delta_{ht}(2) = 21$, etc... as shown in Figure 4.5.

For the vertical distances $\delta_v(r)$ and $\delta_{vt}(r)$, we denote the indices of two vertical neighbor nodes by p_n and p_s, respectively (where the subscripts n and s stand for *north* and *south*). For $\delta_v(r)$, we use the recurrence equations

$$\begin{aligned} p_n(0) &= 4p + 1, \\ p_n(r) &= 4p_n(r-1) + 4, r > 0, \end{aligned} \tag{4.38}$$

and

$$\begin{aligned} p_s(0) &= 4p + 3, \\ p_s(r) &= 4p_s(r-1) + 2, r > 0, \end{aligned} \tag{4.39}$$

which have the solutions, respectively

$$p_n(r) = 4^{r+1}p + \frac{7}{3}4^r - \frac{4}{3}, \tag{4.40}$$

$$p_s(r) = 4^{r+1}p + \frac{11}{3}4^r - \frac{2}{3}. \tag{4.41}$$

The distance in closed-form is obtained by computing $p_s(r) - p_n(r)$, i.e.

$$\begin{aligned} \delta_v(r) &= \frac{1}{3}4^{r+1} + \frac{2}{3}, \\ &= 2\delta_h(r). \end{aligned} \tag{4.42}$$

Finally, for $\delta_{vt}(r)$, we use the recurrence equation

$$\begin{aligned} p_s(0) &= 4p + 3, \\ p_s(r) &= 4p_s(r-1) + 3, r > 0, \end{aligned} \tag{4.43}$$

and 4.33 for $p_n(r)$. Therefore, the solutions are given by

$$p_n(r) = 4^{r+1}p + \frac{1}{3}4^{r+1} - \frac{1}{3}, \tag{4.44}$$

$$p_s(r) = 4^{r+1}p + 4^{r+1} - 1, \tag{4.45}$$

and the distance $p_s(r) - p_n(r)$ is given by

$$\begin{aligned} \delta_{vt}(r) &= \frac{2}{3}4^{r+1} - \frac{2}{3}, \\ &= 2\delta_{ht}, \end{aligned} \tag{4.46}$$

which concludes this proof. □

Chapter 5

Progressive meshes in an operational rate-distortion sense

5.1 Introduction

5.1.1 Motivation

In Section 2.6, we presented an optimal tree-constrained decimation algorithm for polylines. This algorithm has been reported to be the only one fulfilling all the requirements for our transmission framework (Table 2.3). In particular, the generated approximations are progressive and achieve error monotonicity across rate. Moreover, the error estimate computed by the algorithm is global. Finally, the algorithm is computationally efficient and achieves the best solution quality in the constrained case. In this chapter, we generalize this algorithm to 4-8 meshes approximation. Recall that our algorithm is derived from the optimal tree pruning algorithm given in [9, 13]. The latter implementation is used to compute adaptive quantizers for compression.

We aim at processing large polygonal meshes. Then, we reuse the results of Chapter 3 to efficiently decimate the dataset and update the vertex errors. In particular, we use the algorithm given in Section 3.4.4, computing merging domain intersections, to optimize the mesh using global error. Our algorithm decimates a 4-8 mesh stored in a quadtree. We implement the decimation of merging domains and the update of vertex errors using the quadtree implementation presented in Chapter 4. Finally, note that previous work in simplification for 4-8 meshes has been previously reviewed in Section 3.1.2. However, we summarize the key issues when processing large meshes in Section 5.2.1.

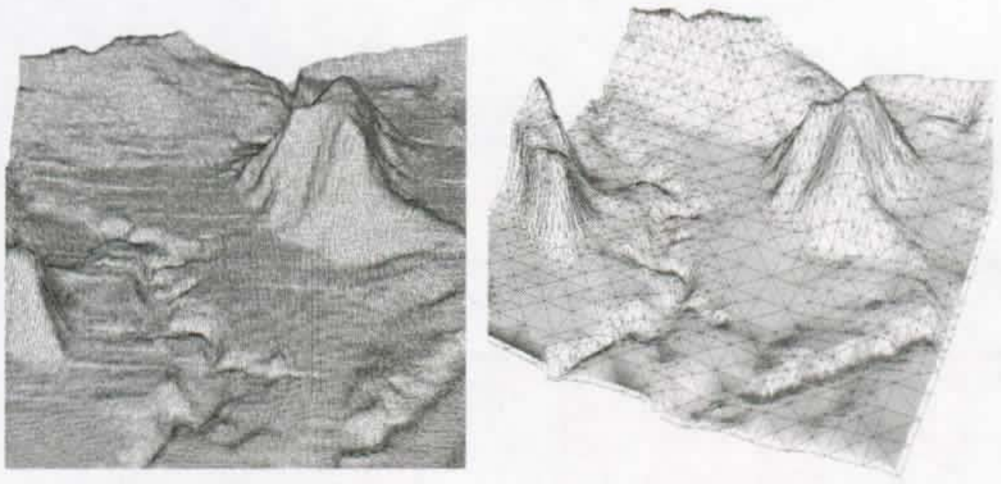


Figure 5.1: Surface approximation: (a) The original mesh contains 131072 triangles. (b) An approximation using 6000 triangles.

5.1.2 Contributions and plan

Efficient simplification algorithm We propose an $\Theta(n \log n)$ algorithm to produce adaptive representations of 4-8 meshes using general decimation and global error. The algorithm decomposes a mesh into a control mesh plus a series of detail meshes. Global error metrics yield better approximation quality than heuristics based on local error, but are often computationally expensive. In comparison, our algorithm using global error is computationally efficient. We shows that a direct approach using the same error criterion requires at least $\Theta(n^2)$ time. Also, decimation approaches yield better results than their refinement counterparts. In particular, general decimation is usually needed to obtain optimal solutions [31].

Rate-distortion framework and global error We study our mesh simplification algorithm in an operational Rate-Distortion (RD) framework. We attach decimation costs to each vertex. More precisely, we measure a cost in rate, given in term of triangles, and a cost in distortion, computed in l_2 norm with respect to the original mesh. We give an $\Theta(\log^2 n)$ algorithm to maintain global characteristics for the vertices during the optimization process. This algorithm is based on a inclusion-exclusion principle used to compute *merging domain intersections* presented in Chapter 3.

Discussion on optimality We discuss the optimality of the solutions and analyze how optimal vertices are chosen at each decimation step. Although we do not prove the optimality of the algorithm, we use our results to show that approximation errors are most of the time monotonic across rate. We prove that, under certain assumptions, monotonicity is achieved. We explain that suboptimal cases leading to nonmonotonicities exist. However, we show experimentally that the approximation errors returned by our algorithm behave almost always monotonically across rate. We compare our algorithm to its restricted counterpart, i.e. we force the algorithm to perform restricted decimation only, and show that monotonicity is no more conserved in this case.

Minimal metric assumptions We show that our algorithm makes minimal assumptions on the mesh functionals. Recall that l denote the interpolation step of a vertex. Such a value is assigned to each vertex during the 4-8 mesh construction depicted in Figure 3.1a-e. Then, we prove that our algorithm only requires the rate functional to be monotonically decreasing with l while the distortion functional can be arbitrary decreasing, i.e. non-monotonic or monotonic. This feature leaves the greatest freedom to elaborate an optimization metric tailored to the application.

Experimental results We show experimentally the superiority of using global error for the vertices over approaches based on local error. Then, we apply our algorithm to a database of 388 terrains [16] and give approximation results and timings. Timings are given for terrains containing up to two millions triangles.

The chapter is organized as follows: In Section 5.2, we introduce our approach: We present our framework and give the algorithm. We analyze the algorithm in Section 5.3: More precisely, in Section 5.3.1, we explain the update method used to maintain global characteristics for the vertices. We evaluate the complexity of the algorithm in Section 5.3.2. We discuss the optimality of the solutions in Section 5.4 and give experimental results in Section 5.5. We summarize the results in this Chapter in Section 5.6.

5.2 Algorithm description

We reviewed the constraints attached to the simplification of 4-8 meshes in Chapter 3. In this section, we describe our algorithm and framework in the following sections. This section is organized as follows: First, we review the key issues when processing large meshes in Section 5.2.1. Second, we pose the central problem and present our solution in Section 5.2.2.

5.2.1 Key issues when processing large meshes

We summarize below the key issues to address when simplifying large meshes: Having an efficient data structure is essential, since many processing steps strongly rely on the ability to efficiently query the dataset. Examples include the recomputation of the connectivity and the update of the errors after a simplification step. An efficient data structure must then provide:

- Information on the spatial orientation of the mesh.
- Fast and simple access to the dataset.
- Minimal storage.

Moreover, the efficiency of the data structure is tightly coupled with the mesh connectivity. For example, a mesh with regular connectivity lowers the storage complexity and allows a better design of data access mechanisms. The semi-linear quadtree (SLQ) presented in Chapter 4 fulfills all the above requirements. For the rest of this chapter, we will refer to the SLQ simply as quadtree when mentioning our work.

Meshes with regular connectivity does not usually achieve the quality of irregular triangulations (see Figure 2.17b), but provides a more flexible framework in terms of optimization and efficiency. The processing of large meshes requires scalable algorithms, as described in [44], with low computational complexity. The algorithms must also be able to handle multiple error metrics. Sophisticated error metrics including, for example, the attributes of the vertices (such as color, shading, texture) should be considered. Our framework is built under these considerations.

5.2.2 Central problem and solution

This section introduces an $\Theta(n \log n)$ algorithm based on general decimation and global error. We show in Section 5.3.2 that it computationally outperforms a direct approach using the same error criterion. We apply it to a mesh built on a matrix of amplitudes z, e.g. terrain data. We use *mesh functionals* $u : M_v \rightarrow \mathbb{R}$ to compute properties for v *over its merging domain* M_v. We use two mesh functionals R and D: R is called the *rate* and counts the number of triangles, whereas D measures the distance in l_2 norm between the original surface and an approximation (Appendix 5.A). Hence, for each v we compute the vector value $\mathrm{u}(M_v) = (R(M_v), D(M_v))$.

Call $M_0 = \{v_0, \ldots, v_{N-1}\}$ the input mesh and M a simplified version, then the problem to solve is

$$D(R) = \min_{|M| \leq |M_0|} \{D(M) | R(M) \leq r\}, \tag{5.1}$$

where r denotes a constraint in rate. A progressive representation for M is found by solving the problem for all values $2 \geq r \geq n$. For a rate budget r, the solution $(R(M_i), D(M_i))$ returned by $D(R)$ satisfies the constraint at minimal incurred distortion. The set of solutions, denoted by

$$|\mathcal{B}| < \ldots < |M_1| < |M_0|, \tag{5.2}$$

where $\mathcal{B}$ is the control mesh, corresponds to a series of embedded approximations. The solutions are embedded in the sense that any approximation can be reconstructed from a coarser solution only by splitting a set of triangles.

Each simplified mesh $(R(M), D(M))$ can be represented as a position in the space of values spanned by R and D. This space is called *rate-distortion (RD) plane* (Figures 5.2a-d). The set of all possible approximations is a cloud of positions in the RD plane. Each optimal configuration is represented by a position $\mathbf{u}(M_i) = (R(M_i), D(M_i))$ on the curve bounding the convex hull of all configurations (Figure 5.2d). This curve is called the *operational RD curve* and the approximations on this curve are optimal in the operational RD sense.

We define the *variation* of a functional as

$$\Delta\mathbf{u}(M_v) = \mathbf{u}(M_v) - \mathbf{u}(\check{M}_v). \tag{5.3}$$

Hence, $\Delta\mathbf{u}(M_v) = (\Delta R(M_v), \Delta D(M_v))$. The variation $\Delta\mathbf{u}(M_v)$ is the change in rate and distortion when M_v is decimated. Therefore, a vector $\Delta\mathbf{u}(M_v)$ links two configurations in the RD plane. More precisely, given a mesh over which $\Delta\mathbf{u}(M_v)$ is computed, the vector leads to the configuration obtained by decimating M_v. Hence, $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is the trade-off between rate and distortion when M_v is decimated and represents a slope in the RD plane (Figure 5.2a).

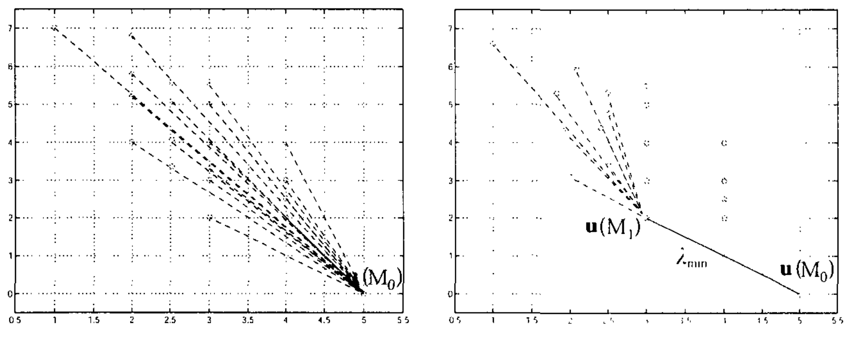


Figure 5.2: Algorithm: (a) Initially, the variations $\Delta\mathbf{u}(M_v)$ and the slopes $\lambda(v)$ are computed for each vertex. (b) The vertex with minimal slope $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is chosen and decimated. The RD characteristics of the ancestor vertices of M_v are updated, hence the corresponding positions in the RD plane are displaced.

The algorithm proceeds as follows: Initially, the variations $\Delta\mathbf{u}(M_v)$ and the slopes $\lambda(v)$ are computed (Figure 5.2a) and stored for each vertex. Note that $\Delta D(M_v) < 0$ (Appendix 5.A), hence $\lambda(v) > 0$. Additionally, we use a value $\lambda_{\min}$ at each vertex to store the minimal slope amongs all its descendants. At each iteration the vertex v with minimal $\lambda(v)$ is chosen and M_v is decimated (Figure 5.2b). The decimation changes the characteristics (i.e. in rate and distortion) of a set of vertices. We call these vertices *ancestors* and denote this set by A_{M_v}. Two types of ancestors a exist: the vertices such that $M_v \subset M_a$ and the vertices such that M_v and M_a partially overlap. In Chapter 3, we explain how to find these vertices efficiently. In particular, we prove that

$$|A_{M_v}| \in \Theta(\log n). \tag{5.4}$$

Also, we show that $\Theta(\log^2 n)$ operations are sufficient to update all the ancestor values. We explain our update mechanism in Section 5.3.1. Once the RD characteristics of the ancestor vertices are updated, the corresponding positions $\mathbf{u}(M_a)$ in the RD plane are displaced. The algorithm is iterated until the configuration with minimal rate is reached (Figures 5.2c-d).

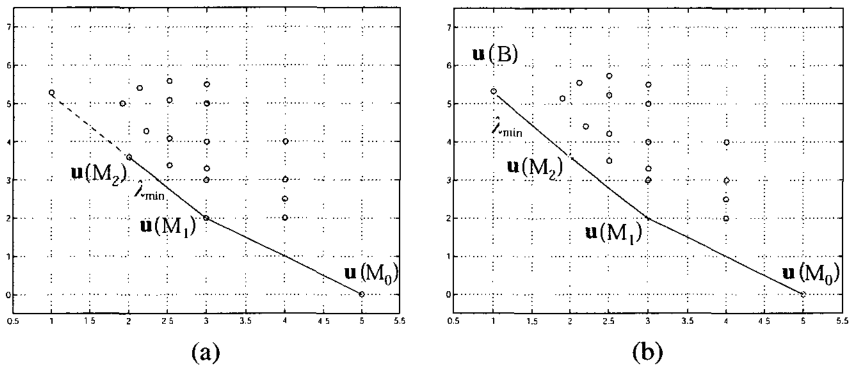


Figure 5.3: Algorithm: (a),(b) The algorithm is iterated. The algorithm aims at the solutions on the curve lowerbounding the set of all possible configurations. These approximations are optimal in the operational RD sense.

