UNIVERSITÀ DEGLI STUDI DI PISA
DIPARTIMENTO DI INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA

Scientific Visualization based on Simplicial Complexes

Paolo Cignoni

March 1998

Istituto per l'Elaborazione dell'Informazione - Consiglio Nazionale delle Ricerche,
Via S. Maria, 46 - 56126 Pisa ITALY - Email: cignoni@iei.pi.cnr.it

Abstract

The main objective of the thesis is the development of a complete and efficient set of solutions for the Scientific Visualization (SciViz), the computer science field which deals with the study, design and implementation of algorithms and data structures for the visualization of scientific data. We propose the adoption of the simplicial complexes as the unifying geometric structure and we show how the choice of this structure as kernel geometric primitive is effective both for the theoretical and practical aspects of SciViz.

The contents of the thesis can be summarized as follows. We interpret the diversified SciViz process as a two-steps mapping problem: a modeling step, in which data are mapped into geometry with visual attributes, and a rendering step where geometry and visual attributes are transformed into images. As unifying geometric structure for the modeling step we propose the adoption of the simplicial complexes. To validate our approach we define new algorithms and data structures for the SciViz problems, based on the use of the simplicial complex as basic geometric representation scheme. In particular the thesis supplies original solutions and results to the following problems:

1. Visualization of scalar volume datasets: optimizing techniques for the extraction of isosurface and for the direct volume rendering;
2. Depth sorting of simplicial complexes (a fundamental topic for the efficient and correct use of direct volume rendering based on projective techniques);
3. Integration of isosurface extraction techniques and direct volume rendering techniques. Definition of the concept of discontinuous transfer functions;
4. Simplification of the geometric complexity of simplicial complexes (in order to speed up the visualization process) while minimizing the introduced error;
5. Multiresolution representation schemes for simplicial complexes. These schemes permit both the visualization of complexes at different levels of details and the visualization of a single complex in which different parts are at different resolution.

Contents

Aknowledgements

I wish to thank my advisors Leila De Floriani and Claudio Montani for their invaluable help. Moreover, a special thank to Roberto Scopigno for the many productive suggestions and discussions. I wish also to thank Enrico Puppo for the collaboration in the development of the multiresolution models. A particular word of aknowledgment goes to the many students that contributed to the coding of the ideas here presented: Giuseppe Lai, Claudio Rocchini, Raffaele Dell'Aversana, Andrea Ciampalini, Davide Costanza, Paola Marino, Pierluigi Sbaiz, Marco Servettini, Donatella Sarti.

My final thanks go to my wife Gabriella, whose patience and encouragement in preparing this thesis made it possible for me to finish the task.

Chapter 1

Introduction

Due to the availability of high resolution monitoring systems, complex simulation models, and powerful graphics devices, the use of visualization techniques for the analysis and understanding of natural phenomena is becoming increasingly important in a wide range of research and application fields. The term *Scientific Visualization* (SciViz) describes the field of computer science which deals with the study and definition of algorithms and data structures for the visualization of scientific data. SciViz aims at helping the comprehension of natural phenomena, promoting the decision making process and supporting a better analysis of large amounts of experimental and simulated data.

Though a lot of terms, techniques or data structures used in SciViz have been borrowed from Computer Graphics, there is a deep conceptual difference between the two disciplines: the goal of Computer Graphics is to reproduce a well-known phenomenon or scene with high realism. the goal of SciViz, is the comprehension of a phenomenon or a process through its visualization. Moreover, input data in Computer Graphics are well defined descriptions of scenes; the objects to be rendered are represented in some analytic or geometric form. Data in SciViz do not provide for the formal specification of objects: the investigated phenomenon is described by means of a large n-dimensional cloud of data in a scalar, vectorial or tensorial form.

Unfortunately, as often occurs in new research fields, SciViz is far from being a well-founded topic of computer science. Most of the existing techniques have been developed to solve specific problems and, generally, they depend on the nature of the data they deal with; the bibliography is wide but almost unstructured.

Overcoming these limitations and giving to the SciViz field a more consistent fundation are the main objectives of this thesis. To get this ambitious result we propose:

- An original intepretation of the diversified SciViz process as a two-step mapping problem: a *modeling* step in which data are mapped into geometries with visual attributes, and a *rendering* step where geometries are transformed into images. This interpretation permits us to break up complex problems into more precisely definite components and to compare different solutions in an uniform way.

- The adoption of an unifying geometric structure for the SciViz problems: the *simplicial complex*. We will show how the choice of the simplicial complex as kernel geometric primitive is effective both for the theoretical and practical aspects of SciViz.

- The definition of new algorithms and data structures (or the re-writing of existing ones) for the SciViz, based on the use of the simplicial complex as a basic geometric representation scheme. Our goal is not to propose an efficient solution for a specific SciViz problem: our goal is to define algorithms and data structures which permit to efficiently solve classes of problems in the complex and multivarious visualization process.

Simplicial complexes do not represent a new concept in computer science: triangulations in Computational Geometry, unstructured grids in finite elements, triangular irregular network in Geographical Information Systems, simplicial complexes in Algebraic Topology are just some examples of the use of this structure in different research fields. It seems quite natural to use this geometric structure systematically in the visualization process.

Moreover, simplicial complexes are dimension independent and this property makes it possible their use as a basic general tool for data modeling [79] and for the definition of multiresolution representation schemes [34]: algorithms and data structures, which adopt them as kernel geometric structure, can be easily adapted to any specific dimension.

We will focus our attention mainly to the three-dimensional case and therefore we will adopt the tetrahedron as basic geometric primitive. This choice does not represent a limitation to the applicability of the algorithm and will certainly make the exposition more clear. Moreover, most of the SciViz problems are three-dimensional problems: the definition of 3D algorithms and data structures makes them immediately usable for real applications.

1.1 Problem Statement and Data Classification

A better understanding of the SciViz can be obtained by defining the data it deals with. In this section we give some definitions and examples of the data sources.

1.1.1 Input Data

Let us consider the data to be visualized as a function f, defined for a finite set V of discrete points v_i over a domain $\Omega \subset \mathbb{E}^{\mathrm{d}}$. (V, f) is generally known as a dataset. No initial assumption on the range of the function f is made. From a mathematical point of view, we can manage all the different cases by means of hypersurfaces (i.e. functions $f : \mathbb{R}^d \to \mathbb{R}$), whereas from a perceptual point of view, it is different to visualize d independent scalar fields from visualizing a vectorial field. We classify the data in SciViz according to their *domain*, *range* and *structure*.

Domain In most cases, data to be visualized are distributed in a domain $\Omega \subset \mathbb{E}^{\mathrm{d}}$ with $d \leq 3$. The simplest case is the two-dimensional one: the domain is a plane or a closed two-manifold [42]. Visualizing discrete three-dimensional data is generally known as volume visualization [60]; the set of points and their associated values are known as a volumetric dataset. The cases in which the examined data are defined in $\mathbb{E}^{\mathrm{d}}$ spaces, with $d > 3$, obviously require different visualization technniques [10, 11, 53]. Moreover, data defined in $\mathbb{E}^{\mathrm{d}}$ space and varying in time can be simply handled considering the data as distributed in E^{d+1} space.

Range For each point $v_i \in \Omega$ a quantity $f(v_i)$ is given; this quantity can be scalar, vectorial or tensorial; it must be noted that visualization techniques are chosen according to the range of data. In fact visualizing a vector field in 3D is different from visualizing three independent scalar fields. This thesis deals with scalar fields.

Topology and Geometric Structure Along with the data a *structure* characterizing in some way the distribution of points v_i is often given. This characterization can be geometrical (i.e. all the points are distributed on an deformed regular grid) and topological (i.e. there exists a structure that, given a point v_i, permits to retrieve the points of V *neighbor* to v_i.)

The topological structure can be implicitly defined; for example, if the data point are distributed on three-dimensional regular grid it is easy to find the neighbors of a given point. An other example of the topological structure can be a simplicial complex having the points v_i as vertices. If such a topological structure does not exist it can be created. The existence of such a structure is exploited in the visualization process.

In the case of a three dimensional domain, the following volumetric dataset classifications are common:

- *regular*: data points are regularly distributed in E^3 on a three dimensional rectilinear grid;
- *curvilinear*: data points are distributed in a deformed (warped) three dimensional grid;
- *unstructured*: data points have no "regular" geometric structure but there is a topological structure;
- *scattered*: data points are scattered with no correlation between them.

In the case of regular and curvilinear datasets the geometric structure of the dataset is implicitly defined; scattered datasets generally imply the creation of a topology by means of $\mathbb{E}^{\mathrm{d}}$ triangulation techniques (for example Delaunay triangulations).

The algorithms and data structures in the following chapters will address mainly the case of three dimensional scalar data, even if most of the proposed techniques are general and can be extended to higher dimensions.

1.1.2 Data Sources

Many research fields can create data for analysis in SciViz. In a lot of sciences, non–invasive inspection techniques provide sources of regular volumetric scalar datasets. Modern techniques of investigation [86] such as CAT, MRI, PET can be used to produce volumetric datasets representing parts of the human body.

Computational fluid dynamics, the study of flows and fluids, is a common source of volumetric vectorial and tensorial datasets. Molecular modeling, the study of new compounds, geosciences and the study of seismic structures are other common sources of volumetric data.

Mathematics and the understanding of high dimensional relationships are the most important sources of datasets with domains in E^d with $d > 3$. Another source of data to be visualized are terrains; their efficient visualization plays a key role in flight simulators. Although terrain visualization is not generally included in SciViz, it can be treated as one: terrain data can be considered as scalar fields defined in a two–dimensional domain (height fields). A typical case in which the data are distributed over a closed two–manifold is, for example, where data are distributed over a sphere (e.g. ozone distribution). Although this problem could be solved by projecting the data over a plane, visualization of the data over a sphere improves the user's understanding.

1.1.3 Data Reduction Techniques

Interactive visualization techniques greatly improve the comprehension of three-dimensional structures. The spatial ambiguities, that may appear from a single image of a three dimensional scene, immediately disappear if the user is able to interactively rotate/translate the 3D scene. The possibility to interactively manage visualization is therefore a must, and this imposes the availability of enormous computational resources. A common technique to speed up rendering, beside the use of parallel hardware [73], is to reduce the quantity of data to be visualized. Such a reduction process is obviously aimed to limit the error introduced by the elimination of part of the data. This technique is increasingly used in SciViz as well [64, 25, 19].

Most of the work in this field covers the simplification of two dimensional surfaces, e.g. terrains or boundary representation of 3D objects [44, 32, 89, 18]. Some of the work in this field originated from the need of reducing the spatial complexity of isosurfaces produced in visualizing large volumetric datasets [89, 74].

Once we have models with different resolutions we must face the problem of managing them. Multiresolution modeling has as objective to structure models at different resolutions into a comprehensive framework that allows data to be manipulated at different resolutions according to the needs of a given application or task.

Data reduction techniques for tetrahedral complexes and multiresolution models for SciViz will be addressed in Chapter 5 and 6, respectively.

1.2 Simplicial Complexes as a basis for Scientific Visualization

In spite of a substantial progress in the last few years, the data structures and algorithms developed for Scientific Visualization still suffers for poor integration or generality. One of the goals of this thesis in the process of standardization of this discipline is the proposal of adopting simplicial complexes as the unifying kernel structure in the representation of geometrical data. We justify this choice in the following both in terms of generality (by showing how a number of different subproblems might be solved adopting a simplicial representation) and of efficiency of the solutions designed.

Some of the characteristics suggesting us the adoption of simplicial complexes as the kernel data representation structure in SciViz are briefly introduced in the following:

- they are a structure suitable to model data in any dimension;

- it is possible to find simplicial decompositions of domains containing scattered data points;
- they are a suitable basis for many interpolation techniques [43];
- data structure design is simplified;
- handling degenerated cases (i.e. coincident points in 3D space) is simplified, because a simplex degenerates to a lower-dimension simplex;
- visualization algorithms are in general faster, integration of different techniques is simpler and can be accelerated by adopting a multiresolution approach.

Among the positive aspects of the simplicial complexes we must highlight their inherent dimensionality-independence: it allows simple interchange of solutions between classes of problems embedded in spaces at different dimensionality (i.e. solving problems in E^d space by extending solutions to similar E^{d-1} problems).

In the next sections we give some formal definitions and a more detailed justification to our assertions, by showing how simplicial complexes might be the unifying structure for a number of visualization problems in E^2 and E^3.

Common techniques for terrain or surface representation and rendering adopt triangular facets as kernel elements. The graphics hardware available nowadays on most of the workstations was designed identifying the "triangular mesh" as the basic primitive. While in the 2D case the use of triangular meshes is now well estabilished, in the visualization of 3D data there is not such a common approach. In the case of Volume Visualization there is a lot of good reasons to use simplicial complexes (actually tetrahedral complexes):

- most cell complexes found in Volume Visualization are easily decomposable in a simplicial complex (even implicitly, in order to avoid the growth of datasets);
- simplicial complexes can be easily rendered using various classes of algorithms (e.g. ray-tracing, scan-line, or projective approaches);
- isosurface extraction from simplicial complexes avoids the ambiguities that occur with hexahedral complexes;
- most of the rendering algorithms for irregular cell complexes are simpler to be described and implemented on tetrahedral complexes;
- integration of different techniques is simpler and it can be accelerated by adopting a multiresolution approach.

It should be noted that tetrahedral complexes in visualization are more and more used; they are becoming a standard for the visualization of unstructured datasets; there are, for example, efficient and robust techniques [4] for reducing various classes of datasets into tetrahedral complexes.

In this thesis we propose the tetrahedron as the volume rendering primitive element in the same way as the triangle is the most common primitive for surface rendering. It must be noted that this aim is much more than the simple use of tetrahedral complexes for solving some problems in Volume Visualization; the final goal should be the fundation of a complete set of general purpose rendering techniques that are tetrahedron-centric, able to manage all the classes of presented data and efficient enough to make their practical use suitable.

Our approach to Volume Visualization based on tetrahedra could resemble the Kaufmann's proposal for Volume Graphics [61] as a subfield of Computer Graphics. In Kaufmann's approach the base element is the voxel and Volume Graphics is concerned with synthesis, manipulation and rendering of 3D objects stored as volumes of voxels. Beside the different choice of the basic element an all the advantages of using an adaptive, all purpose structure, the strenght of our approach lies on the complete dimension independence of the structure: simplicial complexes work in any dimension and the same holds for the associated management techniques. For example, the multiresolution techniques conceived for generic simplicial complexes (like the ones described in Chapter 6) work for both terrains and volume data.

1.3 Data modeling based on Simplicial Complexes

In this Section, we formally introduce simplicial complexes, describing their properties and how they can be used to represent volumetric scalar datasets.