We give the algorithm below. In our application, we use an 4-8 interpolated matrix of amplitudes and the configuration with minimal rate has two triangles and is shown in Figure 3.1a. It's important to note that M_{v_0} contains all the vertices in the mesh. Therefore, since our characteristics are global, the global rate and the global distortion are given by $R(M_{v_0})$ and $D(M_{v_0})$, respectively. Hence, in line 7 we use $R(M_{v_0})$ to test the rate of the current approximation. Similarily, we could use $D(M_{v_0})$ to obtain configurations satisfying a maximum error. The total complexity of the algorithm is computed in Section 5.3.2.

ALGORITHM

```
1 initialization:
    2 for all $v$
        3 COMPUTE $\Delta D(M_v)$, $\Delta R(M_v)$
        4 $\lambda(v) \leftarrow \frac{-\Delta D(M_v)}{\Delta R(M_v)}$
5 iteration:
    6 $i = 1$          (counter for the approximations.)
    7 while $R(M_{v_0}) > 2$
        8 $v^\star = \arg\min_{v \in M} \lambda(v)$.
        9 $M_i \longleftarrow M_{i-1} \setminus M_{v^\star}$.
        10 UPDATE $\Delta D(M_a)$ AND $\Delta R(M_a)$.
    11 end
12 end
```

5.3 Analysis

5.3.1 Update of global error

In this section, we present the algorithm used to update the functional variations of the vertex characteristics. The algorithm has cost $\Theta(\log^2 n)$ and is derived from an algorithm computing merging domain intersections presented in Chapter 3. Assume that a vertex M_v is decimated, then the variations $\Delta \mathbf{u}(M_a)$ are replaced by

$$\Delta \mathbf{u}(M_a) - \Delta \mathbf{u}(M_w), \tag{5.5}$$

where $w \in M_v$ and $a \in A_{M_v}$. We explain below how to find the variations $\Delta \mathbf{u}(M_a)$ updated with $\Delta \mathbf{u}(M_w)$.

We use the algorithm below to update the functional variations computed at the initialization (lines 1-4 of the algorithm in Section 5.2.2) during the mesh optimization. In the algorithm, the updated ancestor functionals are denoted $\Delta \mathbf{u}'(M_a)$. The algorithm finds the set of ancestors A_{M_v} and updates the characteristics using (5.5). More precisely, a set of *parents* (Section 3.3.2) for each vertex $w \in M_v$, denoted by A_w, is traversed (see Chapter 3 for details). The parents

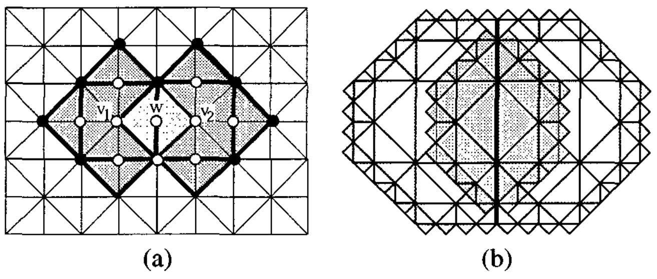


Figure 5.4: Update of the global error: (a) When M_{v_1} and M_{v_2} are decimated, the global characteristics (rate and distortion) of the mesh are $\mathbf{u}(M_{v0}) - \Delta\mathbf{u}(M_{v_1} \cup M_{v_2})$. (b) Intersection between two large merging domains. For clarity, the merging domains are represented using their support. The intersection is depicted in dark shade and the thick line represents the boundary between the domains.

are found using a bottom-up traversal of the mesh. An example of traversal is shown in Figure 5.5. The index next to each vertex is the interpolation step l (Figures 3.1a-e). The update (5.5) is performed for each parent. Once the set of parents for w has been visited, w is decimated. Hence the algorithm is used to update the ancestor variations and decimate the domain M_v, and replaces lines 9 and 10 of the algorithm given in Section 5.2.2. Recall that an important property of 4-8 meshes (3.28), related to the parents of a vertex, is

$$\forall w \in M_v, v \in A_w, \tag{5.6}$$

i.e. all the vertices $w \in M_v$ have v as a parent. Assume that v is interpolated at step l, then the set of vertices $w \in M_v$ has to be visited starting from the vertices in the domain having the largest interpolation step. Therefore, if M_v spreads through m levels, then first the available vertices with step $l + m$ are decimated, then the vertices with step $l + m - 1$, and so on.

ALGORITHM

```
1 for ALL AVAILABLE VERTICES $w \in M_v$ INTERPOLATED AT STEP $l + m \ldots l$
    2 for ALL $a \in A_w$
        3 $\Delta\mathbf{u}'(M_a) = \Delta\mathbf{u}(M_a) - \Delta\mathbf{u}(M_w)$
    4 end
    5 decimate $w$
```

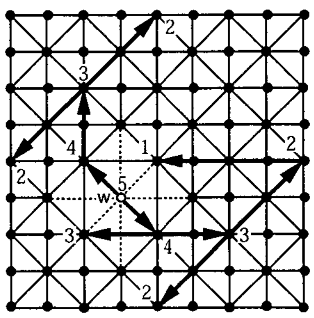


Figure 5.5: Parents of vertex w (A_w): To find the parents of w, interpolated at $l = 5$, the mesh is traversed bottom-up (see Chapter 3 for details). The index next to each vertex indicates the interpolation step.

```
6 end
```

We explain now how global characteristics are maintained using the above algorithm. We start with a simple example, then we address the general case. To do so, we summerize the problem of finding merging domain intersections. A complete analysis of this problem is found in Chapter 3.

After the initialization phase (lines 1-4 of the algorithm in Section 5.2.2), the functional values $\Delta\mathbf{u}(M_v)$ are global since no vertex has been yet decimated. Consider M_{v_1} and M_{v_2} as depicted in Figure 5.4a. Clearly, after decimating both domains, the global characteristics of the mesh (rate and distortion) are

$$\mathbf{u}(M_{v_0}) - \Delta\mathbf{u}(M_{v_1} \cup M_{v_2}), \tag{5.7}$$

where v_0 denotes the root vertex. Recall that this vertex can be used to measure the characteristics of the complete mesh since M_{v_0} contains all the vertices. Unfortunately, $M_{v_1} \cap M_{v_2} \neq \emptyset$ thus

$$\Delta\mathbf{u}(M_{v_1} \cup M_{v_2}) < \Delta\mathbf{u}(M_{v_1}) + \Delta\mathbf{u}(M_{v_2}). \tag{5.8}$$

However, $M_w \subset M_{v_1} \cup M_{v_2}$, as shown in Figure 5.4a, and the surplus of $\Delta\mathbf{u}(M_{v_1}) + \Delta\mathbf{u}(M_{v_2})$ is $\Delta\mathbf{u}(M_w)$ because every triangle tiling the support of M_w is also a triangle of either the support of M_{v_1} or M_{v_2}. Hence, we have

$$\Delta\mathbf{u}(M_{v_1} \cup M_{v_2}) = \Delta\mathbf{u}(M_{v_1}) + \Delta\mathbf{u}(M_{v_2}) - \Delta\mathbf{u}(M_w). \tag{5.9}$$

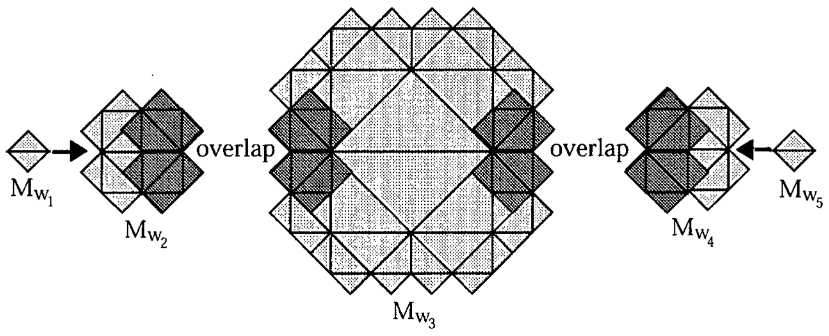


Figure 5.6: Merging domain intersections: Decomposition of the intersection in Figure 5.4b: The intersection is expressed as the union of a set of smaller domains M_{w_i}, $i = 1 : 5$. Note that couples of domains M_{w_2},M_{w_3} and M_{w_3},M_{w_4} overlap.

We decimate now successively M_{v_1} and M_{v_2}. For M_{v_1}, The algorithm first decimates w and the three remaining vertices (depicted in white in Figure 5.4a) interpolated at the same step. Hence, following (5.6), the updated functional characteristics at v_1, v_2 and v_0 (root vertex) are, respectively,

$$\begin{aligned}
&\Delta\mathbf{u}(M_{v_1})-\Delta\mathbf{u}(M_w) - \sum_{k\in M_{v_1},k\neq v_1,k\neq w} \Delta\mathbf{u}(M_k),\\
&\Delta\mathbf{u}(M_{v_2})-\Delta\mathbf{u}(M_w),\\
&\Delta\mathbf{u}(M_{v_0})-\Delta\mathbf{u}(M_w) - \sum_{k\in M_{v_1},k\neq v_1,k\neq w} \Delta\mathbf{u}(M_k).
\end{aligned} \tag{5.10}$$

Then, the algorithm decimates v_1 and the updated value at v_0 is

$$\Delta\mathbf{u}(M_{v_0}) - \Delta\mathbf{u}(M_{v_1}). \tag{5.11}$$

Note that v_2 is not affected by the decimation of v_1 since $v_1 \notin A_{v_2}$. Now M_{v_2} is decimated (starting with the three available vertices $k \in M_{v_2}, k \neq w$), and the updated values at v_2 and v_0 are, respectively,

$$\begin{aligned}
&\Delta\mathbf{u}(M_{v_2})-\Delta\mathbf{u}(M_w) - \sum_{k\in M_{v_2},k\neq v_2,k\neq w} \Delta\mathbf{u}(M_k),\\
&\Delta\mathbf{u}(M_{v_0})-\Delta\mathbf{u}(M_{v_1}) - \sum_{k\in M_{v_2},k\neq v_2,k\neq w} \Delta\mathbf{u}(M_k).
\end{aligned} \tag{5.12}$$

Finally, the algorithm decimates v_2 and the updated value at v_0 is

$$\Delta\mathbf{u}(M_{v_0}) - \Delta\mathbf{u}(M_{v_1}) - \Delta\mathbf{u}(M_{v_2}) + \Delta\mathbf{u}(M_w), \tag{5.13}$$

which shows that (5.9) is obtained, i.e. the characteristics computed at v_0 are global.

The above example shows that the algorithm uses an inclusion-exclusion principle to compute the global error at each vertex. We presented a simple example where the intersection between the decimated domains M_{v_1} and M_{v_2} is the singleton domain $M_w = \{w\}$. In general, intersections between domains are more complex. For example, consider the intersection between the two domains in Figure 5.4b. In the figure, the domains are depicted using their support for clarity. Following (5.4), we have $\Theta(\log n)$ possible arrangements for intersections. Hence the examples in Figures 5.4a and 5.4b are just particular cases. Figure 5.7 depicts another example of arrangement.

The intersection in Figure 5.4b can be decomposed in terms of smaller merging domains, as shown in Figure 5.6. The number of domains is proportional to the size of the intersection. In the example of Figure 5.4a, a single domain M_w is sufficient to express the intersection, whereas in Figure 5.6, the intersection is written

$$M_{w_1} \cup M_{w_2} \cup M_{w_3} \cup M_{w_4} \cup M_{w_5}. \qquad (5.14)$$

Unfortunately, the domains M_i forming the intersection overlap (Figure 5.6), i.e. $M_{w_2} \cap M_{w_3} \neq \emptyset$ and $M_{w_3} \cap M_{w_4} \neq \emptyset$. Therefore to compute the union (3.32), we use an inclusion-exclusion approach to resolve all embedded intersections. In Chapter 3, we call this problem *merging domain intersections* and we show that the union (3.32), i.e. *exclusive intersection*, can be computed in $\Theta(\log n)$ time. Also, we show that exclusive intersections between a decimated domain M_v and the domains of all its ancestors A_{M_v} are computed in $\Theta(\log^2 n)$ time.

The update algorithm given at the beginning of this section automatically computes all exclusive intersections after decimating a domain M_v. The computed values at each vertex are the global RD characteristics to substract to $\mathbf{u}(M_{v_0})$ after decimating the vertex. Hence, the characterstics $\mathbf{u}(M_{v_0})$, i.e. at the root vertex, are the global characteristics of the mesh.

We conclude now this section with the following general example: Assume that all vertices in M_v are decimated except v. Therefore, following (5.6) the updated variations at v and v_0 are, respectively,

$$\begin{array}{c} \Delta\mathbf{u}(M_v) - \sum_{w \in M_v, w \neq v} \Delta\mathbf{u}(M_w), \\ \Delta\mathbf{u}(M_{v_0}) - \sum_{w \in M_v, w \neq v} \Delta\mathbf{u}(M_w). \end{array} \qquad (5.15)$$

Assume that v is now decimated, then using (5.5), the variation at v_0 is now

$$\Delta\mathbf{u}(M_{v_0}) - \Delta\mathbf{u}(M_v), \qquad (5.16)$$

which corresponds to the global characteritics of the mesh after the decimation of M_v.

5.3.2 Complexity

The cost of the algorithm in Section 5.2.2, i.e. computing a complete decomposition of the mesh, is found as follows: In Chapter 3, we show that merging domains have size $\Theta(\log n)$ on average, thus assuming a mesh of n triangles, the initialization has cost $\Theta(n \log n)$. At each iteration, the optimal vertex $v^\star$ (having minimal slope $\lambda(v^\star)$) is found in $\Theta(\log n)$ operations using the values λ_{min}. The cost to decimate M_v and update the variations for the vertices in A_{M_v} is $\Theta(\log^2 n)$. Also, $\Theta(\log n)$ values $\lambda(v)$ and λ_{min} are recomputed and the algorithm is iterated. On average, $n/\Theta(\log n)$ steps are necessary to decompose the mesh, since at each step, $\Theta(\log n)$ vertices on average are decimated (average size of merging domains). Hence, the cost to compute the full decomposition is $\Theta(n \log n)$.

A direct algorithm needs to recompute the global error over each ancestors' domain. A lowerbound for this update is obtained as follows: We have roughly $\Theta(4^{l+1})$ vertices at step l and $l \in \Theta(\log n)$ ancestors exists. Call a any such ancestor, then $|M_a|_\triangle(i,n) \approx n/4^{i-1}$, $1 \leq i \leq l$. Therefore, a lowerbound for the complexity is

$$\sum_{i=0}^{\log_4 n} 4^i \sum_{j=0}^{i} \frac{n}{4^j} = \frac{4}{9}n^2 - \frac{1}{3}n \log_4 n - \frac{7}{9}n \in \Theta(n^2). \tag{5.17}$$

Note the above approximation accounts only for the ancestors a such as $M_v \subset M_a$. Accounting for the update of the ancestors whose domain partially overlaps does not change the order of magnitude. However, this evaluation is complex due to the $\Theta(\log n)$ cases of overlap, i.e. arrangements for intersections, one has to deal with (Section 5.3.1).

We conclude our analysis of the algorithm with the following proposition:

Proposition 5.1 *On average, an algorithm based on global error and using general decimation requires $\Theta(n \log n)$ operations to fully decompose a 4-8 mesh with n triangles when merging domain intersections are used to update the vertex errors.*

5.4 Discussion of optimality

In this section, we discuss the optimality of the algorithm. First, we explain how an optimal vertex is chosen at each decimation step (Section 5.4.1). Second, we discuss issues related to intersections between domains and how optimal choices are affected (Section 5.4.2). Third, we explain how the monotonicity of the approximation errors (i.e. distortions) are conserved across rate (Section 5.4.3).

5.4.1 Optimal choice

Recall that our rate functional measures the number of triangles, hence the functional is monotonically increasing with the mesh size. Consider now the following example: Consider two vertices v_1 and v_2 such that $v_2 \in M_{v_1}$, i.e. $\Delta R(M_{v_2}) < \Delta R(M_{v_1})$. Furthermore, assume that

$$\Delta D(M_{v_2}) > \Delta D(M_{v_1}). \tag{5.18}$$

Such a case is possible with the l_2 or the l_∞ norms since both are nonmonotonic with the mesh size [41]. Recall that $\Delta D(M_{v_1}) < 0$ and $\Delta D(M_{v_2}) < 0$ (Section 5.2.2). If v_2 is decimated, then following (5.5) and (5.6), we have that

$$\Delta D(M_{v_1}) - \Delta D(M_{v_2}) > 0, \tag{5.19}$$

and the new slope

$$\lambda(v_1) = \frac{-(D(M_{v_1}) - D(M_{v_2}))}{(\Delta R(M_{v_1}) - \Delta R(M_{v_2}))} < 0, \tag{5.20}$$

i.e. the sign of the slope changes. In consequence, M_{v_1} will be the optimal domain to decimate at the next iteration, and the global error will decrease, i.e the RD curve will be nonmonotonic. We say that M_{v_1} is a *nonmonotonic merging domain* with respect to M_{v_2}, i.e. decimating M_{v_2} creates a nonmonotonicity at v_1.

The algorithm avoids the above situation using general decimation as follows: If the decimation of a domain M_v provokes a nonmonotonicity at a parent of v, then the algorithm will decimate the domain of the parent instead. In Proposition 5.2, we show that only the rate functional needs to be monotonic and that the distortion functional can be arbitrary (i.e. monotonic or nonmonotonic), both with respect to the mesh size, in order for the algorithm to make the optimal choice.

Proposition 5.2 *Given v_1 and v_2, such that $v_2 \in M_{v_1}$, and $\Delta R(M_v) \geq 0$ (monotonicity of the rate functional), then M_{v_2} is decimated before M_{v_1} if and only if*

$$\frac{\Delta D(M_{v_1})}{\Delta D(M_{v_2})} > \delta > 1, \tag{5.21}$$

where $\delta = \Delta R(M_{v_1})/\Delta R(M_{v_2})$. When v_1 does not meet condition (5.21), the domain M_{v_1} is said to be nonmonotonic with respect to M_{v_2}.

Proof. For M_{v_2} to be decimated before M_{v_1}, we need to have

$$\Delta D(M_{v_1})\Delta R(M_{v_2}) > \Delta R(M_{v_1})\Delta D(M_{v_2}). \tag{5.22}$$

Since the functional R is monotonically increasing, we can write

$$\Delta R(M_{v_1}) = \delta \Delta R(M_{v_2}), \delta > 1 \tag{5.23}$$

Then, replacing (5.23) in (5.22) yields

$$\Delta D(M_{v_0}) > \delta \Delta D(M_{v_1}). \tag{5.24}$$

□

5.4.2 Intersection between domains and optimal choice

In Section 5.2.2, we explained that a type of ancestors is the vertices a such that M_v and M_a partially overlap. Figure 5.7 illustrates such a case. Assume now that M_v is the optimal domain to decimate at some iteration and that M_a is nonmonotonic with respect to M_w. Since $w \in M_v$, M_w is decimated jointly to M_v. Following (5.6), the decimation creates a nonmonotonicity at M_a. The above example shows that, due to the overlap between domains, the algorithm using general decimation cannot avoid nonmonotonicities across rate.

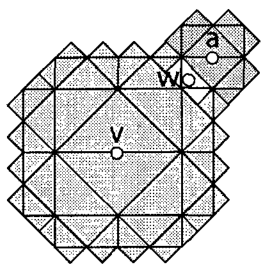


Figure 5.7: Suboptimal choice of the algorithm: $w \in M_v \cap M_a$ and M_a is a nonmonotonic merging domain with respect to M_w (see Proposition 5.2). Decimating M_w provokes a nonmonotonicity at M_a.

We perform experiments using matrices of amplitudes z (terrain data [16]) and compare our algorithm to its restricted counterpart, i.e. we force the algorithm to perform restricted decimation only (Section 3.2.3). Hence, only singleton domains M_v, i.e. $|M_v| = 1$, are decimated, preventing the algorithm to make optimal choices as explained in Section 5.4.1. We find that, although the algorithm using general decimation cannot avoid monotonicity, the RD curve (top curve in Figure 5.8) is very stable compared to the one obtained with its restricted counterpart (bottom curve in Figure 5.8). For the restricted version, nonmonotonicities often occur at low rate. We conclude that, for our dataset, few cases of nonmonotonicity are observed.

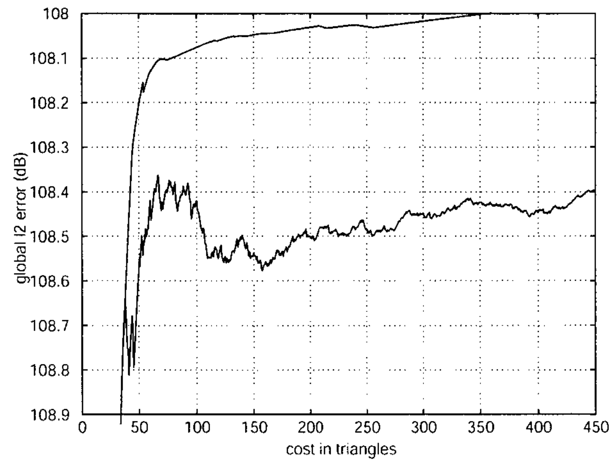


Figure 5.8: Stability of error across rate: We compare the monotoncity of the RD curves obtained using general decimation (top curve) and its restricted counterpart (bottom curve). The curve obtained with the restricted version is unstable at low rates.

5.4.3 Monotonicity

In this section, we show that a rate functional monotonic with the mesh size is necessary and sufficient for the approximation errors to be monotonic across rate. To do so, we assume that no suboptimal choice, as explained in Section 5.4.3, is made. Under the above assumptions, we actually show that the slopes λ, corresponding to optimal choices made during the mesh decomposition, monotonically decrease across rates. This fact is stated in the following proposition:

Proposition 5.3 (Monotonicity of the RD Curve) *Consider that $\Delta R(M_v) \geq 0$ and $\Delta D(M_v)$ is arbitrary, $\forall v$. Call $v^\star$ the optimal vertex to decimate and assume that there is no ancestor $a \in A_{M_{v^\star}}$ such that M_a is a nonmonotonic domain with respect to a vertex $w \in M_{v^\star} \cap M_a$. Then, for all updated vertices $a \in A_{M_{v^\star}}$ we have*

$$\lambda(a) > \lambda(v^\star). \tag{5.25}$$

Proof. We have to show that $\lambda(v^\star)$ is a lowerbound for $\{\lambda(v)\}_{v\in M}$. We only have to consider the updated vertices $a \in A_{M_{v^\star}}$, i.e.

$$\lambda(a) = \frac{\Delta D(M_a) - \Delta D(M_{v^\star})}{\Delta R(M_a) - \Delta R(M_{v^\star})}, \forall a \in A_{M_{v^\star}}. \tag{5.26}$$

Moreover, $M_{v^\star}$ satisfies Proposition 5.2, then

$$\begin{aligned}\lambda(a) &> \frac{\delta\Delta D(M_{v^\star}) - \Delta D(M_{v^\star})}{\delta\Delta R(M_{v^\star}) - \Delta R(M_{v^\star})},\\ &> \frac{(\delta-1)\Delta D(M_{v^\star})}{(\delta-1)\Delta R(M_{v^\star})}\\ &> \frac{\Delta D(M_{v^\star})}{\Delta R(M_{v^\star})} > \lambda(v^\star).\end{aligned} \tag{5.27}$$

□

5.5 Experimental results

We organize our experimental results as follows: In Section 5.5.2, we first demonstrate the efficiency of our global error estimate using the polyline model. We simply recall some of the results obtained in Chapter 2.

Recall that the hierarchy imposed over the vertices by the 4-8 construction restricts the space of approximations (Section 3.2.1). As seen in Chapter 2, this constraint can be applied to the polyline model using a binary tree, i.e. a decimation (or insertion) algorithm must preserve the tree hierarchy when optimizing the model. We refer to these algorithms as *constrained* (Section 5.5.2). We compare our algorithm to two constrained approaches using local error: vertex insertion and vertex decimation.

We also compare our results to the optimal, *unconstrained*, approximations obtained using dynamic programming. In the mesh case, this could be seen as the optimal solutions using irregular triangulations. However, Agarwal *et al.* [1] have demonstrated that finding these solutions is NP-Hard. In Chapter 2, we explain and compare approaches to approximate polylines in detail. Finally, we apply our algorithm to terrain data (Section 5.5.3).

5.5.1 Terrain dataset and measures

In this section, we apply our algorithm to a large set of DETD's. Our dataset represents the southwest region of Switzerland (the Alps) [16] and is composed of 388 terrains of size 257×257 vertices. Consider $e_{\max}$ the maximum error obtained by decimating all the vertices in the mesh and e_r the error measured between an approximation of rate r and M_0. We compute the peak-to-signal noise ratio, or *PSNR* between the two meshes. The error, denoted by E_r, is given in dB and is defined as

$$E_r = 10\log_{10}(\frac{e_{\max}}{e_r+1}). \qquad \text{(dB)} \tag{5.28}$$

5.5.2 Comparison with methods using local error

In this section, we recall some of the results obtained in Chapter 2. We use the polyline model to test the efficiency of our global error estimate for the vertices. As explained previously, this allows us to compare our approach to several approximation methods. In particular, we can compare our approximations to the optimal ones obtained using dynamic programming. In the later case, the space of solutions is not constrained.

Figure 5.9 shows an example of constrained decimation. The top curve is the original one and has 7 interior knots. These knots are iteratively decimated and the bottom curve can be seen as the *control curve*, i.e. similar to the control mesh showed in Figure 3.1a. The binary tree constraining the decimation is depicted using bold lines in the figure.

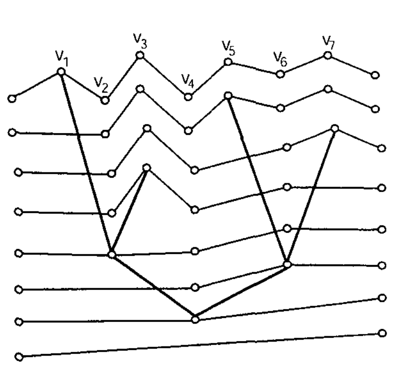


Figure 5.9: Successive approximations of a polyline using constrained knot decimation: Knots are iteratively decimated from top to bottom. The binary tree constraining the decimation is depicted using bold lines.

Our experimental results are shown in Figure 5.10: The graph shows a comparison of the RD curves. The rate is computed as the number of segments forming an approximation and the distortion is computed in l_2 norm with respect to the original curve. We run the experiment using 256 curves obtained from terrain data and we average the results of each algorithm. To compute the average, we normalize the errors and fix the gain to 50 dB.

The top curve shows the errors of the optimal approximations found using dynamic programming. The dashed curve is obtained with our algorithm using general decimation and global error. The two bottom curves are obtained with restricted insertion and decimation using local error. Both approaches accumulate errors through the iterative approximation process. Hence,

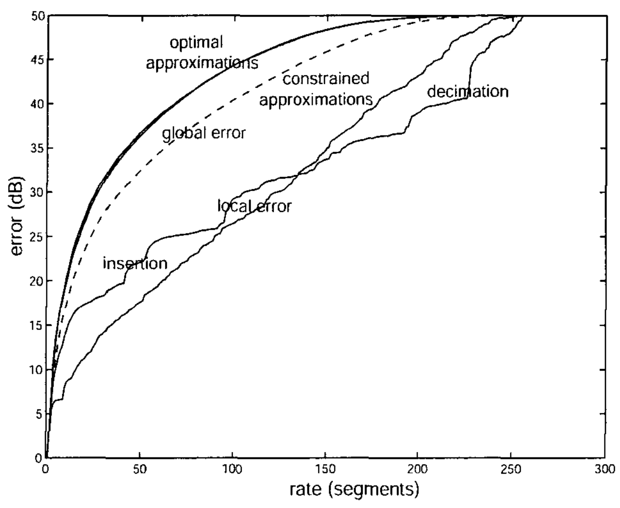


Figure 5.10: Comparison of RD curves: The rate is computed as the number of segments forming an approximation and the distortion is computed in l_2 norm with respect to the original curve. The top curve is obtained using dynamic programming. The dashed curve is obtained with our algorithm. Finally, the two bottom curves compare the decimation and insertion approaches using local error.

the insertion method achieves better quality than the one using decimation at low rates and, symmetrically, the decimation method achieves better quality than the one using insertion at high rates.

5.5.3 Decomposition of terrain data

We apply now our algorithm to terrain data. We run experiments on 388 terrains [16]. Each matrix of amplitudes has size 257 × 257, hence each model has 131'072 triangles. The errors obtained using each terrain is averaged using the maximum error computed as the distance in l_2 norm between the control mesh (Figure 3.1a) and the original mesh. Then, the maximum gain is fixed to 50 dB and the average curve is shown in Figure 5.11. An example of terrain decomposition is shown in Figure 5.13.

We give now some timings when computing a complete decomposition. We use a PC equipped with a Pentium III 500 Mhz and 256 Mb of RAM. The implementation is in C++ and is not optimized. We store the 4-8 meshes with the quadtree described in [55]. For the timings, we use a subset of the available terrains in [16]. We use 50 terrains with sizes up to two millions

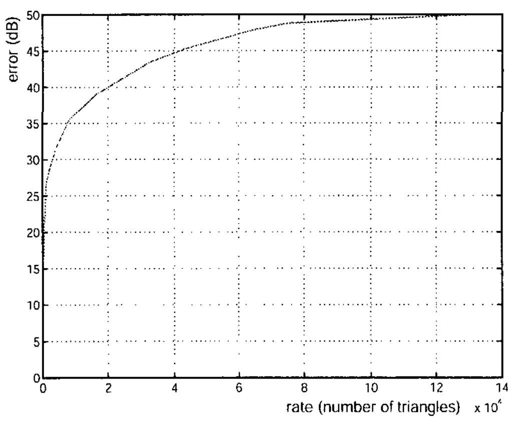


Figure 5.11: Average RD curve: We apply our algorithm to a database of 388 terrains [16]. To compute the average curve, the errors obtained using each terrain are normalized and the maximum gain is fixed to 50 dB.

triangles.

The results in Table 5.5.3 are obtained by measuring the average decomposition time for each terrain. Separate measurements are given for the initialization and the decomposition (lines 1-4 and lines 5-12 of the algorithm in Section 5.2.2, respectively).

number of triangles n	init. time (s)	decomposition time (s)
2048	0.003	0.057
8192	0.011	0.30
32766	0.05	1.7
131072	0.24	11.4
524288	1.07	48
2097152	4.81	238

Table 5.1: Average decomposition times: The timings are obtained using a PC equipped with a Pentium III 500 Mhz and 256 Mb of RAM.

5.6 Summary

We presented a scalable mesh approximation framework for 4-8 meshes. Our method allows us to perform approximations using global error at low computational cost $\Theta(n \log n)$. Our solution has several advantages:

- ▶ The variable-rate optimization approach allows us to best conserve the monotonicity of the distortion across rates.
- ▶ Minimal requirements for the specification of the rate and distortion functionals provide high flexibility for the user to target their definition according to the application.
- ▶ The functionals are only evaluated during the initialization stage, allowing us to keep their utilization apart from the optimization process using an efficient update mechanism.
- ▶ Our update algorithm allows us to compute a global error estimate for the mesh improving the quality of the approximations in the constrained setting, moreover the update is carried on at minimal computational cost.
- ▶ Our framework is computationally efficient and can handle large datasets.