Consider a set $V = \{v_0, v_1, \ldots, v_d\}$ of $d+1$ linearly independent points in the k-dimensional Euclidean space $\mathbb{E}^{\mathrm{k}}$, with $d \leq k$. The subset σ of $\mathbb{E}^{\mathrm{k}}$, formed by all points which can be expressed as linear convex combination of the points of V, is called a *d-simplex.* The points of V are called *vertices* of σ, while d is the *order* of σ. Any s-simplex ξ, $0 \leq s \leq d$, which is generated by a subset of $s+1$ vertices of σ, is called an *s-face* of σ. If $s < d$, then ξ is called a *proper face* of σ.

A collection Σ of simplices is called a *d-simplicial complex* when the following conditions hold:

- for each simplex $\sigma \in \Sigma$, all the faces of σ belong to Σ;
- for each pair of simplices $\sigma, \tau \in \Sigma$, either $\sigma \cap \tau = \emptyset$ or $\sigma \cap \tau$ is a proper face of both σ and τ;

- d is the maximum of the orders of the simplices belonging to Σ (d is called the *order* of Σ).

The union of all s-simplices of Σ, with $0 \leq s \leq d$, regarded as a point set, is called the *domain* of Σ, and denoted Ω); any proper face of a simplex of Σ is called a *boundary face* if it belongs to the boundary of Ω, an *internal face* otherwise.

In practice, d-simplices are used as building blocks to cover the domain. Boundary faces form the boundary of the domain, while internal faces separate such blocks from one another. If general polyhedra are used as building blocks, instead of simplices, the previous concepts can be generalized to define a *cell complex*. We will not formalize this concept for the sake of brevity, since in this work we just use simplicial complexes. Nevertheless, we wish to point out that a cell complex is a very general structure that can be used to formalize digital hypersurface models: such use will be informally discussed in Section 1.3.1.

An advantage of *simplicial complexes* with respect to the more general *cell complexes* can be found in the design of data representation schemes; this is because:

- a d-simplicial complex Σ is fully characterized by its combinatorial description plus the coordinates of its vertices;

- any simplex σ implicitly defines all its faces;

- the number of k faces of a d-simplex is a constant;

- the combinatorial structure of Σ is completely characterized by the list of its top simplices; if Σ is regular it is characterized by the list of its d-simplicies.

A special class of d-simplicial complexes is the one of the Delaunay complexes. A d-simplicial complex Σ in $\mathbb{E}^{\mathrm{d}}$ is called a *Delaunay simplicial complex* if and only if it covers the convex hull of its vertices and the hypersphere circumscribing each d-simplex of Σ does not contain any vertex of Σ in its interior. For a given set $V \subset \mathbb{E}^3$ of n points ($n \geq d+1$), there exists a unique Delaunay simplicial complex having V as vertex set if and only if there are no $d+2$ points of V that are cospherical.
A Delaunay simplicial complex can be built on any set of vertices V, and the shape of its simplices is the most regular among all possible simplicial complexes built on V. Moreover, efficient algorithms have been proposed for the construction of the Delaunay simplicial complex, given as set of vertices V [98].

From the visualization point of view, a number of rendering algorithms able to handle simplicial complexes exist. The use of simplicial decompositions to manipulate and render curvilinear dataset implies an increase in the number of cells (at least 5 simplices for each hexahedral cell), but simplifies problems caused by occasional degenerate cells (e.g., non-hexahedral cells due to coincident sites) and cells with non-planar faces. Handling such cases is generally more complex if a non-simplicial cell decomposition is used. In fact, many visualization algorithms have been described with high generality (i.e. they have been specified "on the paper" for general convex cell complexes), but they have often been implemented only on simplicial complexes.

In the following, we deal with 3-simplicial complexes in $\mathbb{E}^3$, called also *tetrahedralizations.*

1.3.1 Digital Hypersurface Models

A formal approach to volume dataset modeling, in order to express both geometry and field values, is to embed the volume dataset in $\mathbb{E}^4$ and to consider it as a four-dimensional hypersurface, defined piecewise over a decomposition of the domain into cells.

Given a domain $\Omega \subseteq \mathbb{E}^3$, and a function $f : \Omega \rightarrow \mathbb{E}$, the hypersurface corresponding to f over Ω is $\psi(\Omega, f) = \{(x, y, z, f(x, y, z)) \mid (x, y, z) \in \Omega\}$. We call the pair $\mathcal{M} = (\Omega, f)$ a *mathematical hypersurface model.* In practical applications, function f will be known only at a finite set of points in the domain Ω. Let $S = \{s_1, \ldots, s_N\}$ be a finite subset of Ω, at which f is known, called a *dataset.* The pair $\mathcal{M}_S = (S, f|_S)$ is called a *sampled model* (of f). Here, we consider the problem of building a digital model on Ω approximating a given sampled model $\mathcal{M}_S$ with a certain precision.

It is hardly possible to describe a hypersurface ψ by means of a single analytic function over the whole domain Ω. A finite description can be given by tessellating Ω into cells, such that ψ can be described by a function over each of such cells. To this aim, we define a digital hypersurface model as a special case of a mathematical hypersurface model, where function f is defined piecewise over a cell complex Γ, having a set of 3-cells $\{\gamma_1, \ldots, \gamma_m\}$. Thus, a *digital hypersurface model* is a pair $\phi = (\Gamma, F)$, where $F = \{f_1, f_2, \ldots, f_m\}$, with m number of cells of Γ, such that each function f_i is defined over a 3-cell γ_i of Γ. If f represents a continuous function, all f_i's must have the same value on the common faces of adjacent cells, otherwise suitable extensions of the f_i on the proper faces of Γ should be specified.

A simple example of digital hypersurface model is the well-known *voxel model.* In this case, the dataset S is a grid of regularly spaced points in $\mathbb{E}^3$, and the cell complex is composed of cubes: each cube γ_i is centered on a datum s_i and the function f_i is constantly equal to $f(s_i)$ over γ_i. Such a model represents all data in $\mathcal{M}_S$ exactly, and requires a number of cells (voxels) equal to the cardinality of S. A similar case can also occur for tetrahedral complexes: many finite analysis simulation algorithms operating on tetrahedral domains associate values directly to the cells, and not to their vertices.

In the rest of our thesis we will use tetrahedral complexes to represent volume data so we adopt the following hypersurface model; given a sampled model $\mathcal{M}_S = (S, f|_S)$, a *Tetrahedral Hypersurface Model* (THM) is a digital hypersurface model $\phi = (\Sigma, F)$ having the following properties:

1. the set V of vertices of Σ is the set of sample points S;

2. Σ is a tetrahedralization of V;

3. all functions of F are linear interpolants of f at the vertices of V: all such functions are uniquely defined by the values of f on V, denoted by $f(V)$, and the hypersurface ϕ is continuous and piecewise linear.

It should be noted that the choice of linear interpolation function is motivated by simplicity but higher order interpolation functions can be adopted [117].

1.4 Outline of the Thesis

In Chapter 2 we state our interpretation of SciViz as a two-step mapping problem: a *modeling* step where data are mapped into geometries with visual attributes, and a *rendering* step where geometries are transformed into images. Following this interpretation we give a general survey of the main techniques for visualizing a three-dimensional scalar field described with a tetrahedral complex. Then we describe the state of the art and our contributions in the field of isosurface extraction and direct volume render of a tetrahedral dataset.

Direct volume rendering (DVR) requires efficient depth sorting of the geometric primitive thus in Chapter 3 we describe in detail the problem of depth sorting a tetrahedral complex and we introduce a new technique for sorting complexes which belong to the class of projective complexes. The approach is based on the preliminary construction of the *lifted* complex corresponding to the given one and on its representation as a power diagram. This approach exhibits a $O(n \log n)$ runtime complexity to sort a complex and require only linear storage

Isosurface extraction and DVR are, for many aspects *complementary* techniques: they give to user different kind of information. In Chapter 4, we present an original solution to correctly integrate, using projective techniques, isosurfaces and DVR. The proposed technique is based on a tabular on-the-fly decomposition of tetrahedral cells along isosurfaces. In the second part of the Chapter we introduce the concept of Discontinuos Transfer Function: it unifies, in a single framework, the managment of visualization of volume data in which both isosurface extraction and DVR techniques are used.

Very often datasets are so large that they cannot be rendered interactively, for this reason in Chapter 5 we discuss how to build smaller datasets using simplification techniques. Two simplification algorithms for tetrahedral meshes are proposed.

A collection of simplified models can be managed in a single unified framework by adopting a multiresolution representation; in this way we can adapt the resolution and the size of the dataset to the needs of the user. In Chapter 6 we show how these methods permit the compact representation of many different approximations of the dataset. Multiresolution techniques allow, for example, the use of low resolution models for interactive phases and the extraction of variable resolution representations according to viewing parameters and/or to the user specification of a particular region of interest. Using the existing framework of Multiresolution Simplicial Model (MSM) we introduce the original concept of Hyper Simplicial Complex (HySC) that codifies a MSM in $\mathbb{E}^{\mathrm{d}}$ as a simplicial complex in $\mathbb{E}^{\mathrm{d+1}}$. In particular we propose an algorithm for the extraction of a variable resolution model with the full face-adjacency topological relation. We also give some details on the effective use in volume visualization of multiresolution, and in particular, on the use of variable resolution representations.

Chapter 7 concludes the thesis and presents some open research issues.

Chapter 2

Tetrahedral Volume Visualization

In this Chapter we state our interpretation of SciViz as a two-step mapping problem: a modeling *step where data are mapped into geometries with visual attributes, and a* rendering *step where geometries are transformed into images. Following this interpretation we give a general survey of the main techniques for visualizing a scalar field described with a tetrahedral complex. Then we describe the state of the art and our contributions in the field of extracting isosurfaces and directly render a tetrahedral volume dataset.*

The purpose of scientific visualization is to help the user to understand the *structure* of data under analysis through its visual representation. Observing in the most general way this visualization process we can identify two distinct steps: in the first one, called *the modeling step*, a meaningful geometric structure is extracted from our data; such structure, composed for example of triangles or lines, together with its visual properties (like color and transparency) exhibits some relevant features of the dataset. In the second step of the visualization process, called *the rendering step*, we can exploit the algorithms, tools and techniques supplied by computer graphics to *render* the previously extracted geometric structure. In other words during the first step we transform the data precisely into coloured geometries, then we render them into images.

In the next section we will justify/verify this interpretation by surveying the existing techniques for the visualization of a scalar field and identifying the two steps in them. In Section 2.2 and 2.3 we will describe in detail the two major visualization techniques, Isosurface Extraction and Direct Volume Rendering; particular attention will be payed in section 2.2 (in which we will

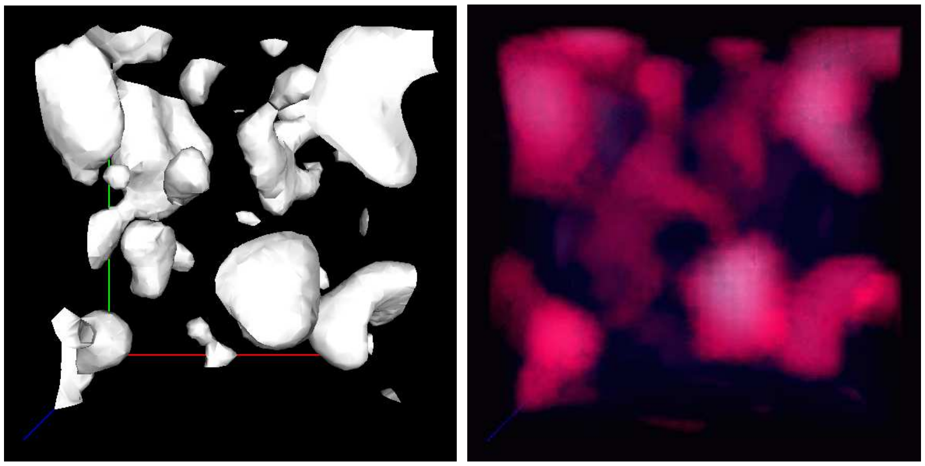


Figure 2.1: Isosurface and direct volume rendering of the same dataset.

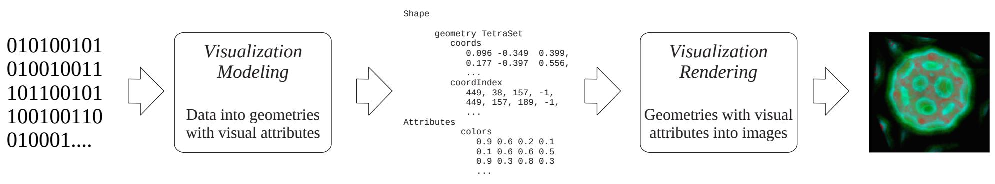
discuss modeling and rendering problems of isosurface extraction) to our contribution to the optimal solution of the problem of purging non-active cells during isosurface extraction.

2.1 Modeling vs. Rendering in Volume Visualization

Many visualization strategies have been proposed to reveal the inner structure of a three-dimensional scalar field. Here we shortly survey these visualization techniques, describing them in terms of a modeling/rendering process.

Slicing: Using a common visualization technique the values of the field are mapped with colors, so that the domain of the dataset is modeled as a solid object whose interior is colored according to the field (like a sort of agate). The dataset is then *sliced* in order to reveal the colors in the inner parts of the domain [95]. Only a 2D subset of the information in the dataset is visualized for each slice; generalized cutting plane/surfaces can be defined by the user to improve the power of this technique.

Isosurface fitting: In this case the objective of the modeling step is the reconstruction of polygonal surfaces representing, in most cases, an approximation, of an isosurface, the subset of points of the domain Ω where the value of the field is equal to a given threshold value. Another example of the modeling step in isosurface extraction is the work done by Interrante



```
Shape

      geometry TetraSet
         coords
            0.096 -0.349  0.399,
            0.177 -0.397  0.556,
            ...
         coordIndex
            449, 38, 157, -1,
            449, 157, 189, -1,
            ...
Attributes
           colors
              0.9 0.6 0.2 0.1
              0.1 0.6 0.6 0.5
              0.9 0.3 0.8 0.3
              ...
```

Figure 2.2: Separating modeling from rendering in Visualization.

[55, 100] on improving the comprehension of transparent isosurfaces where opacity textures are applied on isosurfaces as an "artistic device" for indicating their shapes and its essential features more explicitly.

Scalar Topology Graph Another form of visualization modeling, introduced for the two dimensional case by Bajaj and Schikore in [8], consists in the reconstruction of the scalar topology graph obtained by connecting all the critical points (minima, maxima and saddles) of the field. This approach can be seen as the *dual* of isosurface extraction: the arcs of the graph follow the gradient of the field and are orthogonal to the isocontours; this graph shows complementary information w.r.t. the isocontours: a quantitative view of the general behaviour of the field instead of the distribuition of a single value over the domain.

Direct Volume Rendering: The idea of mapping the field value into colors is extended to include also opacity: in the modeling step we transform our dataset into a semitransparent blurry object, where the color and opacity of each point precisely reflects, and therefore visualizes, the field value. In the case of Direct Volume Rendering the distinction between modeling and rendering is blurred by the fact that before the rise of Volume Visualization, the rendering of translucent solid objects received little attention, so that many of the algorithms for DVR were developed as an answer to a specific need in Scientific Visualization, rather than in the quest for photorealism.