We used a metric where the rate functional counts the number of triangles and the distortion functional measures the l_2 norm in world-space. However, recall that the flexibility of our framework allows us to tailor the metric to the application with great freedom (recall the minimal assumptions on the functionals presented in Section 5.4.1). This characteristic allows us to address several problems inherent to our transmission framework (Section 1.2.6). In Chapter 6, we will use our algorithm to perform optimization in screen-space, i.e. to obtain view-dependent approximations.

Appendix 5.A Evaluation of mesh functionals

We compute the costs in rate for each domain, measured as the number of triangles, in closed form using the results in Chapter 3. More precisely, we give closed-forms to compute $R(M_v)$ and $R(\check{M}_v)$.

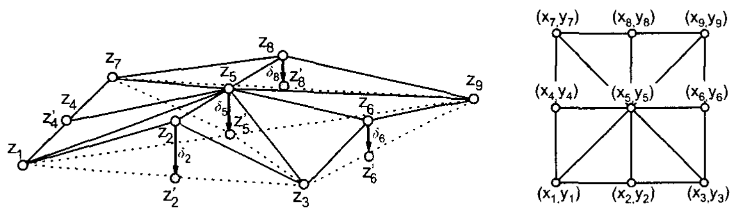


Figure 5.12: Computation of the error in squared l_2 norm for a triangulated quad. The right part shows a top view of the triangulated quad.

We use the squared l_2 norm as a measure of distortion between the original mesh and the approximations. More precisely, each vertex is projected in its corresponding triangles in the support of the domain. Consider the simple case of a matrix of amplitudes z. Figure 5.12 shows how errors are measured on a triangulated quadrilateral. The total distortion is evaluated similarily on the domain M_v.

Consider the triangulated quadrilateral in the left-hand side of Figure 5.12. Each amplitude z_i is projected in a triangle of the support. For example, z_2 is projected in the triangle formed by vertices $(x_1, y_1, z_1), (x_5, y_5, z_5), (x_3, y_3, z_3)$ (right-hand side of Figure 5.12). Denote by z'_i a projected amplitude, hence the error for each vertex is then given by

$$\delta_i = |z_i - z'_i|^2. \tag{5.29}$$

To evaluate the error $D(\check{M}_v)$, all the vertices in the domain are projected in the support of M_v. The distortion functional is computed as

$$D(\check{M}_v) = \sum_{v \in M_v} \delta_v, \quad D(M_v) = 0. \tag{5.30}$$

hence

$$\Delta D(M_v) = -D(\check{M}_v), \tag{5.31}$$

i.e. intially $\Delta D(M_v) < 0$ for all the vertices.

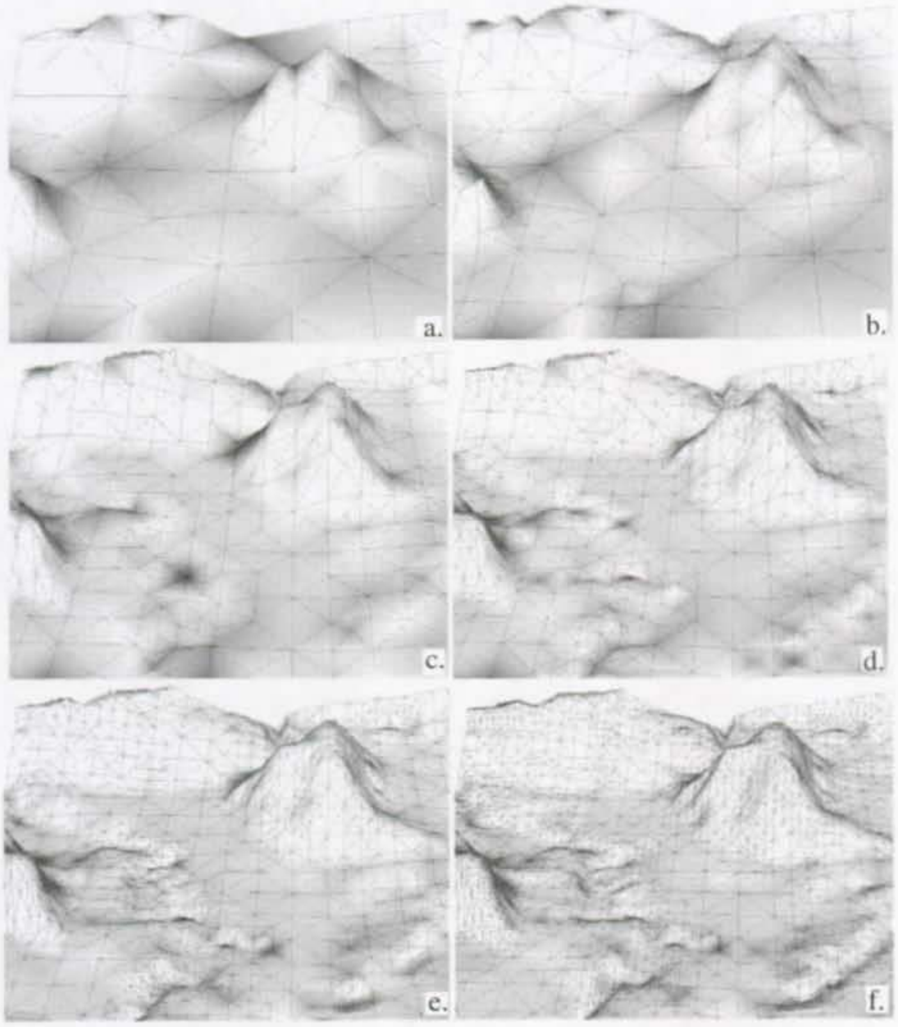


Figure 5.13: Progressive refinement of a terrain mesh: (a) A coarse resolution of the mesh is generated with: 200 triangles (PSNR 14.9 dB), successively refined to (b) 400 (PSNR 18.46 dB), (c) 800 (PSNR 22.2 dB), (d) 1600 (PSNR 25.914 dB), (e) 3200 (PSNR 29.81 dB) and finally (f) 6400 triangles (PSNR 33.8 dB).

Chapter 6

Joint mesh and texture optimization

6.1 Introduction

6.1.1 Motivation

In this chapter[1], we address a very simple yet basic question of computer graphics: Given a *surface specified by a mesh and a texture to be mapped on that mesh, what is the resolution* required for the mesh and the texture to achieve the best display quality, respectively? This question is relevant every time we are resource constrained, and we need to simplify meshes and/or textures mapped on the meshes. The constraints might be of different nature:

- Computational: For example, complexity of rendering meshes and mapping textures.
- Storage space or bandwidth: For example, bitrate needed to represent, transmit or update a complex model and its associated texture information.
- Both in computation and bandwidth.

In such cases, it is critical to choose the correct resolution or level of details for both the mesh and the texture mapped on it. If more resources become available (e.g. in progressive downloads over the Internet), it is important to know if the mesh or the texture needs to be refined first.

To make our discussion more concrete, consider a specific case where the interplay of texture and mesh resolution is particularly intuitive, for example, very large terrain models with aerial photographs to be mapped on the terrain models. Consider two extreme cases:

[1]*This work was done in collaboration with Stefan Horbelt (Stefan.Horbelt@epfl.ch)*

First, assume mapping a very fine texture (a very detailed aerial photograph) onto a very coarse terrain model. This gives an unsatisfactory result: While many details are available, the surface model is too simplistic compared to the texture resolution. At the other extreme, take a very detailed terrain model, but with an overly coarse texture. The mesh is now very complex, but the texture is trivial, thus not reaching a good trade-off. The above two extreme cases hint at a better solution, where the correct trade-off between mesh and texture resolution is found.

This investigation targets the characterization of control information for the *system control module* of our transmission framework (Figure 1.4). Recall that this module operates as a puppeteer for the encoder set and the multiplexer. Our joint encoding investigation provides a means to share resources between geometry and texture data inside the multiplexed bitstream such that the encoded information minimizes the distortion criterion for a given bit budget.

To the author's knowledge, this joint optimization problem has not been previously addressed. A "single-sided" point of view would twig our ambitions as optimizing geometry with texture information (see [74] for example), whereas in our problem there truly is a concurrent approach to achieve the best rendering quality. To solve this problem, we propose an efficient algorithm based on *marginal analysis* [6, 7]. Marginal analysis is a greedy optimization method used in economy to solve convex and linear programming problems [46]. More recently, this technique has been successfully exploited to address problems in image processing [92]. The aim of this chapter is to show that there exists a non-trivial interaction between geometry and texture contribution to the overall quality of the rendered image. We address a simplified version of the problem by evaluating the interaction between progressive view-dependent encoded textures [45] and meshes. In contrast, the general problem would consist in generating jointly both progressive descriptions.

6.1.2 Contributions and plan

The following points summarize the main contributions of our work:

Format framework for joint mesh-texture optimization We explain how the joint optimization problem can be addressed efficiently using a pyramid of view-dependent simplified meshes and view-dependent encoded textures.

Efficient heuristic We introduce a greedy algorithm to compute the set of pairs (M_i, T_j) based on marginal analysis. The algorithm runs in linear time with the number of optimal pairs.

Near optimal algorithm We compare the pairs (M_i, T_j) found by our efficient heuristic to the optimal ones and show that our algorithm performs close to the optimal path found by exhaustive search.

Balancing of mesh and texture data Our work ultimately contributes with a technique to balance the quantity of mesh and texture information in a bandwidth constrained environment in order to maximize the rendered quality.

This chapter is organized as follows: We start with a review of classical texturing approaches and give then our contributions in Section 6.1.2. We pose formally our optimization problem in Section 6.2, then explain how marginal analysis can be used to address this problem in Section 6.2.2. In Section 6.3, we study the complexity of the problem and give an algorithm. We give extensive experimental results in Section 6.4. Finally, Section 6.5 summarizes our results.

6.1.3 Classical approaches to texture processing and encoding

In this section, we first briefly review the main techniques used to encode and process texture information and discuss their interest within a transmission framework. For a more detailed survey, we refer the reader to [39].

Raw data and mip-mapping Most of the time in computer graphics applications, textures don't have a specific encoding other than the description provided by their original image format. Once loaded into memory, the data is converted to a format supported by the rendering engine [64]. In order to deal with seamless replication of the texture on the mesh, duplicates at multiple resolution are passed to the engine. This technique, known as *mip-mapping*, is

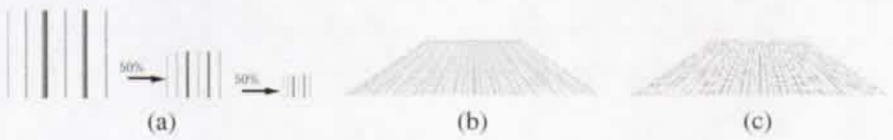


Figure 6.1: Mip-mapping example: (a) The texture is stored at multiple resolutions, passing at set of *pre-filtered* textures to the engine. (b) Using mip-mapping, low-resolution duplicates are replicated on polygons distant from the viewer. (c) The uniform replication of a high-resolution texture produces *aliasing*.

commonly supported in hardware and allows us to avoid aliasing when patches are replicated on polygons distant from the viewer[2].

Figure 6.1a depicts the original texture and its pre-filtered duplicates. In Figure 6.1b, we show the effect of tiling the texture using mip-mapping, whereas in Figure 6.1c, we illustrate the aliasing effect when only the highest resolution component is tiled on the polygons.

[2]Using a perspective projection, far polygons will appear smaller than the ones close to the viewer.

Subband coding The use of subband coding methods such as the wavelet transform [88] can be used to obtain a progressive description of the texture. This encoding scheme is well adapted to transmission since few texture data, represented as transform coefficients, can be transmitted in order to obtain an approximation. The wavelet transform describes the texture as a set of embedded coefficients. Therefore, the refinement of a previously transmitted set can be achieved by sending only further *detail coefficients*. Finally, the coefficients reflect local properties of the texture, therefore the refinement can be restricted to regions allowing us to perform *view-dependent optimization* [45], as depicted in Figure 6.5. Using this property, the local resolution of the texture can be adapted in order to avoid aliasing such as in the case of mip-mapping (Figure 6.1). For more informations, we refer the reader to [88, 60, 90].

Texture generation Texture generation techniques are quite new to the field of image processing and aim at producing artificial texture patterns with statistical values measured from an input image [65, 79]. Although this technique can only describe simple patterns, its strength relies on its ability to describe patches using small sets of parameters.

Procedural textures Procedural techniques can be seen as a more general case of texture generation. These methods aim at simulating natural materials and phenomena using algorithmic descriptions [91, 73, 63] and rendering them into a texture. The functions generating the texture are then evaluated when needed. For example, in Figures 6.2a-c, procedural textures are used to simulate fog [21, 22], fire [28] and clouds [27], respectively. The examples in the figure were obtained from the advanced SIGGRAPH openGL course. Procedural methods are usually computationally demanding since the functions must be constantly evaluated in order to update the texture. A good survey of procedural texturing is given in [23].

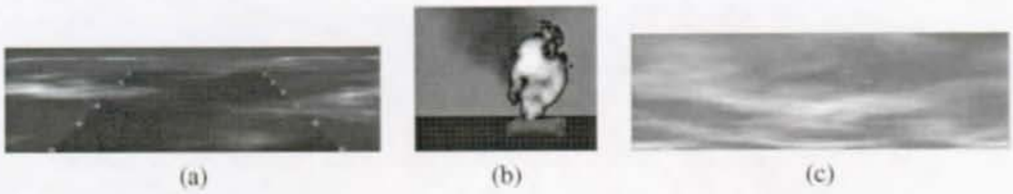


Figure 6.2: Texture generation using procedural approaches: Simulation of (a) fog, (b) fire and (c) clouds. The functions encapsulating the texture synthesis are evaluated periodically, hence producing animated textures. These examples were taken from the advanced openGL SIGGRAPH course 1997.

How fit the above encoding methods in a transmission framework? Sending raw data, doubtlessly, is too limiting since it may waste bandwidth resources. As said previously, our encoding scheme uses subband coding methods to achieve progressiveness for the bitstream. The last two techniques, namely texture generation and procedural textures, based on synthesis are interesting for their compactness, but complicate the design of the *texture decoder module* (Figure 1.4), i.e. the decoder should handle multiple decoding schemes. We will therefore consider them no further, although we keep in mind that they offer exciting alternatives to classical transform coding. Note that a procedural method would also require us to define an interaction protocol between the texture decoder and the rendering engine since procedural textures are periodically synthesized.

6.2 Problem formulation

6.2.1 Joint mesh-texture optimization

Consider a surface mesh at full resolution M_0 and a family of simplified meshes $\{M_i\}_{i=0\ldots N-1}$. A progressive set is obtained with the algorithm presented in Chapter 5. We associate a value $C_M(M_j)$ to each mesh representing its cost (number of bits needed to represent it, or its computational rendering cost).

Likewise, given a full resolution texture T_0 and a family of simplified textures $\{T_i\}_{i=1\ldots M}$, obtained with the method in [45]. We associate a function $C_T(T_j)$ to each texture, which measures its cost (again, bitrate or computational cost of rendering). In Figure 6.3, we depicted our full resolution model using the mesh M_0 and the texture T_0. The mesh is built on a 257×257 DETD (see Section 5.1.1) and the texture is a 1024×1024 pixels, 24 bits image.

Finally, for a given pair (M_i, T_j), we define a distortion measure $D(M_i, T_j)$ which evaluates the error in screen space between the image generated by the full resolution version (M_0, T_0) and the approximated version (M_i, T_j). Calling the image on the screen $I(M_i, T_j)$ to reflect its dependence on the underlying model and texture, a possible distortion is the l_2 norm, or

$$D(M_i, T_j) = \|I(M_0, T_0) - I(M_i, T_j)\|_2 \,. \tag{6.1}$$

Note that $I(M_i, T_j)$ depends on the rendering process (e.g. lighting), but we assume the same rendering process for the various approximate images. The statement of the problem is now the following: Given a total budget for the combined complexity of the mesh and texture:

$$C = C_M + C_T, \tag{6.2}$$

find the pair (M_i, T_j) that minimizes $D(M_i, T_j), i \in [0 \ldots N], j \in [0 \ldots M]$, i.e.

$$D_{min} = \min_{i,j}(D(M_i, T_j)), \tag{6.3}$$

under the constraint that

$$C_M(M_i) + C_T(T_j) \leq C. \tag{6.4}$$

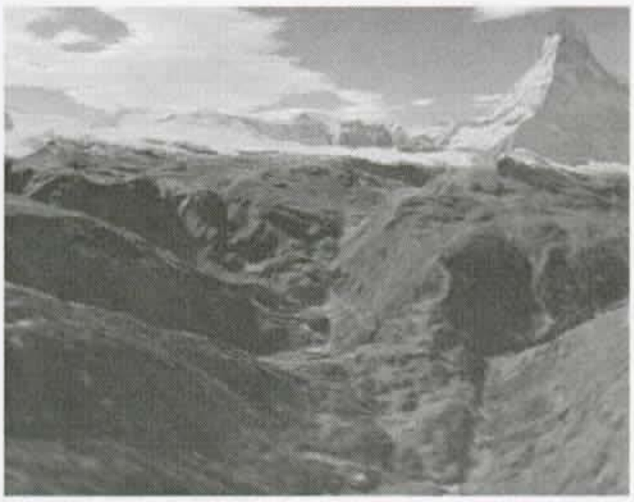


Figure 6.3: Full resolution model: The model is rendered using the mesh M_0 and texture T_0.

Such a formulation is very reminiscent of compression systems, and indeed, when the cost is the required bitrate for model and texture, the above gives a best compression scheme in an operational rate-distortion sense.

6.2.2 Marginal analysis

The search for the minimum error in (6.3) can be done by exhaustive search (i.e. comparing all possible pairs of mesh and texture resolutions satisfying the bitrate constraint), but this is clearly impractical (see Section 6.3.1). Thus it is critical to find computationally efficient methods for searching a wide variety of possible approximations. In the following, we focus our attention on the case where the cost is thé bitrate required for representing the mesh and the texture. This is a well defined cost and it is relevant for applications where storage and/or transmission are involved.

Finally, We restrict our attention progressive meshes and textures. While this is a restricted class, it is a very important one for the case where interactivity and communication is involved (e.g. progressive downloads).

Dataset, cost and distortion operators We use a set of N progressive, view-dependent approximated meshes obtained with digital elevation terrain data (DETD) [16]. In Figure 6.4a, an example of view-dependent approximation used for the experiments is shown. Figure 6.4b depict the view-dependent simplification from a location slightly behind the one using for the simplification. Finally, a world-space simplification of the model is shown in Figure 6.4c for comparison.

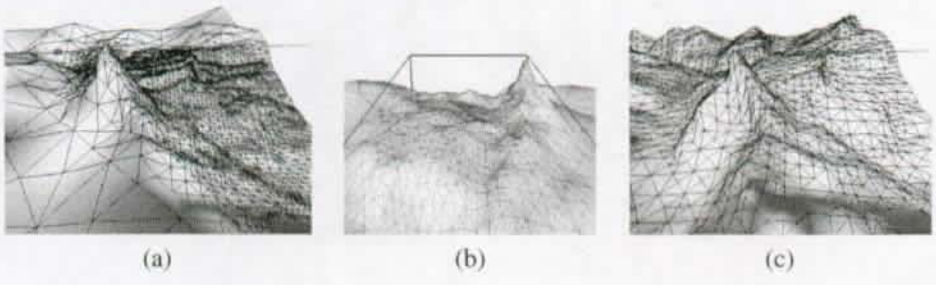

(a) (b) (c)

Figure 6.4: View-dependent and world-space optimizations: In (a), the model has been optimized for a specific viewpoint (see the concentration of triangles) In (b), the terrain is viewed from a location slightly behing the viewpoint used for simplification. In (c), the model using a metric in world-space.

The simplification is We use well-known results in geometry compression [85] to obtain an estimate for the cost of encoding both vertices and connectivity. The set of M progressively refined textures is generated with the aerial photograph in Figure 6.5.

The cost of a texture accounts for the number of 16-bits quantized wavelet coefficients used for the reconstruction. Since we have view-dependent representation of both mesh and texture, the error is computed in screen-space. To do so, we compute the distance between the two rendered images in the screen, assuming the same rendering conditions for the various approximated images.

Greedy optimization using marginal analysis We propose an optimization scheme based on marginal analysis [6, 7] to solve the problem. Marginal analysis is primarily a technique used in Economics [46], but recently has been successfully applied in image processing [92].

Intuitively, marginal analysis can be explained as follows: Economists say that when making a decision, people think at the margin, which simply means that they compare marginal costs with marginal benefits. A rational person will then pursue an activity until the marginal benefits equal the marginal costs. Basically, marginal analysis is based on the idea that it is the future costs and benefits that everyone bases decisions on. For example, say you are deciding whether to read this thesis further. You will evaluate the marginal benefits against the marginal costs (all the things you should give up by spending more time reading). If the marginal benefits are greater than or equal to the marginal costs, then you continue the reading. In fact, you continue

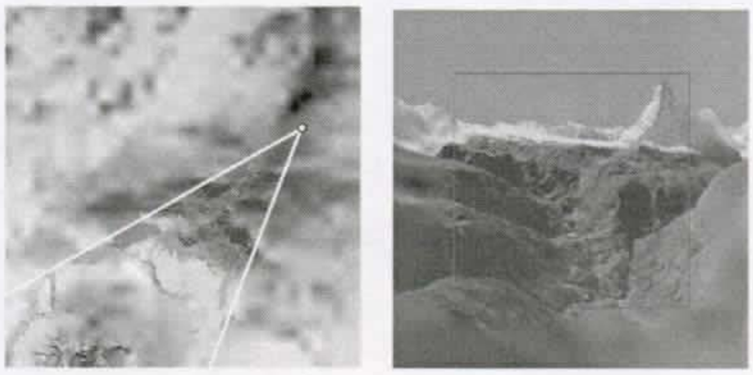


Figure 6.5: View-dependent encoding of a texture obtained from an aerial photograph of the Alps (courtesy of Stefan Horbelt): (a) The texture is encoded such that it can be locally refined in the visible regions in the field of view. The point represents the user viewpoint and the rays delimit the projected pyramid of view, (b) the texture is projected on the terrain and the scene is shown from a viewpoint located slightly behind the real viewpoint. We can see that regions outside the field of view are blurred since less information is encoded for these parts.

until the benefits are equal to the costs. Going any further would cause costs to become greater than benefits, and would be irrational.

When applied to our problem, marginal analysis translates as follows: A necessary condition for a pair (M_i, T_j) to be a candidate that achieves a good trade-off between mesh and texture resolution is that removing a certain bit budget from either mesh or texture will lead to a similar increase in distortion. The intuition behind the result is simple: If it were not so, some of the bits could be moved from mesh to texture (or vice versa) in order to reduce the distortion. While this result is exact only when assuming independence of the mesh and texture (which is an approximation), the heuristic using it is quite competitive.

Two applications are possible: We can either *increase* or *reduce* the available resources, i.e. the bit budget in our case. Resource reduction approaches are usually more competitive than resource increase methods. This fact is well know in optimization [31] and has been illustrated by the greedy approximation algorithms in Section 2.4.1 where we used refinement and decimation. In the next sections, we first explain how our problem can be solved using exhaustive search in order to illustrate the complexity of the solution space. Then, we give the algorithm based on marginal analysis using the *resources reduction* approach to solve it.

6.3 Algorithm description

6.3.1 Space of solutions and complexity

We first conduct an exhaustive search experiment for a set of pairs (M_i, T_i) of meshes and textures approximated for a specific viewpoint. The viewer location, corresponding to the one in Figure 6.4b, is such that only 10% of the terrain mesh is visible. To generate the approximation M_i and T_j, we increase exponentially the rate of the mesh and the texture from 0.01% to 10% of the available vertices and wavelet coefficients. In total, 32 textures and 60 meshes are generated giving a total of 1920 pairs.

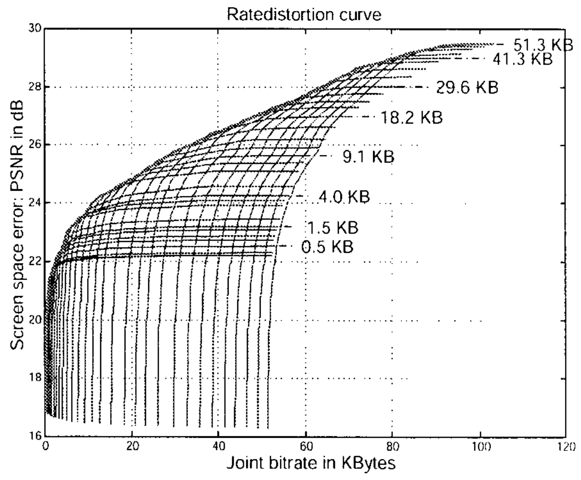


Figure 6.6: Exhaustive search in the space of solutions: The thick curve shows the optimal rate-distortion pairs upper bounding the convex hull of all configurations found by exhaustive search. Each thin curve represents a constant texture rate, where the rate value is indicated for some of them.

Figure 6.6 shows the results of the experiment: For each pair $(M_{i=1...59}, T_{j=1...31})$, the PSNR with respect to the full resolution pair (M_0, T_0) is evaluated in the screen space. Each thin curve in the figure corresponds to a fixed texture rate. The rate of the mesh increases on the x-axis, while the y-axis shows the resulting PSNR. Note that only some of the texture rates are indicated in the figure. The optimal path (thick curve in the figure) upperbounds the convex hull of all curves. Following this path gives the highest PSNR for a given bitrate.

6.3.2 Efficient greedy strategy

In this section, we explain the algorithm that performs a marginal analysis to search the solutions to (6.3). The algorithm searches greedily for the best tradeoff while reducing the resolution of both mesh and texture. In other words, at each step the resource -the bit budget in our case- is dwindled. In this *resources reduction mode*, a full resolution with full resolution texture is successively coarsened. At each step, given a target budget reduction of B bits, the best of lessening the mesh or the texture description by B bits is chosen (or possibly, reducing both by $B/2$).

In practice, the next coarser version of the corresponding entity fixes the bit reduction B. The downgrading is evaluated by comparing the PSNR of the degraded model (i.e. textured mesh) with respect to the original in screen space. Albeit we apply a greedy approach, we will show in Section 6.4 that it performs quite close to an optimal selection in our experiments. The computational complexity of the greedy algorithm is very attractive since its magnitude is linear with the optimal path in the space of configurations.

Although marginal analysis is a greedy strategy, optimality can be obtained when the two entities are independent. In this particular setting, the problem turns out to be standard Lagrangian optimization [51]. As explained previously, an alternate way to operate the algorithm is to increase the resolution of both texture and mesh. However, increasing resources is doubly greedy and would not even perform optimally in the case of independence [31], thus the reduction approach is preferable. We give now the algorithm:

ALGORITHM

```
▷ Input: PAIRS $(M_i, T_j)$ WITH $\mathbf{i} = \{0, \ldots, N\}$ AND $\mathbf{j} = \{0, \ldots, M\}$
◁ Output: A SET OF PROGRESSIVE PAIRS $(M_l, T_m)$, WITH $l \in \mathbf{i}$ AND $m \in \mathbf{j}$
1 $i = 0, j = 0$
2 while $i \leq N$ or $i \leq M$ do
    3 $D_{\min} = \arg\min[D(M_{i+1}, T_j), D(M_i, T_{j+1})]$
    4 if $D_{\min}$ is $D(M_{i+1}, T_j)$
        5 STORE THE PAIR $(M_{i+1}, T_j)$
        6 $i = i + 1$
    7 else
```

```
            8 STORE THE PAIR $(M_i, T_{j+1})$
            9 $j = j + 1$
        10 end
    11 end
```

6.4 Experimental results

In this section we give the experimental results obtained for our DETD and aerial photograph. We compare the greedy path to the optimal path found by exhaustive search (Section 6.3.1). Figures 6.7 and 6.8 both show the optimal RD path found by exhaustive search (thick line) and the path found by marginal analysis (thin line). In both figures, the number of triangles increases along the x axis, whereas the amount of texture coefficients increases along the y axis. The figures clearly illustrate that our greedy method gives a good approximation of the optimal path.

Constant PSNR curves Additionally, Figure 6.7 illustrates (for all combinations of the 60 meshes and 32 textures) the iso-lines of constant PSNR. The PSNR value is given by the line intensity, which refers to the intensity bar on the right hand-side of the figure. Low PSNR curves are on the left hand side, whereas high PSNR curves are on the right hand side. This figure reveals an interesting behavior of the error criterion (recall that we are using as distance the l_2 norm in the screen): Consider the white points in the figure locating a series of model of constant PSNR. They show that, when using a coarse resolution mesh (in this example 0.2% of the vertices are used), increasing the resolution of the texture surprisingly does not reduce the distortion, although one might infer the appearance to be more appealing (compare for example Figure 6.9a and 6.9f). This illustrates the well-known problem that a perceptual measure would be a better adapted error criterion than the l_2 norm.

Constant joint rate curves In Figure 6.8, we represent the iso-lines corresponding to configurations of constant joint rate. Again, the magnitudes can be read using the intensity bar on the right hand-side. Low rate curves are on the bottom left corner, whereas high rate curves are on the top right corner of the figure. The white points locate a series of models of constant joint rate, represented in Figures 6.10a-f. Practically, this set of figures shows 6 different bit repartitions leading to the same rate. This example points out that computing the optimal joint rate for the model is definitively important since we can see in the figures that the perceptual quality greatly varies according to the resource attribution.

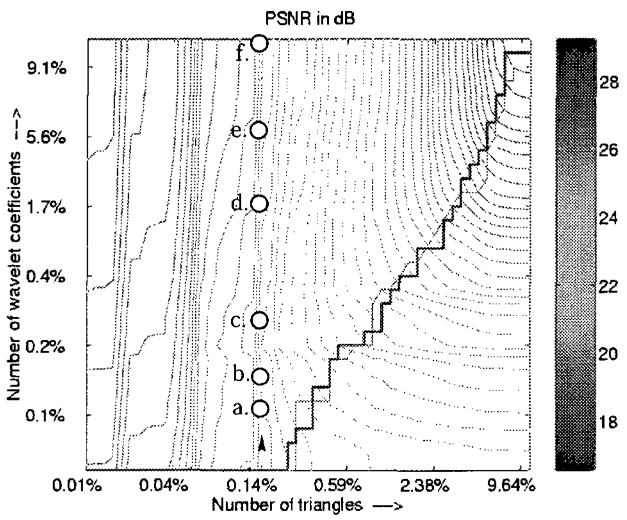


Figure 6.7: Iso-lines of constant PSNR for all mesh and texture combinations, whose magnitude can be found using the intensity bar on the right hand-side. Low PSNR curves are on the left hand side, whereas high PSNR curves are on the right hand side. The thick lines show the optimal RD path found by exhaustive search and the thin lines form the path found by marginal analysis. The white points locate the models shown in Figures 6.9a-f.

Greedy approach complexity The path found by the marginal analysis has a length of 84 frames. To find it, twice as much frames had to be rendered and their screen space error evaluated (recall that the computational complexity is linear with the path length). Both Figure 6.7 and 6.8 show that the path is close to the optimal path found by exhaustive search. Exhaustive search requires to evaluate about 1920 mesh-texture combinations, whereas marginal analysis needs only 168 evaluations. We can conclude that it provides an efficient tool to select pairs (M_i, T_i) that are close to optimal. In Appendix 6.A, we show a series of models on the near optimal path by found marginal analysis.