This interpretation of Visualization enbles us to focus better between problems and objectives of modeling and rendering:

- **modeling**: discovering new significant means of extracting from the dataset geometric coloured objects that convey meaningful information.
- **rendering**: supplying all the tools for transforming the previous objects into images, such tools should be general, efficient and possibly not too closely linked with a particular visualization technique.

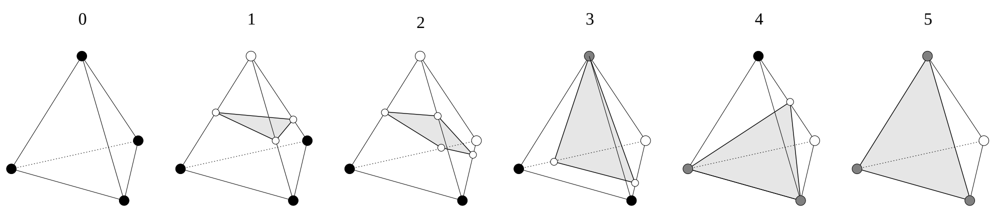


Figure 2.3: Marching Tetrahedra: vertex configurations and corresponding isosurface patches.

Our choice of simplicial complexes as a unifying kernel for scientific visualization reflects this belief; in particular we think that choosing a particular primitive, the tetrahedron, for representing visual properties in a portion of space and defining how the color and opacity are rendered into images allows us to concentrate on the modeling problems of visualization. From this point of view it will be straightforward to propose the unified view of Isosurfacing and DVR presented in Chapter 4 as a unique visualization *modeling* strategy.

2.2 Isosurface Fitting

In this section we discuss the isosurface extraction issue for tetrahedral complexes. Some parts of the following description, like the three-value vertex classification with correct handling of degenerate situations and the optimization techniques described in section 2.2.1, are original.

Given a volume dataset described by a tetrahedral complex Σ with a set of scalar values $f(v_i)$ associated with the vertices v_i of the complex, and a threshold value δ, the isosurface passing through the points of the volume dataset having a value of δ can be reconstructed by using a *per cell* approach similar to the Marching Cubes algorithm [69]. It is therefore usually called Marching Tetrahedra (MT). Some details of the algorithm that we are now going to describe will be used later in Chapter 4 when we will show how to integrate Isosurface and DVR.

The main idea behind the algorithm is to traverse all given cells and to extract for each cell σ crossed by the isosurface (active cell) the isosurface patch passing through σ. Each vertex v of the dataset is classified as black, grey or white if the value associated with v is, respectively, less, equal or greater than the given threshold δ. Such classification of a cell vertices can generate $3^4 = 81$ different combinatorial configurations. By exploiting symmetries, the latter can be fitted into the six main classes shown in Figure 2.3.

Once the class for each tetrahedron has been identified, the position of the isosurface vertices are calculated by linear interpolation on the tetrahedron edges. The choice of which edges must be used and how to connect the vertices found is made using a table accessed through the tetrahedron class. A surface normal should also be calculated for each isosurface vertex, to improve the smoothness of the resulting isosurface.

Our three–value classification and the consequent six classes of figure 2.3 are necessary to correctly handle the degenerate situation of vertices with field value w_i equal to the isosurface threshold δ. The usual two-value classification generates only the first three classes of figure 2.3 (the ones with no grey vertices): fitting the situation of class 3, 4 or 5 into class 1 or 2 can generate a null triangle that should be purged in a postprocessing phase.

Particular attention must be payed to the management of class 5, since, when an isosurface facet coincides with a face of the tetrahedron, we could generate the same facet twice. To get round this problem, we modify the 81-entry table of the MT in order to generate isosurface facets of class 5 cells only if the vertex of the tetrahedron not on the isosurface is greater that the threshold value. This solution represents a compromise and its shortcomings reflect the fact that it is not possible to solve all the ambiguities looking only at the vertices configurations. In fact, this trick correctly solves the problem for all cases except two: isosurfaces passing exactly through the boundary of the domain of the dataset, and the non-2-manifold situation of isosurfaces passing exactly through a face that is an area of local minimum/maximum of the field. A more correct solution can be found only by exploiting the topology of the cell complex marking the adjacent class 5 cell in order to prevent a redundant generation of this facet.

Moreover, it should be noted that an isosurface, defined as the set of points where the family of linear interpolating functions has a given value, is not necessarily a 2-manifold surface with boundary. For example, if three cell vertices with the same value define an area of local maximum δ (or minimum), then the isosurface with threshold δ is the one shown in Figure 2.4: the edges of the central triangular face are not a 2-manifold set.

2.2.1 Purging non–active cells

A drawback of the isosurface visualization is that only a small subset of the dataset, and therefore of the information contained in it, is represented in the final image; in most cases the ability to modify interactively the threshold value while viewing the resulting isosurface permits us to infer the global structure of the scalar field. For this reason efficiency issues are very important when visualizing isosurfaces. The isosurface extraction can be made more efficient by introducing some optimizations; this subsection describe

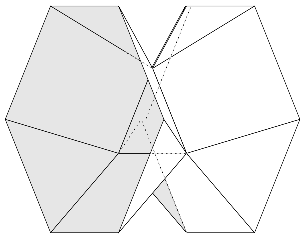


Figure 2.4: Non-manifold situation in isosurface fitting: the central triangular face is a local minimum area passing exactly through a face of the dataset.

our contribution to the optimal solution to the problem of the identification of the set of cells crossed by the isosurface; infact the determination of this set usually needs the traversal of the whole dataset, even if the searched isosurface crosses few cells. The time spent in the exhaustive search of the crossed cells is, in many cases, the dominant part of the whole isosurface extraction process. Speedup techniques were proposed in order to avoid the analysis of non-active cells, and can be classified according to two main criteria:

- *search modality*: active cells can be searched either in the geometric space (*geometric approach*) or, alternatively in interval space (*interval approach*), defined as the set of the min-max data value intervals of each cell. In the former case a data structure is built over the domain of the dataset in order to find the parts of the domains containing cells traversed by the isosurface, in the latter case the intervals containing the given threshold are searched for and then the corresponding active cells recovered. The selection of the search modality is often subject to the geometric structure of the underlying dataset: the geometric approach is generally well suited for regular datasets, the interval approach is independent of the geometric strucute of the dataset, though it generally implies higher costs in terms of memory requirements.

- *locality exploitation*: if an isosurface crosses a given cell it certainly crosses also some of the adjacent cells. This coherency information can be exploited both to find efficiently the cells intersected by the isosurface and to reduce redundancy in geometric computations (isosurface vertices are shared by adjacent cells, so their position can be calculated just once). This approach is particularly well suited to the case of regular or curvilinear dataset where the adjacency information is implicitly maintained.

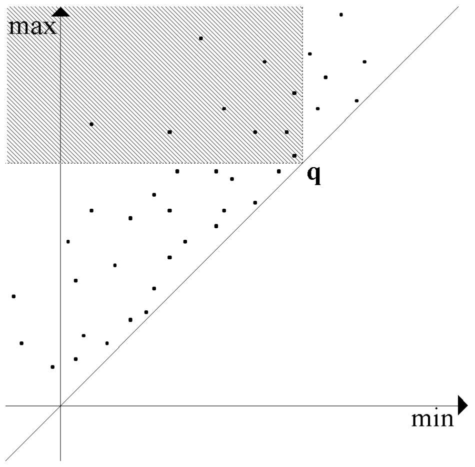


Figure 2.5: The span space. Each interval $I = [a, b]$ is represented as a point of coordinates (a, b). To detect the intervals that contain the query value q we have to find the points which lie to the left of the line $min = q$ and above the line $max = q$.

Before presenting our solution we review some of the results in the localization of the active cells. Wilhelms and Van Gelder [111] use a branch-on-need *octree* to purge sub-volumes while fitting isosurfaces, based on the range interval spanned by each sub-volume. This method achieves a worst case time efficiency $O(k + k \ \log(n/k))$ (where n is the total number of cells, and k is the number of active cells) [68], with small overhead (the octree increases storage occupancy only for about 16%).

Gallagher [47] proposes a method based on a subdivision of the range domain into buckets, and on a classification of intervals based on the buckets they intersect. The tradeoff between efficiency and memory requirements is highly dependent on the resolution of the bucketing structure. Giles and Haimes [50] report an approach in which two sorted lists of intervals are constructed in a pre-processing phase by sorting the cells according to their minimum and maximum values, respectively.

In a recent paper, Shen and Johnson [90] try to overcome some limitations of [47], and [50], however, a worst-case computational complexity of $O(n)$ was estimated for all three methods outlined above [68].
Livnat *et al.* [68] introduce the *span space* (see Figure 2.5), which is a two-dimensional space where each point corresponds to an interval in the range domain. The span space is very useful for understanding range-based methods geometrically, therefore we will refer to this representation also in the next sections. A *kd*-tree is used to locate the active intervals in this space,

achieving an $O(\sqrt{n}+k)$ time complexity in the worst case. In a more recent paper, Shen *et al.* [68] propose the use of a uniform grid to locate the active intervals in the span space. Such an approach is suitable for parallel implementation.

Approaches that exploit the locality rely essentially on two requirements: the ability to find an active cell *(seed)* for each connected component of the isosurface and the ability to propagate the surface by traversing the mesh from cell to cell through adjacencies [95]. Adjacencies are implicit in structured datasets, while they need to be stored explicitly in unstructured datasets. Storing adjacencies explicitly roughly doubles the memory requirement of the dataset, hence making the overhead of surface-based methods in the unstructured case either comparable to, or even higher than the overhead of range-based methods. Moreover, further auxiliary structures are needed in order to find seeds.

Itoh et al. [56] base the search for seeds on a graph, whose nodes are the cells holding local minimum or maximum data values: therefore, an arc of the graph spans an interval in the range domain. Each arc supports a list of cells connecting its two end nodes. Given an isovalue, the graph is searched for an active arc, and the cells connected to this arc are sequentially scanned until a seed is found. A propagation method is activated on this seed. Since the isosurface can be made of many connected components, seed search must be repeated until all active arcs have been visited. This can take $O(n)$ time in the worst case [68].

A more efficient method of finding seed sets is proposed by Bajaj et al. [7, 103]; they show the relation between the seed sets selection and the contour tree problem, a tree that captures the contour of the topology of the function represented by the mesh; this structure has been studied before in image processing and GIS research. From the contour tree it is possible to find the minimum size seed set in polynomial time (reduciing the problem to a min-cost flow in a DAG), or find a good approximation of this optimum in $O(n^2)$. The seed set can be encoded in a range-based search structure, in order to locate efficiently active seeds for a given isovalue: optimal time efficiency can be achieved by using an interval tree. The seed set is very small on average, hence causing a small overhead, but it can be as big as $O(n)$ in the worst case (e.g., if the underlying field has a sinusoidal shape or it is simply affected by noise).

2.2.2 Selecting Cells by Using Interval Trees

The technique we have proposed in [20, 22] for active cell selection is in the class of range-based methods, and therefore it can be used both for regular and unstructured datasets. Let Σ be the input mesh. Each cell $\sigma_j \in \Sigma$ is associated with an interval I_j, whose extremes a_j and b_j are the minimum and maximum field values at the vertices of σ_j, respectively. Since σ_j is active for an isovalue q if and only if its corresponding interval I_j contains q, the following general query problem is resolved:

> "given a set $\mathcal{I} = \{I_1, \ldots, I_m\}$ of intervals of the form $[a_i, b_i]$, with $a_i \leq b_i$ on the real line, and a query value q, report all intervals of $\mathcal{I}$ that contain q".

The problem is effectively visualized using the *span space* introduced by Livnat et al. [68] (see Figure 2.5): each interval $I_i = [a_i, b_i]$ is represented as a point in a 2D Cartesian space using the extremes a_i and b_i as the x and y coordinates of the point, respectively. From a geometrical point of view, the problem of reporting all intervals that contain the query value q is reduced to collecting the points in the span space lying in the intersection of the two half–spaces $min \leq q$ and $max \geq q$.

An optimally efficient solution for the query problem above can be obtained by organising the intervals of $\mathcal{I}$ into an *interval tree*, a data structure originally proposed by Edelsbrunner [38] (see also [83]), which is reviewed in the following. For each $i = 1, \ldots, m$, let us consider the sorted sequence of values $X = (x_1, \ldots, x_h)$ corresponding to distinct extremes of intervals (i.e., each extreme a_i, b_i is equal to some x_j). The interval tree for $\mathcal{I}$ consists of a balanced binary search tree $\mathcal{T}$ whose nodes correspond to values of X, plus a structure of lists of intervals appended to non-leaf nodes of $\mathcal{T}$. The interval tree is defined recursively as follows. The root of $\mathcal{T}$ has a discriminant $\delta_r = x_r = x_{\lceil \frac{h}{2} \rceil}$, and $\mathcal{I}$ is partitioned into three subsets as follows:

- $\mathcal{I}_l = \{I_i \in \mathcal{I} \mid b_i < \delta_r\}$;
- $\mathcal{I}_r = \{I_i \in \mathcal{I} \mid a_i > \delta_r\}$;
- $\mathcal{I}_{\delta_r} = \{I_i \in \mathcal{I} \mid a_i \leq \delta_r \leq b_i\}$.

The intervals of $\mathcal{I}_{\delta_r}$ are arranged into two sorted lists $\mathcal{AL}$ and $\mathcal{DR}$ as follows:

- $\mathcal{AL}$ contains all elements of $\mathcal{I}_{\delta_r}$ sorted in $\mathcal{A}$scending order according to their $\mathcal{L}$eft extremes a_i;
- $\mathcal{DR}$ contains all elements of $\mathcal{I}_{\delta_r}$ sorted in $\mathcal{D}$escending order according to their $\mathcal{R}$ight extremes b_i.

The left and the right subtrees are defined recursively by considering interval sets $\mathcal{I}_l$ and $\mathcal{I}_r$, and extreme sets $(x_1, \ldots, x_{\lceil \frac{h}{2} \rceil - 1})$ and $(x_{\lceil \frac{h}{2} \rceil + 1}, \ldots, x_h)$, respectively. The interval tree can be constructed in $O(m \log m)$ time by a direct implementation of its recursive definition. The resulting structure is a

binary balanced tree with h nodes, and a height of $\lceil \log h \rceil$, plus a collection of lists of type $\mathcal{AL}$ and $\mathcal{DR}$, each attached to a node of the tree, for a total of $2m$ list elements.
Given a query value q, tree $\mathcal{T}$ is visited recursively starting at its root:

- if $q < \delta_r$ then list $\mathcal{AL}$ is scanned until an interval I_i is found such that $a_i > q$; all scanned intervals are reported; the left subtree is visited recursively;
- if $q > \delta_r$ then list $\mathcal{DR}$ is scanned until an interval I_i is found such that $b_i < q$; all scanned intervals are reported; the right subtree is visited recursively;
- if $q = \delta_r$ then the whole list $\mathcal{AL}$ is reported.

The geometric interpretation of the search in the span space is also given in Figure 2.6. The regions containing the active intervals are those to the left of and above the dotted lines from q. Each sector of space (node of the tree) which contains the horizontal dotted line (i.e., such that $\delta_r \geq q$) is visited top-down (scanning the $\mathcal{AL}$ list) until such a line is reached; each sector containing the vertical dotted line is visited left to right (scanning the $\mathcal{DR}$ list) until such a line is reached. Therefore, $\lceil \log h \rceil$ nodes of the tree are visited, and for each node only the intervals reported in output, plus one, are visited. Hence, if k is the output size, then the computational complexity of the search is $O(k + \log h)$. Since $\log h$ is the minimum number of bits needed to discriminate between two different extreme values, no query technique could have a computational complexity smaller than $\Omega(\log h)$, hence the computational complexity of querying with the interval tree is output-sensitive optimal. It is interesting to note that the time complexity is independent of the total number m of intervals, i.e., of the input size: indeed it only depends on the output size, and on the number of distinct extremes.

It is worth mentioning that, although our proposal is the first application of the interval trees to speedup isosurface extraction, other authors have used it to address related problems: Laszlo [63] considers the extraction of wireframes from a grid of generic polyhedra, by using an interval tree, where each interval corresponds to an edge of the input grid; van Kreveld [102] extracts isolines from triangulated terrain data, by associating each triangle with the interval of altitudes it spans.

Moreover is should be noted the existence of another data structure, the segment tree [83], that achieves the same worst case complexity $O(k + \log h)$ for queries. The segment tree has a $O(m \log h)$ asymptotic space complexity while the interval tree is $O(m)$, however for small values of h (i.e. for datasets

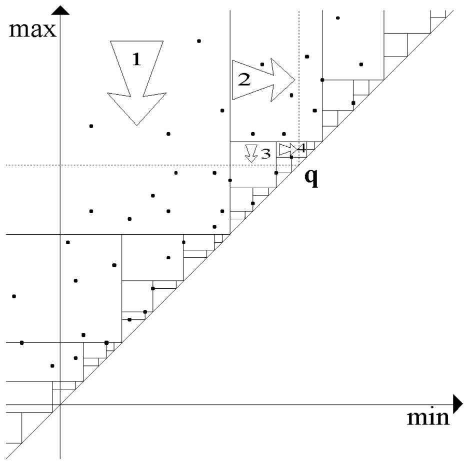


Figure 2.6: A graphical representation of the interval tree for the example of Figure 2.5. By definition, the intervals lying on subdivision lines belong to the upper level of the tree. The tree search for a value q: sectors with $\delta_r < q$ (intersected by the horizontal line $max = q$) are visited top-down; sectors with $\delta_r > q$ (intersected by the vertical line $min = q$) are visited left to right.

where the field assumes a small number of different values) the segment tree shows an effective lower space complexity.

Recently other authors [17] have extended our idea, proposing the creation of the interval tree in a preprocessing phase, and its storage on secondary memory; this structure is then accessed with an I/O optimal algorithm without loading either the interval tree or the whole dataset in main memory, thus making the interval tree suitable also for very large datasets.

Some further optimizations can be introduced to improve the efficiency of isosurface extraction: when calculating the isosurface normals and to reduce redundant computations. Gradients of the scalar field at the vertices of the mesh can be computed in a preprocessing step, as the weighted average of normalized gradients on all tetrahedra sharing the vertex v, where the weight for the contribution of a tetrahedron t is given by the solid angle[1] of t at v. The normal at a vertex of the isosurface I is computed during isosurface extraction by the linear interpolation of gradients at the endpoints of the cell edge where I lies. The redundant computation of both coordinates and normals on common vertices of the isosurface may be avoided either by exploiting the adjacency information to recover already computed values in

[1]The solid angle of a triedral angle is $A = \alpha + \beta + \gamma - \pi$ where α, β, γ are the diedral angles between the tetrahedron facets.

Dataset		**Interval Tree**			
Name	nodes (n)	intervals (m)	depth	nodes (h)	creation time
Fighter	13,832	70,125	15	12,538	1.50
Bluntfin	40,960	224,874	16	28,022	5.34

Table 2.1: Data on the interval trees for the test datasets (times are CPU seconds).

near cells or, when the topology of the complex is not available, through a *hash indexing* [111, 22].

Experimental Results Our proposals, based on the interval tree data structure, were tested on a number of different datasets. We report here the results for two datasets:

Fighter, an dataset built on the Langley Fighter, reporting a wind tunnel model simulation performed at NASA Langley Research Center. The dataset was represented by adopting a 3D Delaunay triangulation;

Bluntfin, originally defined as a curvilinear dataset, it has been represented here by adopting a tetrahedral decomposition; Bluntfin represents the air flow over a flat plate with a blunt fin rising from the plate. Courtesy of NASA Ames Research Center;

The results refer to the use of the interval tree data structure, the hash indexing technique and the pre-computation of the gradients of the field in the vertices of the cells in the case of unstructured datasets, *IT On*, compared with a standard Marching Tetrahedra implementation, *IT Off* (see Table 2.2). Numeric results have been obtained on an SGI Indigo2 (200MHz R4400 CPU, 16K instruction and 16K data caches, 1MB secondary cache, 32MB RAM, IRIX 5.3).

Table 2.1 reports numeric values on the datasets used and on the associated interval trees: the number m of intervals, which is equal to the number of tetrahedral cells, the interval tree *depth*, the number h of nodes of the interval tree and the *time* (in seconds) required to build the interval tree data structures.

2.2.3 Rendering Transparent Isosurfaces

After an analysis of the *modeling* aspects of the isosurface technique of visualization we can now deal with a *rendering* problem that is commonly

Threshold	Facets	IT On	IT Off
NASA Fighter - 70125 Tetrahedral cells			
2.6534	386	3	142
2.4420	1754	13	154
2.2007	5545	41	185
2.0221	9735	78	220
0.5599	20560	164	312
Bluntfin - 224874 Tetrahedral cells			
4.8722	444	3	255
0.3409	1184	7	238
4.2741	2100	12	263
3.2071	5171	33	279
2.1305	10384	72	304
0.5371	20663	154	357

Table 2.2: Isosurface extraction times on tetrahedralized datasets, in milliseconds.

ignored: the correct and efficent rendering of many transparent isosurfaces simultaneously. When rendering more than one single isosurface, to obtain the best comprehension of inner structures of the dataset it is necessary to use transparency. An easy/low quality solution is to render them using a common graphics library like OpenGL and adopting the *screen door* transparency, [41]; to obtain better quality without using high-quality and slow algorithms like scanline or ray-tracing techniques, it is necessary to draw and blend directly the isosurface facets in depth order onto the screen. This problem is strictly related to the problem of sorting the tetrahedral complex itself addressed in Chapter 3: each isosurface facet is entirely contained inside a tetrahedron, so the occlusion relation between two isosurface facets in different cells agrees with the relation between the tetrahedra themselves; for this reason, if a depth ordering for the whole complex is available, it is sufficient to sort the isosurface facets separately inside each tetrahedron and collect these orderings following the tetrahedra depth order for the complex. This in-tetrahedron ordering can be easily achieved in linear time in the following way: we assume we maintain inside each tetrahedron t the isosurface facets contained in it sorted according to increasing values of their threshold, and we assume that the facet normals agree with the field gradient. For each facet f_i, we check if the viewpoint lies on the same side of the normal of f_i, marking f_i with a '+' or a '-'; all the isosurface facets in t are parallel, so that to obtain a correct back to front ordering it is sufficient to draw first all the facets with a '+' sign, in the field value order and then the ones with '-' in reversed field value order.

2.3 Direct Volume Rendering

Direct volume rendering techniques are the visualization approach that aims at producing a projected image directly from the volume data, without intermediate representation as isosurfaces. These techniques rely on the mapping of the field value into the color/opacity of the 3D domain itself; they require some model of how the volume generates, reflects, scatters or occlude light. In the next Sections we will describe the optical models used in DVR and we review direct volume rendering solutions, with emphasis on the ones working on simplicial decompositions; the presentation will follow the classical image-order vs object-order classification:

- *ray tracing* methods process the dataset in *image-order*, and accumulate color and opacities interpolated while tracing each ray;
- *scan line* methods can be considered an evolution of ray tracing methods, because they adopt an *image-order* but exploit the coherency between rays to reduce visualization times;
- *projective* methods are based on an *object-order* visit of the dataset: cell are processed in depth order, and each cell is projected and composed onto the current image in a single pass; they therefore exploit coherence at the object level.

Figure 7 on page 134 gives an high level view of these approaches.

2.3.1 Volume shading models

Many models for simulating propagation, scattering and shadowing through semi-transparent media have been proposed in literature. The first models aimed to simulate natural phenomena, like dust [15] or clouds [58]. Later on these models were extended or modified [37, 88, 65] to fulfill scientific visualization needs. This distinction still remains, and it is therefore useful to clarify the two different objectives in modeling semi-transparent media:

- *photo-realism*: we want to model real semitransparent media, like clouds, fog, dust, to reproduce their aspect with the utmost visual fidelity;
- *comprehension*: in the visualization of a 3D structure, we want to exploit the hints that shading can add to obtain easier comprehension of the information we are visualizing.

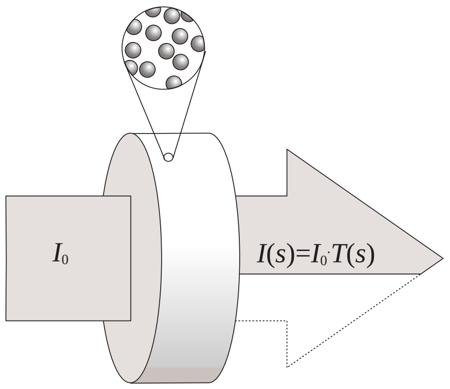


Figure 2.7: An infinitesimally thin cylindrical slab filled with spherical particles.

In the first case, an accurate model of the medium should take into account phenomena like self-shadowing and second and higher order scattering of light inside the medium; while the realism of the images computed in this way is often stunning, the computational times involved are so large that interactivity, a requirement in Visualization, is prevented.

In the following, we sketch the basis of the most common model in scientific visualization, known as the *density emitter* model, following the presentation given in [71].

State of the art of volume density optical models

A volume dataset lighting model can be approximated by a number of infinitely small spherical particles of radius r and projected area $A = \pi r^2$ (Figure 2.7). Consider a small cylindrical slab with a base of area E, a thickness of δs and a volume of $E\delta s$. This slab is filled with particles with a given density ρ (number of particles per unit volumes). The slab therefore contains $N = \rho\, E\, \delta s$ particles. We now consider three different classes of behaviour of the particles: perfectly black particles, perfectly transparent particles glowing, and glowing and absorbing particles.

Absorbing Medium In this case we assume that particles are perfectly black, i.e. they absorb all the light that they occlude without any scattering (reflective) effects. If the thickness of the slab δs is small enough, we can assume that there is no overlap between particles along the height of the cylinder, so the total occlusion area is $NA = \rho\, E\, \delta s\, A$ and the fraction of light blocked by the particles is $\rho A \delta s$. With $\tau(s) = \rho(s) A$ we denote the extinction coefficient that defines the rate of light that is occluded at distance

s. We can write the differential expression showing the fraction of absorbed light when the slab thickness reaches zero as:

$$\frac{dI}{ds} = -\tau(s)I(s)$$

where s is a length parameter and $I(s)$ is the light intensity at distance s. The negative sign in this differential equation is because we are calculating the light that is subtracted when traversing the volume. Remember that the extinction coefficient (also referred to in literature with a misleading term as *opacity*) range from zero (no particles) to infinity (opaque medium). The solution of the above equation gives us the light intensity $I(s)$ that reaches position s along a ray traversing a volume.

$$I(s) = I_0 e^{-\int_0^s \tau(t)dt}$$

where I_0 is the starting light intensity when the ray enter in the volume. The following quantity is the cumulative transparency of the medium between 0 and s, that is the fraction of light that is absorbed between 0 and s:

$$T(s) = e^{\int_0^s \tau(t)dt} \tag{2.1}$$

If the extinction coefficient τ varies linearly, as in the case of the linear interpolation of the visual parameters inside a tetrahedron, $\tau(s) = \tau_0 + \tau s$, the previous equation becomes:

$$T(s) = e^{-\frac{s(2\,\tau_0 + \tau\, s)}{2}}$$

Emitting Medium Proceeding in a similar way, we assume that the particles are perfectly transparent (no absorption) and glowing diffusely (isotropically), with an intensity $g(s) = C(s)\tau(s)$ proportional to their density. The equation that describes the light intensity emitted by this type of media, in the case where the glowing factor is a linear function $g(s) = g_0 + g\,s$, is:

$$I(s) = I_0 + \int_0^s g(t)dt = I_0 + g_0\,s + \frac{g\,s^2}{2}$$

Absorbing and emitting medium (density emitter model) In this case the light emitted by the volume is partially occluded by the opacity of the volume itself. The differential equation becomes:

$$\frac{dI}{ds} = g(s) - \tau(s)I(s)$$

If we consider the general case where the glowing and absorbing function, $\tau(s)$ and $g(s)$ are two linear equations the solution of the above differential equation is:

$$I(s) = I_0 T(s) + \int_0^s g(t) T'(t) dt$$

where

$$T'(k) = exp\left(-\int_k^s \tau(x) dx\right)$$

but there is no simple closed form for the equation above, even if simple numerical techniques for approximating it can be devised [113].

A simplifying assumption is that the glowing factor of the volume $g(s) = G_0 \tau(s)$ depends on the density of the volume. While keeping these two values separate permits a larger flexibility, this assumption allows us to write the close form of the integral:

$$I(s) = I_0 e^{\int_0^s \tau(t)dt} + G_0 \left(1 - e^{\int_0^s \tau(t)dt}\right) \tag{2.2}$$

recalling the definition of transparency in equation 2.1, the equation 2.2 can be written as:

$$I(s) = T(s) I_0 + (1 - T(s)) G_0 \tag{2.3}$$

It is interesting to note that this result can be interpreted as the classical compositing formula [82] of a color G_0 atop a background I_0 with an opacity $\alpha = 1 - T(s)$.

2.3.2 Ray Tracing

The first techniques developed for rendering volumetric data directly were based on the ray tracing approach [58, 65]. For each pixel of the image, a ray is cast and intersected with the cells of the volume data. The transfer function, which transforms the data values into opacities and colors, is then discretely sampled and integrated along the ray. The result of this integration determines the color of the pixel. Such techniques were initially described for regular volume dataset only (regular volumes can be ray-traced in a rather efficient way by exploiting the regular structure of the data).

Garrity presented a technique to perform ray-tracing on irregular datasets exploiting the topological relationships between cells to perform efficiently the color and opacity integration along the ray [48]. Although the technique described works on volumetric datasets composed of convex cells, the author uses only tetrahedral volume datasets to avoid handling hexahedral datasets degeneracies explicitely.