6.5 Summary

In this chapter, we presented a method allowing us to determinate the balance between mesh and texture data in order to maximize the rendered quality when resources are constrained. In particular:

▶ We clearly showed that there is a important interaction between mesh and texture in ren-

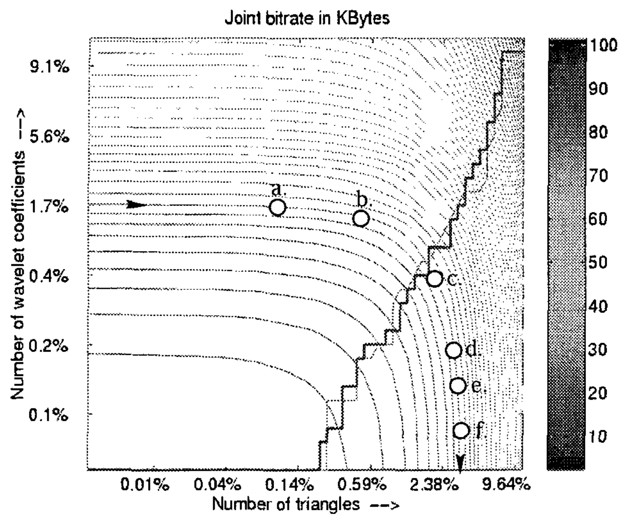


Figure 6.8: Iso-lines of constant joint rate of all mesh and texture combinations, whose magnitude can be found using the intensity bar on the right hand-side. Low rate curves are on the bottom left corner, whereas high rate curves are on the top right corner of the figure. The thick lines show the optimal RD path found by exhaustive search and the thin lines form the path found by marginal analysis. The white points locate the models shown in Figure 6.10a-f. For these models, the constant joint rate is approximatively 10Kb.

dered images (Figures 6.10a-f).

- We used a computationally efficient greedy heuristic to choose the mesh and the texture resolutions satisfying the constraints.
- We showed that we can determine near-optimal mesh and texture pairs with our method.

We also gave an example of a well-known fact in visual sciences: the need for a perceptual measure (Figure 6.9a-f). In Chapter 7, we will explained how our algorithm immediately inserts in the system control module of our transmission framework (Figure 1.4).

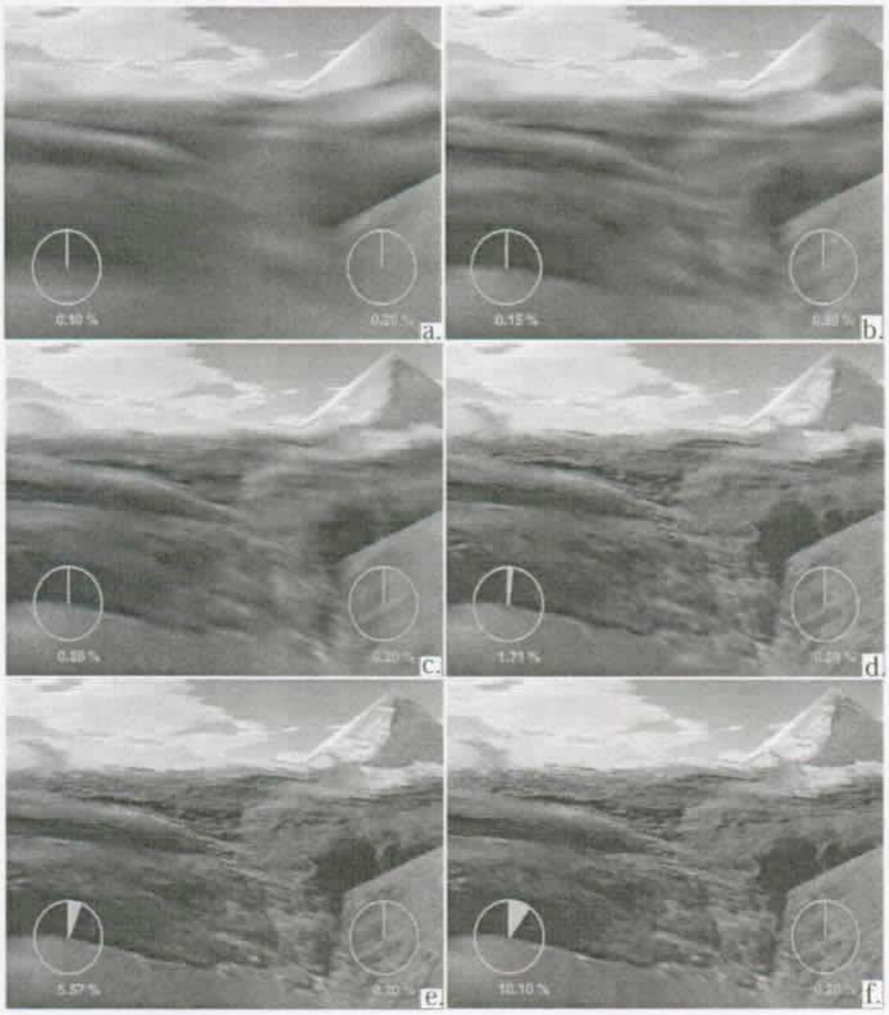


Figure 6.9: Models along a constant PSNR curve: The models in (a) to (f) correspond to the points (bottom to top) in Figure 6.7.

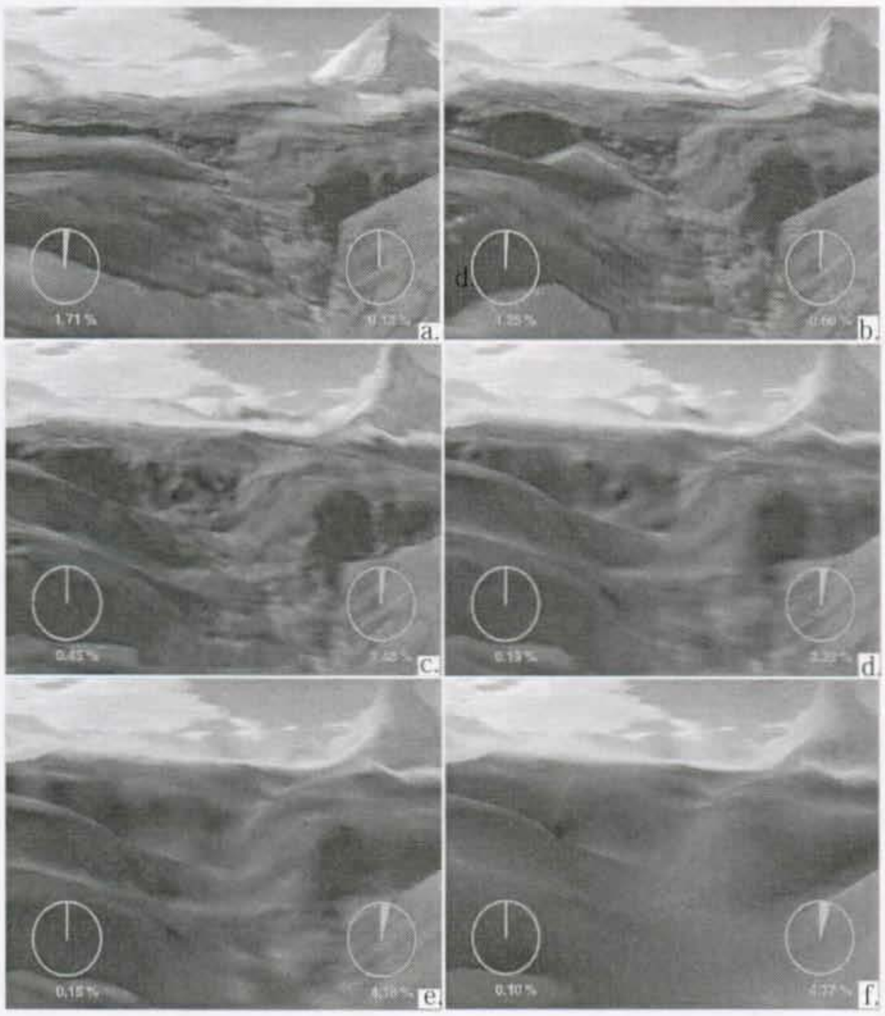
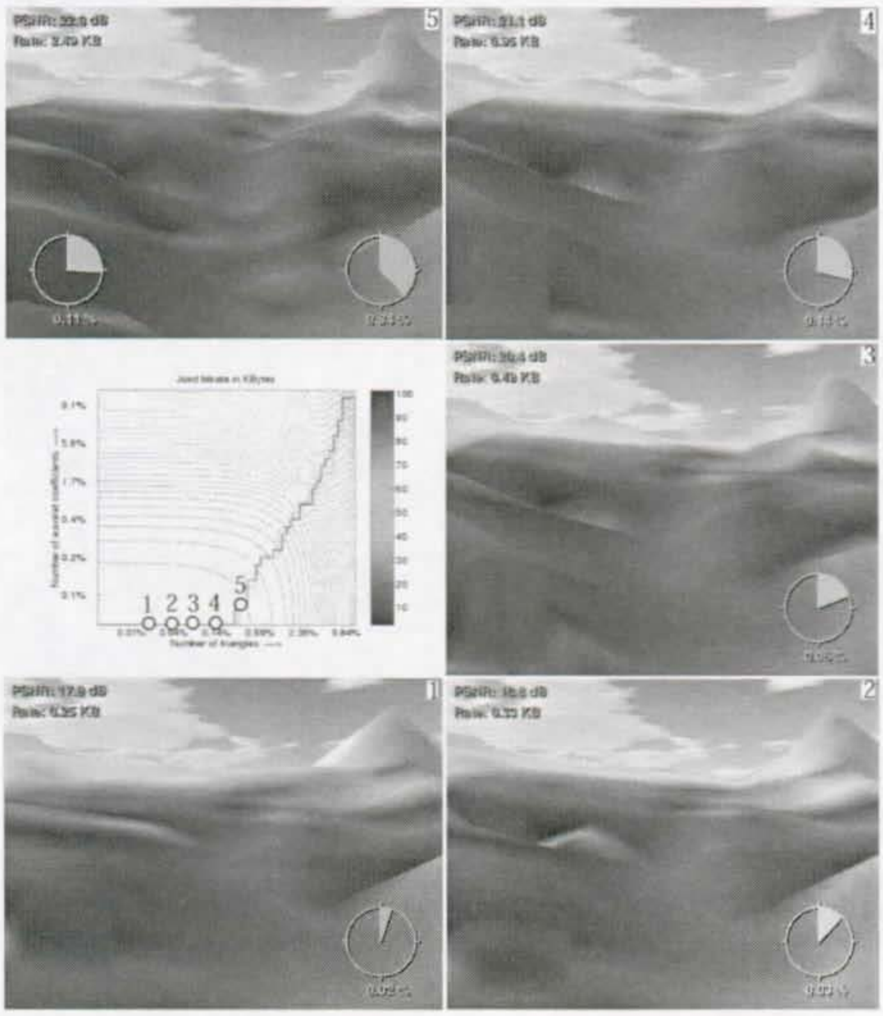


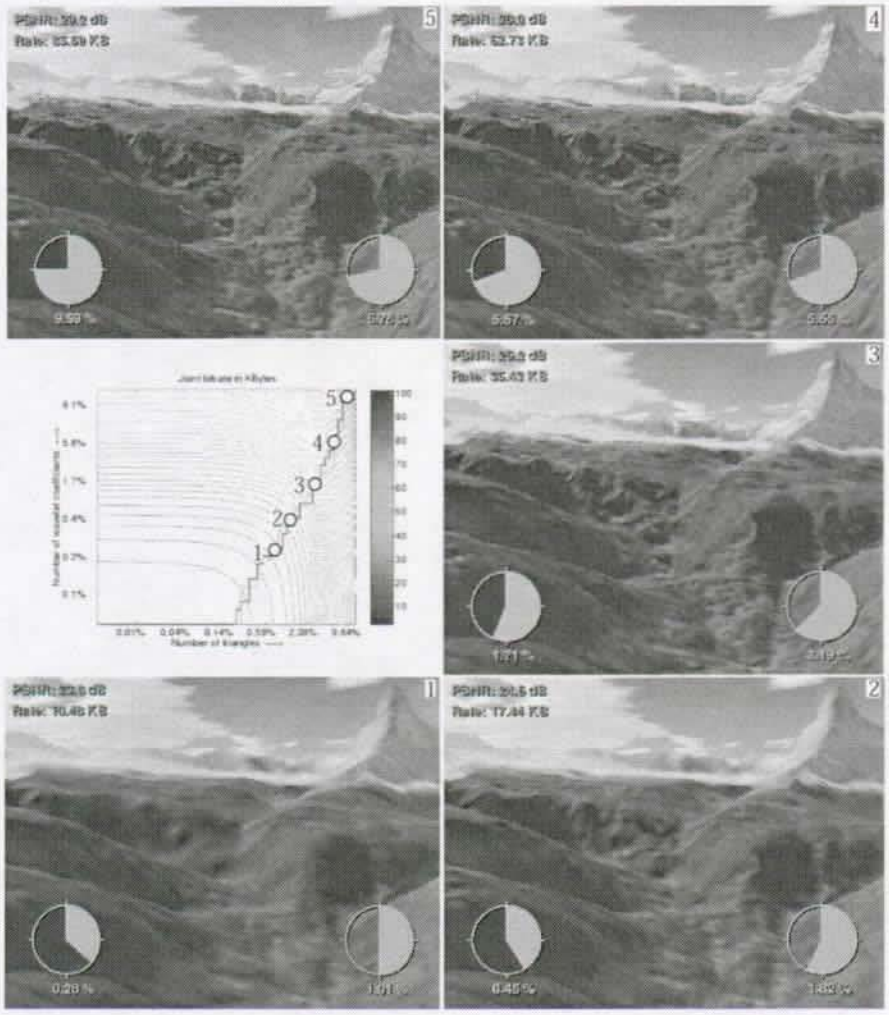
Figure 6.10: Models along a constant joint rate curve (approximatively 10 Kb).

Appendix 6.A Models on the near optimal path

Low joint rate models on the near optimal path The points in the graph, showing the near optimal path, correspond to the model indices. The pie graphs show the repartition between texture (left hand side) and mesh (right hand side). The PSNR's and joint rates are displayed in the figures. Note that in Plates 1 to 4, the texture rate is close to 0 and no pie graph is displayed.



High joint rate models on the near optimal path The points in the graph showing the near optimal path correspond to the model indices. The pie graphs show the repartition between texture (left hand side) and mesh (right hand side). The PSNR's and joint rates are displayed in the figures.



Chapter 7

Conclusion

7.1 Introduction

We close this thesis in two steps: First, we explain how the results obtained in this thesis are placed within the transmission framework presented in Chapter 1. We describe their placement in the transmitter only (Figures 7.1a-b and 7.2a-b), since their existence in the receiver is mainly symmetric. Then, we present several improvements for our framework and delineate future research directions.

7.2 Transmitter characterization and module interplays

7.2.1 Review of approximation algorithms

The technical review of approximation algorithms in Chapter 2 allowed us to understand the advantages and drawbacks of the main approaches to our approximation problem. We chose to generalize to the multi-dimensional case the only algorithm gathering all the requirements of our transmission framework, namely the optimal tree-constrained approach (Section 2.6). We recall below the reasons for our choice.

This algorithm has the following interesting properties: First, it is computationally efficient, which allows for the processing of large datasets. Second, when dealing with surfaces, the constraints imposed over the dataset allows for connecting the vertices using a recursive procedure. A direct consequence is that few bits are necessary to describe the mesh connectivity, yielding compact storage. Another important outcome is that the successive approximating triangulations are embedded, which results in a progressive decomposition of the mesh (Figure 2.18). Third, we showed that a global error estimate can be advantageously used for the vertices if the hierarchy imposed over the dataset is preserved through the simplification process. As a result, we obtain high quality approximations. Finally, the monotonicity of the approximations is conserved through the decomposition (Section 2.2.3). This feature ensures that, when reconstructing the model, each refinement reduces the distortion.