This approach works in two steps. In a preprocessing phase, the cell faces which are on the boundary of the dataset are detected and inserted in a bucketing structure (based on a regular space subdivision). In the second phase, for each ray, the nearest intersected face is searched among the boundary ones, and cell tracing starts from the associated cell. Then, jumping from cell to cell by exploiting the connectivity of the dataset, all the cells intersected by the ray are detected. When the ray exits the dataset, the list of boundary faces is browsed again to see if the ray enters the dataset again (e.g. in the case of non–convex or multi–component datasets). Particular attention must be payed to degenerate cases (ray passing through faces, edges or vertices). The restriction to tetrahedral cells simplifies handling of degenerate cases.

Once the intersected tetrahedra are found, the integration of the transfer function along the ray spans contained inside each tetrahedron can be carried out as explained in Section 2.3.1.

2.3.3 Scan Line

The ray-tracing approach fails to exploit the fact that adjacent rays probably intersect the same cells, and its computational cost is therefore excessively high. Scanline algorithms try to exploit this coherence. The first scanline algorithm to render volume dataset described by arbitrary cell complexes was proposed by Giertsen [49]. The algorithm uses a *scan plane buffer*, a data structure associated with a plane (xz) perpendicular to the viewplane (xy) and passing through a line of the viewplane. Volume cells are progressively sliced by the scan plane; each slice is then triangulated in order to interpolate linearly the values inside each cell slice[2].
The volume cells are maintained y-ordered and the set of cells intersected by the scan plane is updated following the Y-sweeping of the plane. The *scan plane buffer* (*spb*) is a structure that maintains discretely (a two-dimensional array) the intersections between the cells and the scan plane. Cell slices are scan-converted in random order into z segments orthogonal to the viewplane, and the length and the opacity/color contribution of each segment are stored into the *spb* structure. Once all cell slices have been scan–converted, the colors of the pixels on the current scan line are calculated by traversing in the z direction the *spb* and accumulating the opacity/color contributions contained in them.

Another algorithm based on the scanline approach was the *Lazy Sweep Ray Casting* (LSRC) proposed in [94, 93]. Its main contribution is to avoid

[2]It must be noted that this kind of interpolation is not rotational invariant and that therefore it creates aliasing effects when rotating the volume. A more correct solution, as suggested by the same Giertsen [49], is to decompose each cell into tetrahedral elements.

the use of a discrete scanplane, to prevent the possible aliasing effects that can derive from datasets having high variations in cell size (differences of the order of 1:100,000 can occur). The LSRC algorithm works on two phases: a space sweep with a sweep plane like the Giertsen one, orthogonal to the viewing (XY) plane, and a sweep of that plane with a sweep line parallel to the Z axis.
The space sweep proceeds (like any algorithm based on the sweep paradigm [83]) by maintaining a sweep structure which monitors discrete sweep events. In our case events occur when the sweep plane hits vertices of the mesh. The sweep plane structure maintains the subdivision S resulting from the intersection between the plane and the mesh.
For each scanline, we process S sweeping it with a line. The sweep line status this time is a Z-ordered set of intervals (the intersection between the sweep line and S). For each pixel coordinate x, we can easily calculate the resulting pixel color/opacity by compositing the contribution of each segment in the sweepline status structure.

Another scanline algorithm that exploits a spatial hierarchical organization of the dataset has been described by Wilhelms et al. in [110]. The main difference with this approach (that like the previous algorithms uses both a scan plane orthogonal to the view plane and a scan line lying on that plane) is that it renders semi-transparents regions of space between polygons as well as opaque polygonal surfaces immersed in the dataset. The method builds a k-d tree over the polygons (either cell faces or immersed object faces) to improve efficiency. The hierarchy is also used to render approximate images of the dataset. The method has been parallelized on a shared memory MIMD machine. The problem of a rotationally invariant field interpolation inside each cell still holds, unless a tetrahedral decomposition is adopted.

In [105] Westermann and Ertl present a different approach to the scanline paradigm in order to exploit common graphics hardware, during the sorting of cells crossing the scanplane. The key idea lies in a two-pass approach: first the *vsbuffer*, a discrete representation of the scanplane, is filled with the intersection of the cells that it crosses, then this buffer is traversed in front to back order to perform the volume integration. The *vsbuffer* approach can be seen an extension of the original *scan plane buffer* proposed by Giertsen [49]; in this case the discrete scan plane is filled by looking the scanplane from the top and rendering all the cells crossing it twice: in the first rendering pass objects above the screenplane are clipped and their backfacets drawn, in the second one objects below the scan plane are clipped and front facets drawn. The RGB color of each cell is chosen to unambiguously code the cell index. The result of the two rendering are compared and if two corresponding pixels share the same color/index then that pixel is covered by that cell. At the end of this process each pixel of the *vsbuffer* contains the index of the cell

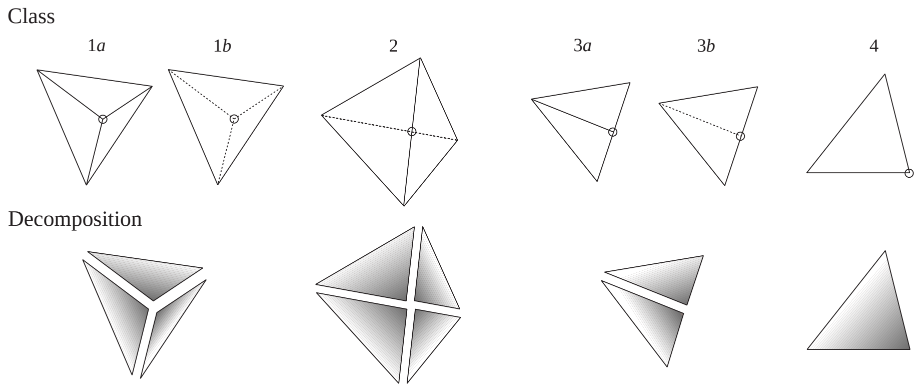


Figure 2.8: Classification and decomposition into triangles of the projection of a tetrahedron.

(coded with color) that covers it.

2.3.4 Tetrahedra Projection

The problem of rendering directly a volume dataset can be solved using a object order approach: the visual contribution of each tetrahedron is computed and accumulated onto the screen buffer. To obtain a correct result the compositing must be done in depth order (back to front or viceversa) so a correct ordering must be calculated. This particular problem of depth sorting a tetrahedral complex will be dealt with greater detail in the next Chapter.

Once the complex has been sorted, various techniques are available to calculate and compose the contribution of each tetrahedron to the intermediate image.
A first approach is to scan-convert each tetrahedral cell, adopting a pure software threedimensional scan-conversion process [72]. For each pixel, the contribution in color and opacity can be directly calculated during the 3D sampling, according to the shading model chosen.

A faster approach renders tetrahedral cells by approximating each cell with semi-transparent triangles, and compositing such triangles using standard graphics libraries/hardware [92]. The main idea is to classify the shape resulting from the projection of a tetrahedron (see Figure 2.8) in a limited number of classes which can be easily decomposed into triangles. Then, the correct opacity/color of the thickest point of the projected tetrahedron (indicated with a small circle in Figure 2.8) is computed, and the cells contribution to the current frame is obtained by gouraud–interpolating the pro-

jected shape.
The same idea of rendering the silhouette of the projected cell as a bunch of semi–transparent triangles was applied by Wilhelms and Van Gelder to rectilinear hexahedral (*coherent projection*) cells [108] and to slightly more generic convex hexahedral cell (*incoherent projection*) in [101].

The classification of the projected silhouette and the identification of the thickest point can be done in, at least, two different ways, according to the sorting algorithm used:

- *Cell classification after a topological sort.* In this case the transformation of the adjacency graph into a direct one gives, for each tetrahedron, the number of faces which are oriented toward the observer. Starting from this number, together with the number of facets orthogonal to the viewplane, we can automatically deduce the class of projection. The equations of the planes passing through the facets can also be used to calculate the coordinates of the thickest point.

- *Cell classification after a numerical or an NNS sort.* In this case, each tetrahedron must be classified independently. The easiest technique is to test the clockwise ordering of vertices of each tetrahedron facets according to the viewpoint. This can be done by a simple cross product between two edges of each face.

Aliasing One of the drawbacks of using graphics hardware to render the projected silhouette of tetrahedral cells with transparent triangles is the aliasing introduced, as pointed out by Wilhelms and Van Gelder [101] and Steiner et al. [97]. There are two main reasons for these errors in rendering:

- limited numerical precision of rendering subsystems to compute the rgbα compositing,

- linear interpolation of the opacities, while an exponential interpolation should be used to approximate opacity contribution correctly.

To reduce the latter problem, the *multipass blending* approach [101] renders and composes each triangle three times to give a quadratic interpolation. With the first two passes the quadratic interpolation of opacity is obtained by a double linear interpolation and composition of opacity; the third pass applies the color contribution. In a more efficient solution [97], color and opacity are correctly interpolated between vertices using hardware assisted texture mapping (the texture map is, in this case, a two dimensional table with the values of the correct exponential opacity).

Approximated Projection Technique

An example of an approximate rendering technique is the *Incremental Slicing* method proposed by Yagel et [116]. Given the current view direction, the volume is transformed into viewing space and then the 2D polygonal subdivisions resulting from the slicing of that volume with a set of planes parallel to the view plane are calculated and stored. Such polygonal meshes are then rendered and composed in visibility order using graphics hardware. The number of slices is chosen adaptively to reduce the number of cells not intersected by any slicing plane.

Splatting techniques

A different image/order approach for DVR, based on samples rather than on cells, is the splatting technique. This approach was developed by Westover in [106] to accelerate the rendering of regular datasets. The first step of the volume rendering process is to define the field value over the whole domain of the dataset, that is, using Westover terms, to reconstruct a continuous signal from discrete samples. Instead of using the common trilinear approach, Westover reconstructs the sampled signal by convolving the reconstruction kernel with the sampled data. The volume rendering process is regarded as a integration along the viewing direction: the contribution of each sample to the final image can be obtained by the integration of the kernel function used in the interpolation process. Choosing a rotational invariant kernel function it is possible to precompute the *footprint* of such an integral. In this way the contribution of each sample to the final image, called *splat*, can be composed onto the screen in the usual back-to-front way. This approach can be implemented very efficiently by using various approximations of the kernel function, usually a simple 2D elliptical Gaussian sampled and stored in a table [107] or directly drawn with a small collection of Gouraud shaded polygons [64].

The direct extension of Westover's splatting algorithm to tetrahedral complexes or curvilinar grids is difficult because, in many cases, it is almost impossible to select a simple splatting kernel for each grid point. In [70] Mao propose an algorithm to resample the curvilinear grid with a set of new points so that it is possible to reconstruct the original signal using common ellipsoidal kernel function around these new points. Using these points the volume is then rendered with the usual splatting approach.

One of the main disadvantages of this class of algorithms is the low quality of rendering when the sample points are not uniformly distributed in the image space.

2.3.5 Conclusions

Summarizing the various key aspects of techniques presented above we can mention some pros and cons:

- **Ray Tracing**:
  Pros: High quality, general technique; it can integrate volume models with surface geometry, photorealistic effects like refractions or self shadowing, and it can use oversampling techniques to generate high quality images with no aliasing problems.
  Cons: Very slow; acceleration techniques work but they present low flexibility and rendering capabilities; it cannot be used interactively.

- **Scan Line**
  Pros: High quality, it can integrate volume models with standard surface geometry, much faster than ray tracing but less general.
  Cons: Still too slow, and not very flexible.

- **Projective**
  Pros: Fast, it can exploit graphics hardware, it shares the same flexibilty of the triangle-based graphics library.
  Cons: the rendering quality can rely on the graphics hardware capabilities.

Given the importance of the interactivity issue in visualization we have chosen to investigate some of the problems that the projective approaches still present. In the next chapter we will face the problem of depth sorting a tetrahedral complex, a step needed for projective algorithms to correctly compose tetrahedron contributions on the screen. Later in Chapter 4 we will discuss how to correctly mix isosurfaces and DVR through projective algorithms.

Recalling our distinction between modeling and rendering in Visualization, we can add that we can base our work on what we call *tetrahedral graphics*: our guess is that it could be possible face most of the rendering problems in tetrahedral visualization if we assume that we have a simple and robust primitive for the correct rendering and composition of a single tetrahedron on the screen. Given such a primitive, i.e. the thetrahedral analogous of the triangle based primitives present in most graphics library, the main problem of the user is to sort them in the correct order. Other problems, like the problem of correctly manage non linear transfer functions and integrating isosurfaces with DVR using a projective approace, become *modeling* problems rather than *rendering* ones, and within this approach, they will be faced in Chapter 4.

Chapter 3

Depth sorting a Tetrahedral Complex

In this Chapter we describe in detail the problem of depth sorting a tetrahedral complexe, reviewing known results and algorithms and their effective usability or robustness. Then we introduce a new technique for sorting complexes which belong to the class of projective complexes. The approach is based on the preliminary construction of the lifted *complex corresponding to the given one and on its representation as a power diagram. This approach exhibits a $O(n \log n)$ runtime complexity to sort a complex and require only linear storage.*

Projective algorithms render a tetrahedral mesh through direct projection and composition of tetrahedra [72, 92, 112]. They are generally based on a two-phases process: first, cells are sorted in depth; second, depth-ordered cells are projected onto the view plane and rgbα–composed on the frame buffer. To compose rgbα contributions correctly, cells have to be depth–ordered with respect to the given viewpoint.

In this Chapter we address the problem of depth sorting a given complex. In Section 3.1 we give some definitions that will be used throughout the chapter, then in Section 3.2 we review the current solutions to this problem on the basis of their theoretical and practical interest. In Section 3.3.3 we introduce a new technique for sorting that is based on a preprocessing step in which the original complex Γ is *lifted* in the convex polyhedron in $\mathbb{E}^{d+1}$ Γ^* whose projection in $\mathbb{E}^{d}$ gives Γ. Γ^* can be stored as a power diagram and efficently used to depth sort Γ along any view direction. This approach makes it possible also to establish if Γ is a projective complex and therefore can be depth ordered from any viewpoint.

3.1 Definitions

In this section we give some definitions regarding depth ordering, acyclicity, projective complexes and power diagrams that will be used in sections 3.3 to present our results.

Obstruction Relation The obstruction relation $\prec_p$ (usually called *in-front/behind* relation) for a pair of non self-intersecting cells γ_1, γ_2 with respect to a viewpoint p can be formally expressed as follows:

> $\gamma_1 \prec_p \gamma_2$ iff $\exists$ a ray r emanating from p and intersecting both γ_1 and γ_2, such that all points in $r \cap \gamma_1$ are closer to p than any point in $r \cap \gamma_2$.

A visibility order of a set of objects, with respect to a viewpoint p, is a sequence of such objects such that, if object A obstructs object B when seen from p, then A preceeds B in the sequence.

Acyclicity A cell complex Γ is called *acyclic* with respect to a given viewpoint p if and only if relation $\prec_p$ defines a partial order on the cells of Γ. In this case, it is possible to order the cells of Γ either front-to-back or back-to-front with respect to the viewpoint.

In the following a cell complex that is acyclic w.r.t. any viewpoint will be denoted, for sake of conciseness, as an *acyclic cell complex.*

3.1.1 Projective Complexes

Cell complexes in $\mathbb{E}^{\mathrm{d}}$ that can be obtained as the orthogonal projection of the lower part of the boundary of a convex polytope in $\mathbb{E}^{\mathrm{d}+1}$ are called *projective complexes.* These complexes are also known as *regular cell complexes* [39].

An important result relative to projective complexes is that they are acyclic:

Theorem 1. *(Edelsbrunner [39]) The in-front/behind relation defined for the faces of any projective complex and for any fixed viewpoint in $\mathbb{E}^{\mathrm{d}}$ is acyclic.*

The proof of this theorem is based on the construction, given a viewpoint p, of a function $\Phi : \sigma \in \Sigma \to \mathbb{R}$ that *agrees* with the occlusion relation, that is: $\Phi_p(\sigma)$ is such that $\Phi_p(\sigma) < \Phi_p(\tau)$ if $\tau \prec_p \sigma$. Clearly, if such a function exists for a given viewpoint then $\prec_p$ is acyclic indeed it is impossible to have a set of cells $\sigma 1, \ldots, \sigma_k$ that forms an occlusion cycle:

$$\sigma_1 \prec_p \cdots \prec_p \sigma_k \prec_p \sigma_1$$

because the corresponding function Φ should hold:

$$\Phi_p(\sigma_1) < \Phi_p(\sigma_2) < \cdots < \Phi_p(\sigma_k) < \Phi_p(\sigma_1)$$

The construction of the function Φ used by Edelsbrunner is rather complicated and not reported here for brevity. In Section 3.3 we will show a simpler way to build this function.

It should be noted that this theorem has been stated only in one direction: if a complex is projective then it is acyclic; we know nothing about the reverse of this theorem, if any acyiclic complex is also a projective one. In other words we do not know if the projective and acyclic complexes represent the same class: it is an open problem to show the existence of an acyclic complex in $\mathbb{E}^{\mathrm{d}}$. that is not obtainable as the orthogonal projection of a convex complex in $\mathbb{E}^{\mathrm{d}+1}$.

Delaunay simplicial complexes (see definition in Section 1.3) are acyclic with respect to any viewpoint [39]. This is an important property, because it assures that a volume dataset represented by a Delaunay complex can always be sorted and correctly visualized.

3.1.2 Power diagrams

Now, we recall some definitions and results regarding power diagrams; a complete introduction about power diagrams can be found in [6].

The power of a point p with respect to a sphere $s \subset \mathbb{E}^{\mathrm{d}}$ with center z and radius r, is defined as $pow(p, s) = d(p, z)^2 - r^2$. Thus $pow(p, s) < 0$ if p belongs to the ball bounded by s, $pow(p, s) = 0$ if p lies on the surface of s and is greater than zero otherwise; in the last case it easy to show that $pow(p, s)$ is equal to the squared distance of p from the touching point of a line tangent to s through p.

Let s and t be two spheres in $\mathbb{E}^{\mathrm{d}}$ with centers $z_s \neq z_t$ and radii r_s, r_t. The points x satisfying $pow(x, s) = pow(x, t)$ describe a hyperplane h that is perpendicular to the line joining z_s and z_t, known as the *chordale* of s and t, or $\mathrm{chor}(s, t)$ for short. A nice property of chordales is that if $s \cap t \neq \emptyset$ then $s \cap t \subset \mathrm{chor}(s, t)$; otherwise s and t are contained in the same open halfspace bounded by $\mathrm{chor}(s, t)$ if and only if s encloses (or it is enclosed in) t.

Let S denote a finite set of spheres in $\mathbb{E}^{\mathrm{d}}$, for $s \in S$ we call the set

$$\mathrm{cell}(s) = \{x \in \mathbb{E}^{\mathrm{d}} | \mathrm{pow(x, s) > pow(x, t) \forall t \in S - \{s\}}\}$$

as *power cell* of s and the collection of all $\mathrm{cell}(s)$, for $s \in S$, the *power diagram* of S, or $\mathrm{PD}(S)$ for short.

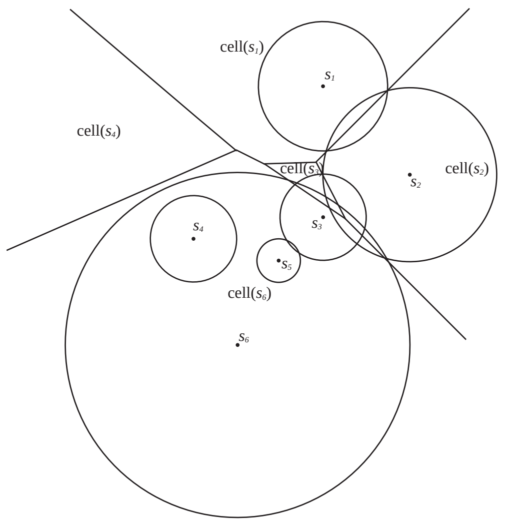


Figure 3.1: A power diagram in two dimensions.

Let S denote a finite set of spheres in $\mathbb{E}^{\mathrm{d}}$, by definition the power cell cell(s) is the intersection of $n-1$ halfspaces bounded by chordales, and therefore is a $d-$polyhedron with at most $n-1$ facets. This implies that PD(S) is a cell complex in $\mathbb{E}^{\mathrm{d}}$.

Figure 3.1 shows a PD of six circles in $\mathbb{E}^2$; note the possible occurrence of empty cells, like cell(s_5) and the fact that cells may be separated from their generating spheres, cell(s_4) is distinct from s_4.

Power Diagrams and Voronoi Diagrams The following relations exists between power diagrams, Voronoi diagrams and Delaunay triangulations [6]. Given a finite set of points P in $\mathbb{E}^{\mathrm{d}}$ the Voronoi diagrams of P correspond to the PD of the set of spheres S with centers in P and radii equal to zero. It should be noted that power diagrams are considered one of the generalization of Voronoi diagrams that have the strongest similarities to the original diagrams. These relations, together with our practical experience in using the numerical sorting for Delaunay triangulations, the dual structure w.r.t. Voronoi Diagrams, led us to the use of PD's as the basis for the design of a new depth sorting algorithm.

3.2 Related Work

The problem of depth sorting has received considerable attention in the literature: a lot of research deals with the two dimensional version of the problem, but here we survey some of the most important, sometimes mainly theoretical, results concerning the three dimensional depth-sorting problem.

It should be noted that in the most general case, a non convex, possibly unconnected arbitrary tetrahedral complex, the problem of calculating a depth ordering (when it exists), w.r.t. a given viewpoint, has a lower bound of $O(n \log n)$ [94], and none of the proposed algorithms still matches this bound.

In [78] Nurmi gave an algorithm for computing three-dimensional depth orders of a set of n line segment in $\mathbb{E}^3$ with a worst case complexity of $O(n \log n + i)$ where i is the number of intersections of the segments when the segments are projected on the xy plane. Note that i can be $O(n^2)$; Nurmi extended his algorithm to 3D polyhedra with a complexity of $O((n+i) \log n)$.

Chazelle et al. in [16] studied the problem of ordering lines in the space and noted that, in absence of cycles, a depth order can be obtained by a standard sorting algorithm, because any two lines can be always compared; in other words if there are no cyclic overlaps between lines, the $\prec$ relation is a total order. Unfortunately, this is not true for a set of segments in space, because not all the pairs can be compared, so this problem can be reduced to the extension of the $\prec$ relation from being a partial order (in the case of absence of cycles) to a total ordering.

This approach to the problem has been adopted by de Berg in [29]; he observes that the depth sorting problem is the same as computing a linear extension of the $\prec$ relation; he describes an algorithm that solves this problem in a general way for a given relation $\prec$ and its transitive closure $\prec_*$ on a set S of n objects; the complexity of the de Berg's algorithm relies on the efficiency of a data structure for storing a subset S' that can return a predecessor (successor) in S' of a query object $o \in S$, if it exists. The data structure should also support efficient deletion of objects from a subset S'.

The basic strategy of this algorithm is divide and conquer: choose a pivot object $o_p \in S$, partition the remaining objects into subsets $S_{\prec}(S_{\succ})$ of objects that must come before (after) o_p and recursively sort these sets. Note that not every pair of objects is comparable with o_p under $\prec_*$, therefore there is a third subset $S_{\approx}$ that contains such objects; this subset should be sorted recursively as well. Using a result obtained by Agarwal and Matousek in [2] for solving ray-shooting queries, de Berg is able to build efficiently the needed data structure for predecessor/successor queries. The resulting algorithm can compute a depth order for a set of segments, or decide that there is a cyclic overlap among the segments, with a worst case complexity of $O(n^{4/3+\epsilon})$. This result is then extended to triangles instead of segments with the same complexity.

An interesting related problem is that one of verifying the correctness of a given depth order; Chazelle et al. in [16] give a $O(n^{3/4+\epsilon})$ algorithm to solve this problem; a solution with the same complexity is also given by de Berg in [29]. Another contribution to lowering the complexity of this problem has been given in [1] by Agarwal et al.; their approach solves the linear extension problem, when the relation $\prec$ is not cyclic, and for triangles whose xy projections are *fat* enough in $O(n \log^6 n)$.

The above results assume that there is no preprocessing of the set of objects to be sorted. If this assumption is relaxed, different approaches can be chosen.

A popular technique for the preprocessing of a set of 3D objects is the Binary Space Partition tree (BSP), proposed by Fuchs et al. [45]. This structure, makes it possible to recover a depth order of a set of objects in time that is linear with the size of the BSP tree. The BSP tree is based on the recursive bi-partitioning of the space with planes, all the objects crossed by these partitioning planes are subdivided. This strategy has the drawback that the best bound of the size of a BSP tree of n objects in a three dimensional space is $O(n^2)$ [80]: that is the polygon subdivision along planes can generate a quadratic number of polygons. Moreover, while this bound is far from reality in common three dimensional scenes, where it is possible to choose space-partition planes such that they subdivide a very small number of objects, for our purposes this bound becomes much more realistic. In a generic tetrahedral mesh, any partitioning plane crosses (and therefore subdivides) a considerable fraction of the dataset creating a large number of new tetrahedra.

3.2.1 Depth Sorting Algorithms

Some practical algorithms for depth sorting a tetrahedral complex have been proposed and used on real problems; in this subsection we give a short overview of them and evaluate their complexities and the main drawbacks of each approach.

Topological sort The cells of an acyclic convex complex can be sorted by exploiting the face-adjacency relation between tetrahedra and face orientation. An algorithm, called Meshed Polyhedra Visibility Ordering (MPVO), based on this approach was proposed by Williams [112, 114].
In a preprocessing phase, the MPVO algorithm constructs the adjacency graph for the given mesh and calculates the plane equation coefficients for each face. At rendering time, the algorithm works in two steps. Given the

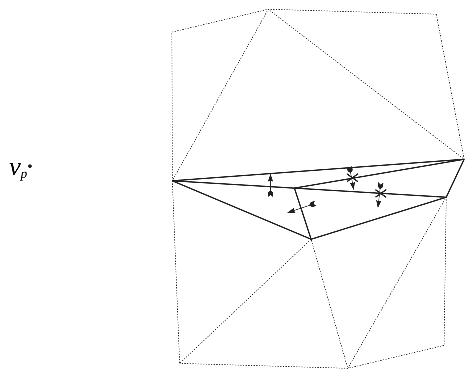


Figure 3.2: Topological sorting fails in the case of cycles created by wrong DAG orientation due to approximation error.

current viewpoint, first, exploiting the stored plane equation, it computes the occlusion relation for all pairs of adjacent cells, and converts the adjacency graph into a direct acyclic graph (DAG). Second, a total ordering of the cells is obtained in linear time by a topological visit of the DAG. If the topological sort is obtained by a depth–first visit of the DAG the presence of cycles can be detected and a partially correct mesh ordering can be calculated.

The author proposes an extension of this algorithm to manage non-convex meshes by sorting all the cells with boundary faces according to their centroid, and then selecting them in the DFS visit of the DAG according to that ordering. It should be noted that this extension can produce an incorrect sort, because the occlusion relation does not always agree with the centroid distance. Another possible extension to managing non-convex meshes is the *filling* of non-convex parts with new cells, in order to obtain a convex complex. This filling operation can be very difficult, like any constrained threedimensional triangulation, and it can create a quadratic number of cells; this will increase the final time and space complexity of this approach.

The robustness of this approach depends on the accuracy of the computation of the occlusion relation between adjacent tetrahedra when considering faces almost perpendicular to the viewpoint; an error in the orientation of a link in the DAG may result in the creation of cycles and therefore sorting errors; the solution proposed by Williams [114] of considering almost perpendicular faces as not related may still introduce visible errors in sorting. The use of geometric primitives [91] that guarantee an error bounded by the machine precision, can solve most of these problems in most cases by introducing a small degradation of performance.

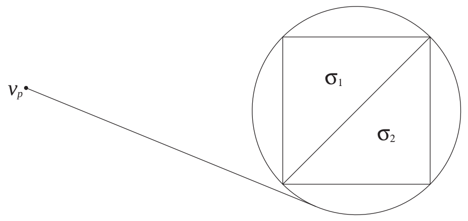


Figure 3.3: Numerical sorting fails on degenerate Delaunay meshes.

Numerical Sort A different technique working only on tetrahedral complexes was proposed in [72]; this technique does not need to store the adjacency relation between tetraehedra and is based on the following property of Delaunay tetrahedral complexes:

> the length of the tangent from the viewpoint to the sphere circumscribed to a tetrahedron reflect the depth ordering of the complex.

A detailed discussion of this property and its relation with power diagrams is presented in Section 3.3. To use this technique the centers c_i and radii r_i of all the tetrahedra σ_i of the complex must be precomputed once and stored together with the dataset. Then, in order to sort the tetrahedral complex it is sufficient to calculate for each tetrahedron σ_i the square of the distance from the current viewpoint to centers c_i, subtract from it the squared radius r_i, and sort the resulting values numerically.

The worst case complexity of this technique is $O(n \log n)$ with n the number of tetrahedral cells. The main drawback of this technique is its applicability only to Delaunay complexes and its sensitivity to Delaunay degeneracies. This sort can fail on many common datasets, such as those obtained by regularly decomposing hexahedral cells in 5/6 tetrahedra: all these 5/6 tetrahedra shares the same circumsphere, so the ordering between them is not calculated at all.

Numerical sorting can be considered, in some aspects, more robust than topological sorting; its weakness is due to the shape of the dataset itself rather than on the particular viewpoint chosen. Moreover, despite its higher worst-case complexity, running times of the two techniques are comparable [24].

This technique can be succesfully applied also to simplicial complexes that are subsets of Delaunay complexes such as the alpha shapes [40]; therefore, as explained in Corollary 1, this technique of sorting is not inherently limited to convex and/or connected domains.