7.2.2 Efficient, scalable geometry encoding

The analysis of basic optimization operations for 4-8 meshes in Chapter 3 proved very useful to evaluate the computational requirements to process this class of meshes. These results led to efficient algorithms for the geometry encoder (Figure 7.1a).

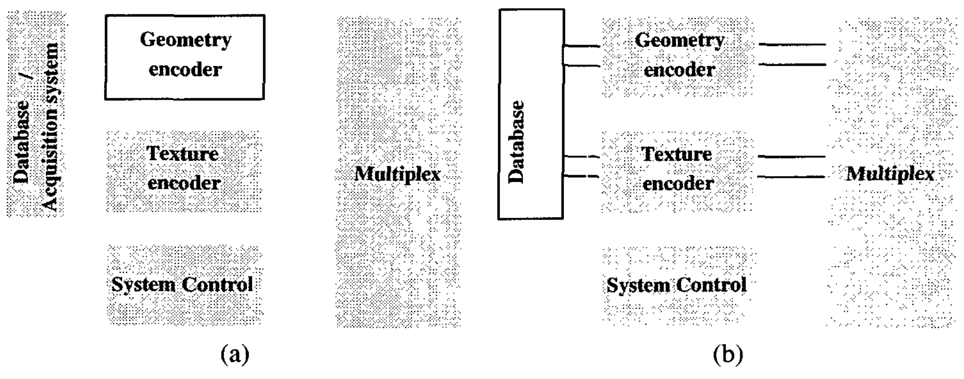


Figure 7.1: Placement of the results in our transmission framework: (a) The investigation of Chapter 3 provided a comprehensive study of the mechanisms implementing the geometry encoder module. (b) The results of Chapter 4 allowed us to implement efficient data accesses between processing modules and the database.

In Chapter 4, we studied a quadtree data structure providing compact storage and efficient access to large datasets. Therefore, this investigation mainly contributed to implement efficient operations between the processing modules and the database (Figure 7.1b).

7.2.3 Model approximation under constraints

Our preliminary investigations in Chapters 3 and 4 permitted us to construct an efficient algorithm in Chapter 5 to compute approximations in an operational rate-distortion (RD) framework. In particular, using the results of Chapter 3, we were able to use a global error estimate to simplify 4-8 meshes. With the results of Chapter 4, we obtained computational optimal implementations our algorithm leading to a computationally efficient mesh approximation framework.

An important aspect of our method is its ability to perform optimizations under constraints. This feature allowed us to address the joint optimizations problems of our transmission framework described in Section 1.2.6 (Figure 7.2a). We illustrated the case of joint source-rendering approximations, i.e. satisfying the constraints of the rendering hardware. We explain how to address the general problem of joint source-channel-rendering in Section 7.3 and leave it as a future research direction.

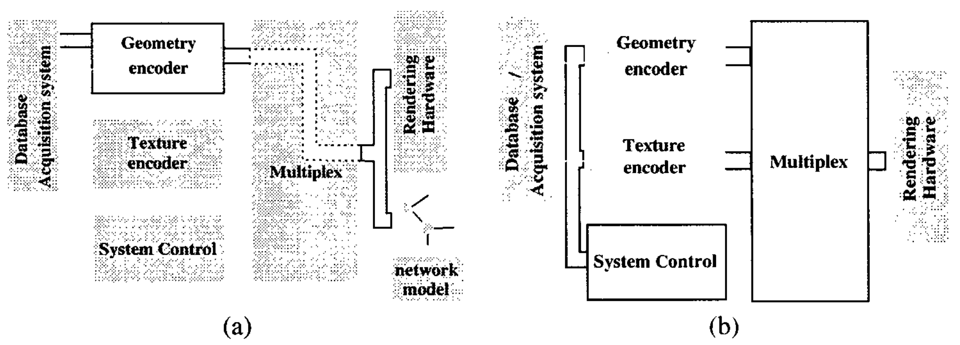


Figure 7.2: Placement of the results in our transmission framework: (a) Our algorithm allows us to perform optimization under constraints. More precisely, meshes can be approximated to satisfy the constraints of the rendering hardware, or to satisfy the bandwidth or storage constraints. (b) Our method to determine the near optimal balancing between mesh and texture information in the multiplexed bitstream allow the system control module to tune the output rates of the encoders.

Finally, in Chapter 6 we investigated how to balance the mesh and texture information in our constrained environment. We showed the interplay between mesh and texture in Figures 6.10a-f: For the same resource constraint (the bitrate in this case), the models showed great disparities in quality. We proposed an efficient heuristic based on marginal analysis to obtain near-optimal balance of the information in the heterogenous bitstream transmitted over the network (Figure 1.6). This algorithm goes into the system control module and tunes the bitrate of the streams entering the multiplexing unit (Figure 7.2b). In the next section, we explore several possible improvements to our solution.

7.3 Future work and research directions

In the next sections, we suggest several future research directions:

7.3.1 Mesh compression in an operational rate-distortion sense

To obtain an estimate for the cost of encoding the mesh, we used recent results obtained in mesh compression [34]. These methods have been developed for meshes with irregular structure. Therefore no assumption is made on the connectivity of the mesh. In our case, our mesh connectivity is obtained using 4-8 connection. Therefore, its regularity allows for more efficient encoding schemes. To the author's knowledge, no work has been yet pursued for 4-8 meshes.

A compression scheme that takes advantage of our variable-rate optimization method, as well as our hierarchical data structure, would provide a means to encode meshes in an operational RD sense. Such an investigation is tightly related to the encoding of location codes for the semi-linear quadtree (Section 4.3.1). The goal of this investigation is the design of a monotonic cost functional evaluating the optimal cost to encode a subtree. Once designed, the new metric could be used without changes in the decimation algorithm presented in this thesis.

7.3.2 Joint approximation-compression

In the framework presented in Chapter 5, we approximate meshes in the following way: At each step we choose to decimate a vertex v and its attached merging domain M_v, where v is selected as the optimal candidate according to a metric. We could extend the possibilities as follows: Assume that we have a set of quantizers [31] for the vertices, then we could either *decimate* the vertex or *quantize it*. The metric should be designed to take the new choices into account. The result is a framework where, at each step, multiple choices for optimization are available. An appropriate cost functional in this case would be similar to the one designed for the problem in Section 7.3.1. Consequently, the range of available bitrates becomes larger. This problem actually solves an instance of a general problem of joint source-channel-rendering encoding constrained to meshes.

7.3.3 Joint encoding of mesh and texture

In Chapter 6, we presented an efficient method to select near-optimal mesh and texture pairs maximizing the rendered quality for a bit budget. To achieve this, we had to generate separately a set of view-dependent approximated meshes and a set of view-dependent approximated textures. This problem could be addressed differently using our simplification framework presented in Chapter 5. It can be seen as an extension to vertex attributes of the problem described in Section 7.3.2. Again, at each optimization step multiple choices are available. In this case, either M_v can be decimated and the mesh coarsened, or the texture resolution can degraded by selecting fewer transform coefficients. Using this method, the mesh and texture approximation is truly performed jointly and avoids the use of separate methods as done in Chapter 6.

7.4 Status and availability

Source code for the semi-linear quadtree has been publicly available [1] since November 30th 1999 at the address `http://lcavwww.epfl.ch/~balmelli/software/quadtree/index.html`. We set up a web site containing documentation, man pages and Java applets. Visitors are free to register to receive further notices. Source code for the mesh approximation algorithm has not been yet released, since it lacks some documentation.

7.5 Summary

From our results, we can see that most of our attention was spent on the geometry optimization aspect of our transmission system. Since well-established methods exist to generate embedded and compact descriptions of textures, we focused our attention on the remaining components of our system. We payed attention to generate models coping with our constrained environment. In particular, we proposed methods to efficiently generate progressive representation under constraints. We performed several experiments with large datasets. We provided a simplified description of a transmission system for computer graphics models and tried to identify important problems in this context.

[1]However, the code usage is restricted to academic research only and is protected in the United States by a pending patent [5].

Bibliography

[1] P.K. Agarwal and P.K. Desikan. An efficient algorithm for terrain simplification. *Proceedings ACM-SIAM Sympo. Discrete Algorithms*, pages 139–147, 1997.

[2] P.K. Agarwal and S. Suri. Surface approximation and geometric partition. *Proceedings of 5th ACM-SIAM Sympo. Discrete Algorithms*, pages 24–33, 1994.

[3] Dana H. Ballard. Strip trees: a hierarchical representation for curves. *Communication of the ACM*, 24(5):310–321, 1981.

[4] L. Balmelli, S. Ayer, and M. Vetterli. Efficient algorithms for embedded rendering of terrain models. *Proceedings of IEEE Int. Conf. Image Processing (ICIP)*, 2:914–918, October 1998.

[5] L. Balmelli, J. Kovačević, and M. Vetterli. Efficient processing of quadtree data. *US patent filed, Lucent Technologies, Bell Labs, Murray Hill (NJ), USA*, November 1999.

[6] S. Benninga. Numerical techniques in finance. *MIT Press, ISBN 0262521415*, 1989.

[7] S. Benninga and B. Czaczkes. Financial modeling. *MIT Press, ISBN 0262024373*, 1997.

[8] L. Boxer, C. Chang, R. Miller, and A. Rau-Chaplin. Polygonal approximation by boundary reduction. *Pattern Recognition Letters*, 14:111–119, 1993.

[9] L. Breiman, J.H. Freidman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. The Wadsworth Statistics/Probability Series, Belmont,CA; Wadsworth, 1984.

[10] D.M. Brophy. An automated methodology for linear generalization in thematic cartography. *Proceedings ACSM 33rd Annual meeting*, pages 300–314, 1973.

[11] E. Catmull. A subdivision algorithm for computer display of curved surfaces. *PhD thesis, Report UTEC-CSs-74-133, Computer Science Department, University of Utah*, December 1974.

[12] E. Catmull. Computer display of curved surface. *Proc. of the Conference on Computer Graphics, Pattern Recognition and Data Structures, Los Angeles*, pages 11–17, May 1975.

[13] P. Chou, T. Lookabaugh, and R. Gray. Optimal pruning with application to tree-structured source coding and modeling. *IEEE Transactions on Information Theory*, 35(2):299–315, March 1989.

[14] J.H. Clark. A fast algorithm for rendering parametric surfaces. *Proceedings of SIGGRAPH*, pages 289–99, 1979.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, algorithms and applications*. Springer-Verlag, 2000.

[16] Office Federal de Topographie. Pixelkarte 1:25000 cd rom 1,2,3. *CH-2084 Wabern*, Mai 1997.

[17] M. Deering. Geometry compression. *Proceedings of SIGGRAPH*, July 1995.

[18] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[19] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *Proceedings of IEEE Visualization*, 1997.

[20] R. O. Duda and P. E. Hart. Pattern classification and scene analysis. *Wiley, New York*, 1973.

[21] D. Ebert, W. Carlson, and R. Parent. Solid space and inverse particle systems for controlling the animation of gases and fluids. *The visual computer*, 10(4):179–190, 1994.

[22] D. Ebert and R. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline a-buffer techniques. *proceeding of SIGGRAPH*, 24:357–366, August 1990.

[23] D. Ebert (Editor), F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley. Texture and modeling, a procedural approach. *AP Professional, Academic Press*, 1994.

[24] G. Fekete. Rendering and managing spherical data with sphere quadtrees. *Proceedings of IEEE Visualization*, pages 176–186, October 1990.

[25] A. Finkelstein and D. H. Salesin. Multiresolution curves. *Computer graphics proceedings, Annual conference series*, 1994.

[26] Foley, van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, The Systems Programming Series, 1992.

[27] G. Gardner. Visual simulation of clouds. *proceeding of SIGGRAPH*, 19:11–20, July 1985.

[28] G. Gardner. Forest fire simulation. *proceeding of SIGGRAPH*, 24:430–436, August 1990.

[29] I. Gargantini. An effective way to represent quadtree. *Communications of the ACM*, 25(12):905–910, December 1982.

[30] M. Garland and P.S. Heckbert. Surface simplification using quadric error metric. *Proceedings of ACM Siggraph*, 1997.

[31] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[32] M.F. Goodchild and S. Yang. A hierarchical spatial data structure for global geographical information system. *CVGIP: Graphical Models and Image Processing*, 54(1):31–44, January 1992.

[33] Network Working Group. Rtp: A transport protocol for real-time applications. *Request for Comments (RFC 1889), available at http://www.cis.ohio-state.edu/htbin/rfc/rfc1889.html*, January 1986.

[34] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. *proceeding of SIGGRAPH*, pages 133–140, 1998.

[35] I. Guskov, W. Swelden, and P. Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH*, pages 325–334, 1999.

[36] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. *to appear in proceedings of SIGGRAPH*, 2000.

[37] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison Wesley, 1996.

[38] P. Heckbert. A minimal ray tracer. In P. Heckbert, editor, *Graphics Gems IV*, pages 375–381. Academic Press, Boston, 1994.

[39] P.S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[40] P.S. Heckbert and M. Garland. Multiresolution modeling for fast rendering. *Carnegie Mellon University http://www.cs.cmu.edu/ ph*, May 1997.

[41] P.S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. *Carnegie Mellon University Technical Report*, May 1997.

[42] B. Von Herzen and A.H. Barr. Accurate triangulation of deformed, intersecting surfaces. *Computer Graphics 21, also Proceeding of ACM SIGGRAPH 87*, 21(4):103–110, July 1987.

[43] H. Hoppe. Progressive meshes. *Proceedings of SIGGRAPH*, pages 99–108, 1996.

[44] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proceedings of IEEE Visualization*, pages 35–42, October 1998.

[45] S. Horbelt, F. Jordan, and T. Ebrahimi. Streaming of photo-realistic texture mapped on 3d surfaces. *Int. Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, Sept 1997.

[46] A. Horsley and A. Wrobel. Marginal analysis and bewley equilibria: The use of subgradients. *London School of Economics, 10 PORTUGAL STREET WC2A 2HA, United Kingdom. Theoretical Economics in its series London School of Economics - Suntory Toyota,Theoretical Economics*, 1991.

[47] H. Imai and M. Iri. Polygonal approximations of a curve - formulation and algorithms. *Computational Morphology - G.T Toussaint (editor), Elsevier Science Publishers*, 1988.

[48] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. *to appear in proceedings of SIGGRAPH*, 2000.

[49] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry*, 14(1-3):5–24, 1999.

[50] D. Koller, P. Lindstrom, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Virtual gis: A real time 3d geographic information system. *Proceedings of IEEE Visualization*, pages 94–100, 1995.

[51] K.Ramchandran and M.Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Transactions on Image Processing*, 2(2):160–175, April 1993.

[52] U. Labsik, L. Kobbelt, R. Schneider, and H.-P. Seidel. Progressive transmission of subdivision surfaces. *Computational Geometry*, 15(1-3):25–39, 2000.

[53] R. Laurini. Graphical databases built on peano space-filling curves. *Proc. Eurographics*, pages 327–338, 1985.

[54] F. Laves. Die Bau-zusammenhänge innerhalb der Kristallstrukturen. *Zeitschrift für Kristallographie*, 73:202–265, 1930.

[55] L.Balmelli, J.Kovačević, and M. Vetterli. Quadtree for embedded surface visualization: Constraints and efficient data structures. *Proceedings of IEEE Int. Conf. Image Processing (ICIP)*, 2:487–491, October 1999.

[56] M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadtrees. *Technical Report, Computer Science Dept., Center for Automation Research, University of Maryland*, CS-TR-3900, April 1998.

[57] J.G. Leu and L. Chen. Polygonal approximation of 2d shapes through boundary merging. *Pattern Recognition Letters*, 7:231–238, 1988.