Newell, Newell and Sancha's Sort Acyclic tetrahedral meshes can also be sorted using an extension of the Newell, Newell and Sancha (NNS) sort algorithm [97]. In the same way as the original sorting algorithm [76] for polygons, the sorting process is organized into two phases. In the first phase the vertices are view transformed and a preliminary approximate sort of the polyhedra is calculated according to the rearmost z component of each polyhedron. The second step is a *fine tuning* of the sort, organized into checks of increasing computational complexity, similarly to the original NSS algorithm.

The goal of *fine tuning* is to find a separating plane between two polyhedral cells which permits us to deceive the correct cell drawing order easily. Given a cell σ on the top of the z rearmost ordered list (result of the first phase) it can be safely rendered only if it does not overlap any cell in the list with a rearmost z less then the nearmost z of σ. If an overlapping cell is found, then it is put on the top of the list and the process is started again. The existence of cycles can be detected by tagging every overlapping cell and testing if a cell is involved in an overlapping more than once. No solutions for breaking the cycles are presented.

The worst case complexity of this approach is $O(n^2)$, and the running times presented in [97] are much higher than the ones of the two previous algorithms. On the other hand, this algorithm is the only one which correctly handles any class of acyclic polyhedral complexes.

3.2.2 Final considerations on sorting tetrahedral complexes

In table 3.1 we summarize the main characteristics of algorithms for sorting three-dimensional cell complexes. It is worth noting that the de Berg's algorithm has never been practically adopted for visualization purposes: its presence in this table denotes the best theoretical result for sorting a generic cell complex.

	Topological	Numerical	de Berg	NNS
Class of complexes	convex complexes	Delaunay (subset of)	any complex	any complex
Worst Case Complexity	$O(n)$	$O(n \log n)$	$O(n^{4/3+\epsilon})$	$O(n^2)$
Storage OverHead	$O(n)$	$O(n)$	$O(1)$	$O(1)$

Table 3.1: Results on computing depth orders for tetrahedral complexes.

It should be noted that the above algorithms can generate a correct depth ordering only if the starting complex is acyclic with respect to the specified viewpoint. Practical algorithms for testing the acyclicity of a simplicial complex for any viewpoint are not known. A brute-force algorithm, based on the idea of depth-sorting the complex from all significant possible viewpoints, was sketched as a personal communication between H. Edelsbrunner and P. Williams [114]. The idea is to place all the significant viewpoints in all the cells γ of the plane arrangement $\mathcal{H}$ generated by the partition of the space with planes passing through all the faces of the cells. All the points p in the same cell γ of $\mathcal{H}$ certainly share the same depth ordering of the original complex Σ w.r.t. p, because the occlusion relation between two cells can change only when crossing the plane affine to one of the faces of the two cells. The main drawback of this approach is its complexity, since the arrangement of the planes affine to facets of a simplicial complex with n vertices can contain $O(n^3)$ cells, therefore if we sort the complex foreach reasonable direction using a topological approach we obtain a $O(n^4)$ complexity that limits the algorithm usability.

3.3 Power Diagram Sorting

In this section we describe the links between acyclic complexes in $\mathbb{E}^{\mathrm{d}}$ with convex polyhedra in $\mathbb{E}^{\mathrm{d}+1}$ and power diagrams, and how to exploit these relations to depth sorting a projective complex.

3.3.1 Power diagrams and convex polyhedra

In this subsection we explore some of the connections between convex polytopes and power diagrams, giving a simple proof of the acyclicity of a projective simplicial complex.

Let $\mathbb{E}^{\mathrm{d}+1}$ be spanned by the coordinate axes $x_1, \cdots, x_{d+1}$ and let h_0 denote the hyperplane $x_{d+1} = 0$. The following result, that relates PD and convex polyhedra, is presented in [6]:

Theorem 2. *(Aurenhammer [6]) For any (d+1)-polyhedron P, which can be expressed as the intersection of upper halfspaces, there exists an affinely equivalent power diagram in h_0, and viceversa.*

It is assumed, without loss of generality, that all the upper halfspaces generating the polyhedron P cross the unitary paraboloid $U : x_{d+1} = \sum_{i=1}^{d} x_i^2$. This theorem is based on the following transform Π that maps a sphere $s \subset h_0$ with center z and radius r in the hyperplane:

$$\Pi(s) : x_{d+1} = 2x \cdot z - z \cdot z + r^2 \tag{3.1}$$

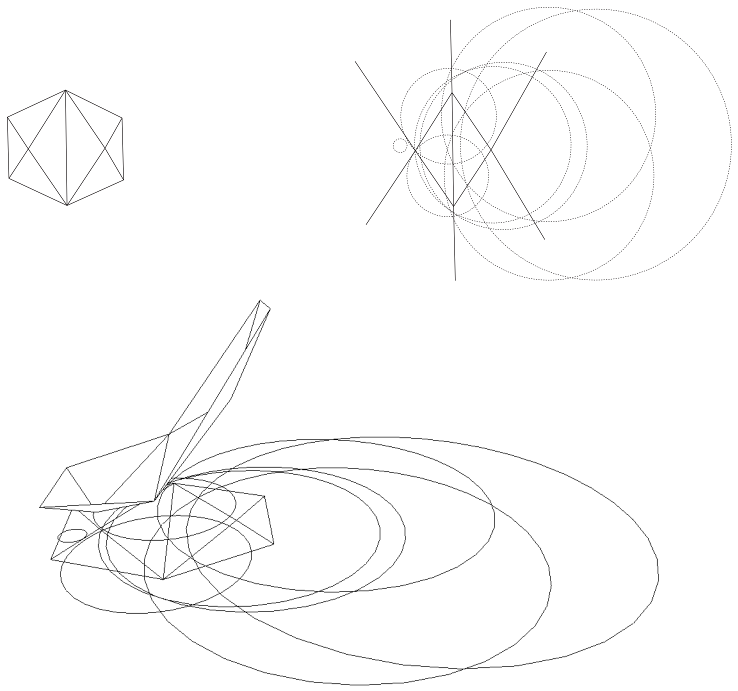


Figure 3.4: A 2D simplicial complex Γ (top left), and the corresponding Power Diagram (top right). Below a 3D representation of the complex Γ on the plane h_0, the lifted complex Γ^* and the resulting power circles on h_0.

The hyperplane $\Pi(s)$ crosses the unitary paraboloid U, such that the projection of $U \cap \Pi(s)$ onto h_0 is equal to s. Conversely, the mapping from planes crossing U back to spheres in h_0 can be defined.

An interesting property of this transform is that, if s and t are two non-concentric spheres in h_0, then $chor(s,t)$ is the vertical projection of $\Pi(s) \cap \Pi(t)$ onto h_0. Therefore, given a polyhedron P, it is possibile to obtain a set of spheres S whose PD is equal to the projection of P onto h_0.

3.3.2 Bounded and Unbounded Convex Polyhedra

In theorem 2 we referred to polyhedra formed by the intersection of non-vertical upper halfspaces, therefore unbounded infinite polyhedra whose projection onto h_0 is a partition of the whole plane. With the following theorem we extend this relation in order to be able to manage bounded polyhedra such as the lower part of a convex hull.

Theorem 3. *Let P be the lower part of a convex polyedron bounded by* *simplicial facets; there exists a power diagrams $PD(S)$ of a set of spheres S whose cells are superset of the projection of the facets of P onto h_0.*

Proof As in the previous case we assume, without loss of generality, that the hyperplanes affine to all the facets of P cross the unitary paraboloid U. Let P_0 be the projection of P onto h_0. Consider the halfspaces passing through the d-facets of P and let P' be the $(d+1)$-polyhedron which is obtained by the intersection of these upper halfspaces. The internal facets of P (the ones not having a $d-1$ facet on the boundary of P) have a direct correspondence in P'. Conversely the d-facets having one or more $d-1$ facets on the boundary of P lose these (d-1)-facets and become larger, possibly unbounded, d facets.

For theorem 2 we can build the PD that is affine to the projection P'_0 of P' onto h_0. It is easy to see that each d-facet σ of P_0 corresponds to a d-facet σ' of P'_0 such that $\sigma = \sigma'$ if σ is an internal d-facet, or $\sigma \subset \sigma'$ if σ is a boundary d-facet. □

In Figure 3.4 we show a 2D simplicial complex P_0 (top left) and the corresponding lower part of a convex polyhedron P in 3D (bottom); in the top right part of the figure we show the power diagrams resulting from the application of theorem 3.3.2 to P.

Acyclicity of projected convex polyhedra From the relations recalled in the previous paragraphs we can devise a new proof for the acyclicity of a simplicial complex that is the projection of the lower part of a convex polyedron in $\mathbb{E}^{d+1}$; we think that this proof is considerably simpler than the one presented in [39].

Given a viewpoint p, we introduce a numerical function Φ that *agrees* with the occlusion relation that is: $\Phi_p(\sigma)$ such that $\Phi_p(\sigma) < \Phi_p(\tau)$ if $\tau \prec_p \sigma$. Clearly if such a numerical function exists for a given viewpoint then $\prec_p$ is acyclic since it is impossible to have a set of cells $\sigma 1, \ldots, \sigma_k$ that forms an occlusion cycle:

$$\sigma_1 \prec_p \cdots \prec_p \sigma_k \prec_p \sigma_1$$

because the corresponding numerical function Φ should give:

$$\Phi_p(\sigma_1) < \Phi_p(\sigma_2) < \cdots < \Phi_p(\sigma_k) < \Phi_p(\sigma_1)$$

It easy to show that, if $\sigma \prec_p \tau$ and σ and τ are not adjacent, we can find a chain $\sigma \prec_p \sigma_1 \prec_p \cdots \prec_p \sigma_k \prec_p \tau$ such that all these simplexes are consecutively adjacent; for example such a chain can be built by choosing the d-simplices crossed by a line segment starting from p and crossing both σ and τ. For this reason we can simplify our proof and reduce it to the case of two (d-1)-adjacent simplexes.

Let Γ^* be the convex polyhedron whose projection onto h_0 is affine to Γ; for theorem 2 using the transform Π we can build the set of spheres S_Γ such that the power diagram $PD(S_\Gamma)$ is affine to Γ.

Let σ and τ be two (d-1)-adjacent simplexes of Γ, σ^* and τ^* be the two corresponding (d-1)-adjacent simplexes of Γ^*, s_σ and s_τ be the two spheres of S_Γ, such that $\text{cell}(s_\sigma) = \sigma$ and $\text{cell}(s_\tau) = \tau$; the hyperplane $\Pi(s_\sigma)$ contains σ^* and $\Pi(s_\tau)$ contains τ^*.

The (d-1)-face f common to σ and τ lies on the chordale $chor(s_\sigma, s_\tau)$, because, as observed in previous section, $chor(s_\sigma, t_\tau)$ is the vertical projection of $\Pi(s_\sigma) \cap \Pi(s_\tau)$ onto h_0.

Now consider the obstruction relation between the two simplexes, for the convexity of σ and τ, it holds $\sigma \prec_p \tau$ if and only if the viewpoint p belongs to the halfspace bounded by the hyperplane passing through the (d-1)-face f, that is $chor(s_\sigma, s_\tau)$; but $chor(s_\sigma, s_\tau)$ partition $\mathbb{E}^\mathrm{d}$ in the region where $pow(x, s_\sigma) < pow(x, s_\tau)$ and viceversa. Therefore $\sigma \prec_p \tau \rightarrow pow(p, s_\sigma) < pow(p, s_\tau)$, so the $pow(p, s_\sigma)$ can be considered the searched numerical function Φ.

3.3.3 Sorting a simplicial complex

The most interesting aspect of the results in the previous section is that they suggest a technique for depth sorting a simplicial complex in $\mathbb{E}^\mathrm{d}$ that is the projection of the lower part of a convex polyhedron in $\mathbb{E}^{\mathrm{d}+1}$. Infact given the viewpoint p it is sufficient to sort the d-cells σ_i of the complex according to $pow(p, s_{\sigma_i})$, where s_{σ_i} is the sphere obtained by the transformation Π from the plane h_i affine to the facet σ_i^*. Hereafter we will refer this approach to depth sorting as *Power Diagram Sorting* or PD sorting.

The main problem of this approach is that in the common cases we have just a simplicial complex Γ in $\mathbb{E}^\mathrm{d}$ and not Γ^*. In some special cases it is simple to find the convex polyhedron, for example if Γ is a Delaunay simplicial complex we can exploit the well known correspondence with convex hull in $\mathbb{E}^{\mathrm{d}+1}$ to find Γ^*. In this section, given a generic complex Γ we address the following problem, hereafter denoted as the *lifting problem*: finding a convex polyhedron Γ^*, if there exists one, such that Γ is the vertical projection of Γ^*.

Finding an efficient and usable solution to this problem means finding a new approach to the depth sorting problem. The most interesting aspect of this approach is the *clearness* of the structure needed for the sorting once the corresponding polyhedron has been found: for each simplex it is only necessary to store the power circle. Apart from the intrinsic simplicity of this approach, its flatness can be useful in the creation of data structures for secondary memory: any subset of the complex can be independently recovered and sorted using just the stored power circles.

Useful corollaries So far we have dealt with the lifting and projection of convex complexes, here we add two corollaries that extend the applicability of this technique of sorting.

Corollary 1. *The acyclicity theorem, and therefore the agreement between* Φ_p *and* $\prec_p$ *holds also for a complex* Γ' *that is a subset of a complex* Γ *that is obtainable as a projection of a convex polyhedron* Γ^**.*

The proof of this corollary immediately derives from the fact that the sort is based only on the numerical value of the Φ function; once this function has been calculated we can discard part of the complex Γ without any risk.

This remark permits us to use the PD sorting also for non-convex complexes which are subsets (maybe *carved out*) of large convex complexes. An important class of such complexes is, for example, Edelsbrunner alpha shapes [40].

Corollary 2. *Given a complex* Γ *obtainable as a projection of a convex polyhedron* Γ^**, the acyclicity theorem, and therefore the agreement between* Φ_p *and* $\prec_p$*, holds also for a complex* Γ' *having the following property: for each d-cell* $\sigma' \in \Gamma'$ *it is possible to find a d-cell* $\sigma \in \Gamma$ *such that* $\sigma' \subseteq \sigma$ *and if* $\sigma', \tau' \in \Gamma'$*,* $\sigma, \tau \in \Gamma$*,* $\sigma' \subseteq \sigma$*,* $\tau' \subseteq \tau$ *then* $\sigma' \neq \tau' \rightarrow \sigma \neq \tau$*.*

Proof The agreement of Φ_p function with $\prec_p$ relation depends on the intepretation of chordales as separating planes. Obviously these separation works also if the cells are smaller and totally included in the larger power cells of the power diagram. Finally the last condition of this corollary ($\sigma' \neq \tau' \rightarrow \sigma \neq \tau$) requires that for each cell of the PD there is at most one smaller cell. $\square$

This means, in other words, that we can use the PD sorting for complexes whose d-cells can be seen as *shrinking* of larger cells, therefore the requirements for the $d+1$ convex polyhedron that we search are that its orthogonal projection be a complex whose d cells cover all the cells of our d-complex.