[58] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH*, pages 109–118, 1996.

[59] C. Loop. Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics*, 1987.

[60] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[61] S. McCanne and M. Vetterli. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal on Selected Areas in Communications*, 15(6):983–1002, August 1997.

[62] S.B. Nadler. Hyperspaces of sets. *Marcel Dekker, New York*, 1978.

[63] N.Chiba, S.Sanakanishi, K.Yokoyama, I.Ootawara, K.Muraoka, and N.Saito. Visual simulation of water currents using a particle-based behavioral model. *The Journal of Visualization and Computer Animation*, 6(3), July-September 1995.

[64] J. Neider, T. David, and Mason Woo. *OpenGL Programming Guide*. Addison Wesley, 1993.

[65] A. Van Nevel. Texture synthesis via matching first and second order statistic of wavelet frame decomposition. *Proceedings of IEEE Int. Conf. Image Processing (ICIP)*, 1:72–76, 1998.

[66] ITU-T Telecommunication Standardization Sector of ITU. Low bitrate system for video telephony over high-speed modem lines(h.324). *Draft ITU-T Recommendation H.324*, 1995.

[67] ITU-T Telecommunication Standardization Sector of ITU. Multimedia system control (h.245). *Draft ITU-T Recommendation H.245*, 1995.

[68] ITU-T Telecommunication Standardization Sector of ITU. Multiplexing protocol for low bitrate multimedia communication (h.223). *Draft ITU-T Recommendation H.223*, 1995.

[69] M.A. Olivier and N.E. Wiseman. Operation on quadtree leaves and related image areas. *Computation Journal*, 26(4):375–380, 1983.

[70] E.J. Otoo and H. Zhu. Indexing on spherical surfaces using semi-quadcodes. *Advances in Spatial Databases - 3rd Int. Sym.*, pages 510–529, 1993.

[71] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. *Proceedings of IEEE Visualization*, pages 299–305, 1998.

[72] T. Pavlidis. Structural pattern recognition. *Springer-Verlag, Berlin*, 1977.

[73] R.E.Rosenblum, W.E.Carlson, and E.Tripp III. Simulating the structure and dynamics of human hair: modeling, rendering and animation. *The Journal of Visualization and Computer Animation*, 2(4), October-December 1991.

[74] H. Rushmeier, B. Rogowitz, and C. Piatko. Perceptual issues in substituting texture for geometry. *SPIE Proceedings, Human Vision and Electronic Imaging V, B. Rogowitz and T. Pappas, Editors*, 3959:372–383, 2000.

[75] H. Samet. Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing*, 18(1):37–57, January 1982.

[76] H. Samet. *Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS.* Addison-Wesley Publishing Company, 1990.

[77] D. Saupe. Optimal piecewise linear image coding. *SPIE Visual Communication and Image Processing (VCIP'98)*, 1998.

[78] C. Schlegel. *Trellis coding*. IEEE press, 445 Hoes Lane, P.O box 1331, Piscataway, NJ 08855-1331, 1997.

[79] C. Schlick. Fast alternatives to Perlin's bias and gain functions. In P. Heckbert, editor, *Graphics Gems IV*, pages 401–403. Academic Press, Boston, 1994.

[80] G. Schrack. Finding neighbors of equal size in linear quadtrees and octrees in constant time. *CVGIP: Image Understanding*, 55(3):221–230, May 1992.

[81] G. Schrack and I. Gargantini. Mirroring and rotating images in linear quadtree form with few machine instructions. *IVC*, 11:112–118, 1993.

[82] J. C. Shepherdson and H. E. Sturgis. Computability of recursive functions. *Journal of the Association for Computing Machinery*, 10(2):217–255, 1963.

[83] International Organization For Standardization. Overview of the mpeg-4 standard. *ISO/IEC, available at http://drogo.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm*, JTC1/SC29/WG11(N3156), December 1999.

[84] M. Tamminen and F.W. Jansen. An integrity filter for recursive subdivision meshes. *Computer Graphics 9*, 4(1):351–363, 1985.

[85] G. Taubin and Jarek Rossignac. Geometric compression through topological surgery. *Research Report RC-20340, IBM T.J. Watson Research Center, NY 10598*, January 1996.

[86] L. Velho and D. Zorin. 4-8 subdivision. *Mathematical Methods in CAGD. Vanderbilt University Press*, 2000.

[87] L. Velho and D. Zorin. 4-8 subdivision. *to appear in CAGD*, 2001.

[88] M. Vetterli and J. Kovačević. *Wavelets and subband coding*. Prentice Hall PTR, Englewood Cliffs, New Jersey 07632, 1995.

[89] E. R. White. Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–27, 1985.

[90] M.V. Wickerhauser. Adapted wavelet analysis from theory to software. *IEEE press, AK Peters, Wellesley, Massachusetts*, 1993.

[91] W.T.Reeves. Particle systems, a technique for modelling a class of fuzzy objects. *IEEE Computer Graphics*, 17(3):359–376, 1983.

[92] Z. Xiong, K. Ramchandran, and M. T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Transactions on Image Processing*, 6:677–693, May 1997.

[93] D. Zorin. A method for analysis of C^1-continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis*, 37(4):1677–1708, 2000.

[94] D. Zorin, P. Schroder, and W. Sweldens. Interactive mesh editing. *Technical Report, Computer Graphics Group, California Institute of technology*, CS-TR-97-06, 1997.

Acknowledgments

I would like to thank many people for their support and their help for making my life happier during my graduate studies:

My parents achieved so many things in their life that many would feel small. I think their point of view, the total freedom they left me and their constant example that, everything comes into focus when sufficient faith and work is involved, gave me the motivation to achieve this work. I am sure that my sister, Carole, will agree on all the above points. She will file her Ph.D in Biology next year.

I would like to thank Antonio Russo and Zoran Pecenovic for their everlasting friendship. I knew that I could always count on Zoran to share my research doubts. I also would like to thank him for his multiple readings and corrections of this thesis, as well as for other publications. Of course, Antonio was there to take care of different, but very important, types of issues. Both Antonio and Zoran have always been around and I feel somehow that I will never get rid of them.

A great friend during my studies was Ander Madinagoitia. We shared a lot of culture during his stay in Switzerland. With him, I learned what it was to be from *Pais Vasco*. Thanks also to Leire, Asier and Eulalia (ok, you're from Barcelona) for their friendship.

When I came back from the US, I was lucky to share my office with David Hasler. Even more enjoyable, we found out that we had a common passion for sport. As a result, we ended up to pass most of our noon breaks together at the sport center in Dorigny. I really appreciated our constant discussions and the shared interests we had for each other research problems.

Advising is a difficult task and I believe that my professor Martin Vetterli has done a great job. I appreciated the very friendly relationship we had during my stay. His wisdom was enlightening and he showed me the way to the ability of doing 10^8 things in parallel without cracking. The way he optimizes his time and resources would fill a thesis by itself.

Advising mister total stranger is even more challenging; But both Jelena Kovačević and Alexandra Mojsilović have done a great job while I was at Bell Labs. They both dedicated themselves to helping me and to making my oversea stay a success. I would like to thank them for their inifinite patience, their support and friendship.

Last but not least: even if we only had sporadic contacts during my thesis, he always listened very carefully and asked the right question. I have known Professor Thomas Liebling since my undergrade studies, but never thought I would sit once in his office to talk about research. I would like to thank him for his interest.

Finally, I would like to thank my labmates and the great secretaries from LCAV and DSC for being so helpful.

Curriculum vitæ

Laurent Balmelli

DI-LCAV-EPFL, Ecole Polytechnique Fédérale, 1015 Lausanne, Switzerland

Phone (+4121) 475 0224 (home); (+4121) 693 5671 (work)

E-mail: balmelli@lcavsun1.epfl.ch ; balmelli@acm.org

EDUCATION

1997 -now Ecole Polytechnique Fédérale, Lausanne, Switzerland. Ph.D in Communication Systems. Graduate courses in Advanced Signal Processing and Wavelets and Computer Networking and Traffic Control.

1992 -1996 Ecole Polytechnique Fédérale, Lausanne, Switzerland. M.S., Computer Science. 9.25/10.0 GPA overall (courses in Optimization, Distributed Systems, Computer Graphics, Networking, Neural Networks, Graph Theory), 10.0/10.0 on Master Thesis (topic: Visualization/Computation of Large Scale Molecular Dynamics Simulations).

REFERENCES

- Professor Martin Vetterli (http://lcavwww.epfl.ch/~vetterli/), Laboratory for Audio-Visual Communications, Ecole Polytechnique Federal, Lausanne, Switzerland. Contact - vetterli@lcavsun1.epfl.ch - tel. +4121 693 56 98 / +4179 277 37 19 - Thesis advisor

- Professor Thomas Liebling (http://rosowww.epfl.ch/lg/), Operational Research chair, Mathematical Dept., Ecole Polytechnique Federal, Lausanne, Switzerland. Contact - Thomas.Liebling@epfl.ch - tel +41 21 693 25 03 - Member of thesis jury

- Dr. Jelena Kovacevic (http://cm.bell-labs.com/cm/ms/who/jelena/index.html), Mathematical Science Center, Bell Labs, Lucent Technologies, Murray Hill, NJ, USA. Contact - jelena@research.bell-labs.com - tel: +1 908 582 6504 - Member of thesis jury, and advisor while at Bell Labs

- Professor Murat Kunt (http://ltswww.epfl.ch/staff/kunt.html), Signal Processing Laboratory, Ecole Polytechnique Federal, Lausanne, Switzerland. Contact - Murat.Kunt@epfl.ch - tel: +41 21 693 26 01 - Undergrade project advisor

LANGUAGES

English : Fluent.

French : Fluent.

Italian : Fluent.

Spanish : Fluent.

Also studied German (7 years), Portuguese (1 year). Currently learning Japanese.

EXPERIENCE

Mar 1997-now Ecole Polytechnique Fédérale, Lausanne, Switzerland

PhD student

Laboratory for Audio-Visual Communications. Thesis: Computer Graphics Models for Communications.

Research on progressive meshes for transmission, source-rendering coding algorithms for joint transmission of meshes and textures, spatial data structure for storage and access of large terrain databases.

Teaching assistant in signal processing - implementation of a distance learning framework in Java.

Advisor: Martin Vetterli - vetterli@lcavsun1.epfl.ch - tel. +4121 693 56 98 / +4179 277 37 19

2 U.S. patents filed, 1 journal paper (submitted), 7 conference papers.

Sept 1998-Jul 1999 Bell Labs, Lucent Technologies, Murray Hill, NJ, USA

Intern

Multimedia Research Laboratory and Mathematical Sciences Center (Mathematics of Communication), research in transmission of computer graphics 3D meshes and textures. In particular: developement of an efficient data structure for spatial data, and a texture discrimination method.

Reference: Jelena Kovacevic - jelena@research.bell-labs.com - tel: +1 908 582 6504

Apr 1996-Mar 1997 Ecole Polytechnique Fédérale, Lausanne, Switzerland

Software Engineer

Laboratory for Audio-Visual Communications. C++ Development of H.324 Low bitrate video codec (European Project VIDAS). The implementation of H.263, G.721 was based on existing implementation. The implementation of H.223 and H.245 was done from scratch.

Oct 1995- Mar 1996 Silicon Graphics Inc. (SGI), Center of Supercomputing Chemistry, Basel, Switzerland

Master Thesis Internship

Development of software for computation of molecular dynamics simulations and visualization on a cluster of four Power Challenge with 18 processors each. Developed an adaptive sampling method to track matter density variation through simulation. The project was done in collaboration with the Computer Graphics Laboratory at the Ecole Polytechnique. See publication (10).

Oct 1994 - July 1995 Ecole Polytechnique Fédérale, Lausanne, Switzerland

Undergraduate Research

Signal Processing Laboratory (LTS-EPFL). Prof. Murat Kunt. Implementation and optimization for a MPEG4 coder on a Cray T3D massive parallel computer with 256 processors. (graded 10.0/10.0, awarded semester's best undergrade student of LTS).

Theoretical Computer Science Laboratory. Design and Implementation of a C++ architecture for distributed software (on computer cluster). Project achieved in a local company (Linkvest Inc.) (graded 9.5/10.0).

Jul.-Sept. 1994 Cray Research, Inc. Eagan, Minnesota, USA

Summer student internship

Development of a networking monitoring tool for tape backup systems.

PATENTS AND PUBLICATIONS

1. L. Balmelli, J. Kovacevic and M. Vetterli, Efficient Processing of Quadtree Data, US patent filed, November 1999.

2. L. Balmelli, A. Mojsilovic, Method and Apparatus for Texture Analysis and Replicability Determination, US patent filed October 1999.

3. L. Balmelli, J. Kovacevic and M. Vetterli, Progressive Meshes in an Operational Rate-Distortion Sense, to appear in IEEE Transactions on Visualization and Computer Graphics, also EPFL/LCAV Tech. Report no DSC/2000/019

4. L. Balmelli, J. Kovacevic and M. Vetterli, Solving the Coplanarity Problem of Embedded Regular Triangulations, Proceedings of Vision, Modeling and Visualization, November 1999, Erlangen, Germany

5. L. Balmelli, J. Kovacevic and M. Vetterli, Quadtrees for Embedded Surface Visualization: Constraints and Efficient Data Structures, Proceeding of the IEEE International Conference on Image Processing (ICIP), October 1999, Kobe, Japan.

6. L. Balmelli and A. Mojsilovic, Wavelet Domain Features for Texture Description, Classification and Replicability Analysis, Proceeding of the IEEE International Conference on Image Processing (ICIP), October 1999, Kobe, Japan

7. L. Balmelli, S. Ayer and M. Vetterli, Efficient Algorithms for Embedded Terrain Simplification, Proceeding of the IEEE International Conference on Image Processing (ICIP), October 1998, Chicago, USA

8. L. Balmelli, S. Ayer, Y. Cheneval and M. Vetterli, A Framework for Interactive Courses and Virtual Laboratories, Proceedings of Multimedia Signal Processing conference, December 1998, Portofino, USA.

9. Y. Cheneval, L. Balmelli, P. Prandoni, M. Vetterli and J. Kovacevic, DSP Education using Java, Proceeding of ICASSP 98, Seattle, USA.

10. L. Balmelli, Adaptive Sampling of Very Large Particle Systems using an incremental SOFM, Proceedings of Eurographics, June 1996, Poitier, France.

14. Cvetković, Zoran. *Overcomplete Expansions for Digital Signal Processing.* UCB/ERL M95/114, December 1995.

15. McCanne, Steven Ray. *Scalable Compression and Transmission of Internet Multicast Video.* Coadv. with Van Jacobson, Lawrence Berkeley National Laboratory. UCB/CSD 96/928, December 1996.

16. Goodwin, Michael Mark. *Adaptive Signal Models: Theory, Algorithms, and Audio Applications.* Coadv. with Edward A. Lee, EECS, U.C. Berkeley. UCB/ERL M97/91, December 1997.

17. Goyal, Vivek K. *Beyond Traditional Transform Coding.* UCB/ERL M99/2, September 1998.

18. Chang, Sai-Hsueh Grace. *Image Denoising and Interpolation based on Compression and Edge Models.* Coadv. with Bin Yu, Statistics, U.C. Berkeley. UCB/ERL M99/57, Fall 1998.

The Lausanne time

19. Prandoni, Paolo. *Optimal Segmentation Techniques for Piecewise Stationary Signals.* EPFL 1993(1999), June 1999.

20. Lebrun, Jérôme. *Balancing MultiWavelets.* EPFL 2192(2000), May 2000.

21. Weidmann, Claudio. *Oligoquantization in Low-Rate Lossy Source Coding.* EPFL 2234(2000), July 2000.

22. Balmelli, Laurent. *Rate-Distortion Optimal Mesh Simplification for Communication.* EPFL 2260(2000), September 2000.