A consequence of these corollaries is that this approach can also be used to sort scattered triangles in space: in this case, for each triangle f, it is necessary to find a supporting tetrahedron σ such that f belongs to the facets of σ and σ does not intersect any other triangle. This can be done by adding a vertex sufficiently close to the face f. Using this approach we could sort a set of triangles with a complexity of $O(n \log n)$, with just a linear storage overhead (the center and radius for each sphere).

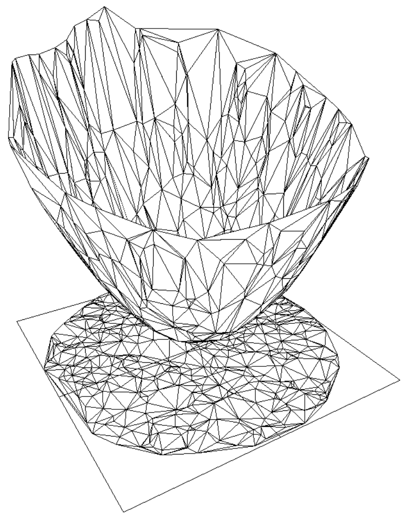


Figure 3.5: A lifted complex of ≈ 1000 facets in $\mathbb{E}^2$.

3.3.4 Lifting a tetrahedral complex

The *lifting problem* consists in lifting each vertex v of a complex Γ in $\mathbb{E}^{\mathrm{d}}$ along the $d+1$ axis in order to obtain a convex polyhedron in $\mathbb{E}^{\mathrm{d+1}}$. The first observation is that such a problem does not have a single solution: there exists an infinite number of different convex polyhedra having the same projection onto h_0, and therefore there are infinite sets of spheres that can be used to sort our complex.

Now we will formulate the lifting problem as a linear programming problem. The lifted complex is convex if and only if the dihedral angle between any two lifted $(d-1)$-adjacent simplex is convex. This condition can be expressed in the following way: let σ and τ be two simplexes that are (d-1)-adjacent through the common (d-1)-face f; let v_σ and v_τ be the two non shared vertices of σ and τ,

$$h_\sigma : x_{d+1} = \mu_{\sigma,1}x_1 + \mu_{\sigma,2}x_2 + \cdots + \mu_{\sigma,d}x_d + k_\sigma$$

be the equation of the non vertical hyperplane in $\mathbb{E}^{\mathrm{d+1}}$ passing through σ.

The convexity of the dihedral angle between σ and τ, is guaranteed if the projection of v_τ onto h_σ is strictly higher than the $d+1$ coordinate of v_τ:

$$v_{\tau,d+1} > \mu_{\sigma,1}v_{\tau,1} + \mu_{\sigma,2}v_{\tau,2} + \cdots + \mu_{\sigma,d}v_{\tau,d-1} + k_\sigma$$

It can be observed that the coefficients $\mu_{\sigma,i}$ linearly depend only on the vertices of σ, so we can express such linear inequalities having as unknown variables the $d+1$ coordinates of vertices of σ and v_τ.

In this way we can express the lifting problem as a set of m linear inequalities, with n unknowns, where n is the number of vertices, and m is the number of internal (d-1)-facets of Γ.

A solution of this system can be easily found used the simplex algorithm; this requires to transform all the strict inequalities by adding a slack constant δ in order to fix the minimum distance between v_τ and h_σ:

$$v_{\tau,d+1} - \mu_{\sigma,1} v_{\tau,1} - \mu_{\sigma,2} v_{\tau,2} - \cdots - \mu_{\sigma,d} v_{\tau,d-1} - k_\sigma \geq \delta$$

and to set as objective function the minimization of the complessive sum of the new (d+1) coordinates of vertices. It should be noted that the coefficient matrix of this LP problem is very large, but fortunately it is very sparse: each line of the system has at most d+2 non zero elements.

If the simplex algorithm does not find a solution we have demonstrated that the given complex Γ is not a projective one. It is still an open problem whether this fact implies the existence of a viewpoint p such that the $\prec_p$ relation for Γ is cyclic.

Some experiments in two dimensions showed that the solution of the linear programming problem generated by a complex of one thousand triangles (Figure 3.5) can be found in less than a minute on a small personal computer using a public domain LP solver. This preliminary result appears to be a reasonable preprocessing step, and it allows us to claim the practical relevance of the proposed solution.

Non Convex Simplicial Complexes The technique presented works only if the complex to be lifted is convex; for corollaries 1 and 2 we know that if we consider a non-convex complex Γ' as a subset of a larger convex polyhedron Γ, that is projection of the lower part of a convex polyhedron Γ^* in $\mathbb{E}^{d+1}$, our sorting method is still applicable, but the lifting technique previously proposed does not work.

We can solve this problem including for each (d-1)-face f on the boundary, with f belonging to a simplex σ, the constraint that the halfspace h_σ in $\mathbb{E}^{d+1}$ passing through the lifting of σ, does not contain any other lifted vertex. It should be noted that this approach can increase the number of constraints of the linear programming problem from $O(m)$ to $O(m^2)$. An application of this techinque to the lifting of a non convex two dimensional complex is shown in Figure 3.6.

3.4 Conclusions

In this chapter we have presented a new approach to the problem of depth sorting a complex. The approach is based on the connection between power

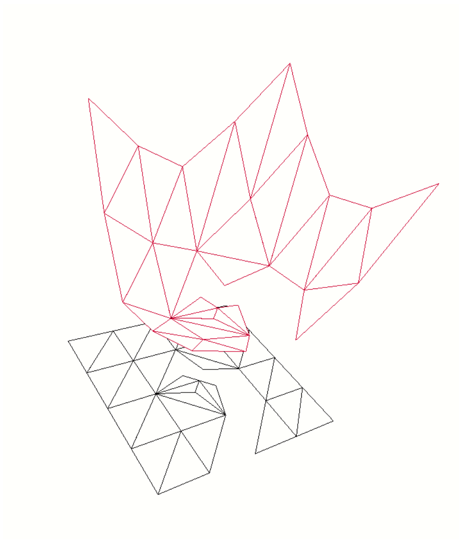


Figure 3.6: A the lifting of a 2D non-convex 2D complex.

diagrams and the class of projective complexes. Given a simplicial complex $\Gamma = \{\sigma_0, \ldots, \sigma_n\}$ in $\mathbb{E}^{\mathrm{d}}$ this approach has two steps:

- **Preprocessing:** lift the complex in $\mathbb{E}^{\mathrm{d+1}}$ obtaining a complex Γ^* that is the lower part of a convex polyhedron in $\mathbb{E}^{\mathrm{d+1}}$. From Γ^*, using the Π transformation defined in (3.1), build the set of spheres $\S_\Gamma = \{s_{\sigma_0}, \ldots, s_{\sigma_n}\}$, one sphere foreach simplex of Γ, such that $PD(S)$ is affine to Γ.

- **Depth-Sort:** Given a viewpoint p, calculate, for each simplex σ_i, $pow(p, s_{\sigma_i})$ and sort the simplices according these numerical values.

Summarizing, we can briefly describe some pros and cons of the proposed depth sorting technique:
Pros:

- **run-time efficiency:** the depth sorting consists in the calculation of power distance for each cell and the simple sorting of these values;

- **data structure simplicity:** for each cell we need to store just a center of a sphere and a radius, there is no need for face-adjacency information;

Cons:

- **time consuming preprocessing:** the lifting in $\mathbb{E}^{\mathrm{d+1}}$ of the original complex can be a very long process for large non convex datasets, this fact also implies that the visualized dataset cannot be interactively modified; the definition of better algorithms for the lifting problem is a critical issue;

- **universality:** we do not know if any acyclic complex can be lifted and then depth sorted using our technique;

Besides this innovative sorting approach, this chapter provided also a constructive solution (the lifting technique) to the problem of testing if a given complex is projective (and therefore acyclic). It should be noted that if we fail to lift the complex we have a proof that the given complex is not a projective one.

As a final note, we want to remember that it is still an open problem the exact relation between the class of acyclic complexes and the class of projective complexes. We do not know if a complex Γ that is not projective is necessarily a cyclic complex, that is, if Γ not projective implies the existence of a viewpoint p such that the $\prec_p$ for Γ is cyclic.

Chapter 4

Integrating DVR and Isosurfaces

In this Chapter we present an original projective technique which is able to correctly integrate isosurfaces and direct volume rendering. The proposed technique is based on a tabular on-the-fly decomposition of the tetrahedral cells crossed by isosurfaces. In the second part of this Chapter we introduce the concept of Discontinuos Transfer Functions; this concept unifies, in a single framework, the management of visualization of isosurfaces and direct volume rendering in the visualization of volume data.

The two most commonly used techniques to inspect volume data, isosurface extraction and direct volume rendering, cannot be easily integrated if we use a projective approach. The intersection of isosurfaces with tetrahedral volume elements can generate rendering artifacts due to the incorrect depth ordering of isosurface facets and tetrahedra. No trivial solution (e.g. drawing the isosurface facet f before/after the tetrahedron containing f) can give a correct solution to this problem.

The main idea of our approach is to split each tetrahedron along the internal isosurface patches and project in the correct order all the resulting parts; we will show that such cutting can be implicitly done during the rendering process using a tabular approach driven by the class of the isosurfaces crossing the tetrahedron. In Section 4.1 we show the basic splitting technique in the simple case of a single isosurface; in the next section we extend this approach to manage the case of multiple isosurfaces crossing a single tetrahedron. In Section 4.3 we apply the developed techniques for integrating DVR and isosurfaces in order to solve the problem of the correct rendering of discontinuous transfer functions, and we introduce the Discontinuos Transfer Function unifying in a single framework the visualization of volume data with isosurface and direct volume rendering.

4.1 Integrating a Single Isosurface with DVR

A tetrahedron σ crossed by a facet f of an isosurface can be correctly rendered if we cut it along the isosurface, decompose the two resulting blocks into smaller tetrahedra and project, in the correct order, these tetrahedra and the isosurface f. In this chapter we show that this process can be done automatically and efficiently at rendering time using a table-driven approach. A sketch of this process is shown in Figure 4.1. It can be immediately seen that this splitting process does not involve the creation (and therefore the computation) of any new vertex, all the resulting tetrahedra have vertices that belong either to σ or to f. How to build the final tetrahedra starting from vertices of σ and f depends only on *how* the isosurface crosses σ, i.e. it depends on the class of the isosurface.

All of these decomposition and splitting could be obviously done in a preprocessing phase (immediately after the extraction of the isosurface) and the resulting mesh could be stored and managed as usual, but such a naive approach has two main negative aspects:

- storage overhead: the splitting and decompositions can generate a large number of tetrahedra, that must be deleted and recreated every time the user changes the isosurface threshold.

- increasing in the time for the isosurface extraction: when creating the isosurface all the new tetrahedra must be created and conveniently inserted in the data structure storing the tetrahedral complex; this operation can be time consuming (e.g. updating all the adjacency information).

Fortunately there is no need to explicitly store the splitted parts of each tetrahedron, but they can simply be built on-the-fly with a table driven process during the tetrahedra projection starting from the isosurface class. Moreover we observe that the depth ordering of the tetrahedra resulting from the splitting is independent w.r.t. the global depth ordering of the original tetrahedral mesh, and therefore can be computated on the fly for each splitted tetrahedron during the rendering phase. Such local sort (local to each splitted tetrahedron) can be accomplished in two steps: first, determine the depth ordering between the two block, then sort the tetrahedra derived from the decomposition of each block. This process will be described later in section 4.1.2.

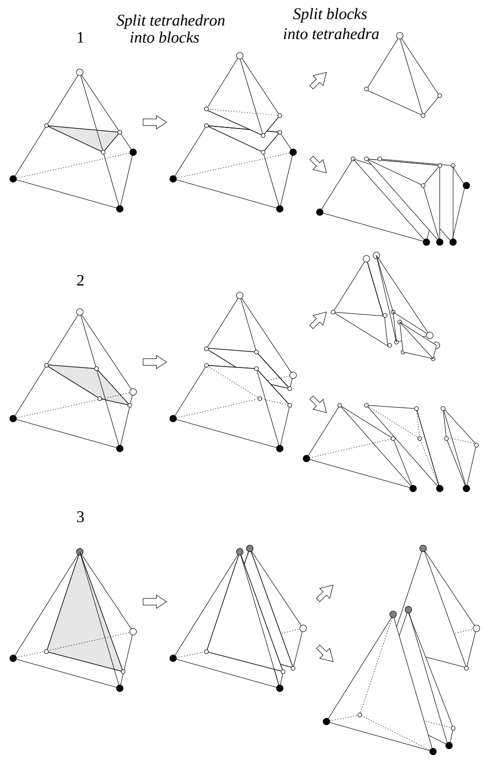


Figure 4.1: Splitting a tetrahedron with an isosurface into blocks and blocks into tetrahedra: all the configuration can be combinatorially determined starting from the Marching Tetrahedra vertex configurations.

4.1.1 Splitting a Tetrahedron

To efficiently decompose a tetrahedron during the rendering process we must carefully code all the needed information into a table. We assume that, for each tetrahedron σ, we can retrieve the isosurface facet crossing it, and the class `c` the tetrahedron belongs to (determined during the isosurface extraction). As explained in Section 2.2 the class number, in the range $0..3^4$, classifies the tetrahedron vertices according to their field value w.r.t. the threshold δ of the isosurface and unambiguously determines the shape of the isosurface facet crossing σ.

Our table has one entry for each `c` value and describes the resulting blocks in terms of vertices and tetrahedra composing them. These blocks have vertices in both the sets of vertices of dataset Σ and of the isosurface itself and these two sets are commonly distinct, so an adequate addressing/referring technique must be used. For each of the 81 entries of the table we store two block descriptions: the first one is relative to the block below the isosurface threshold and the second one to the above. One or both of these block descriptions can be empty if, respectively, the isosurface crosses one of the facets of σ or if the tetrahedron is not crossed by the isosurface. Each block description contains the following information:

- the number of the tetrahedra in which the block has to be decomposed;
- the indexes of the vertices of each tetrahedron composing the block.

The following simple mapping strategy can be used to address the vertices of the block using the tetrahedron and isosurface vertices: the indexes of the vertices of the block are denoted with integers in the range [0..7], indexes in the range [0..3] refer to the tetrahedron vertices and indexes in the range [4..7] refer to the isosurface vertices.

It should be noted that, by exploiting the symmetry of the isosurface configurations, the crossing of an isosurface can decompose a tetrahedron only in five different ways corresponding to the five isosurface classes. Therefore the size of the table describing the decomposition of the blocks into tetrahedra could be reduced to only five entries. However the mapping of isosurface and tetrahedron vertices into block vertices must always be done on the basis of a larger 81-entry table, therefore we decided for the simpler approach of using a unique, although larger, table.

4.1.2 Sorting the decomposition

Once the decomposition of the original tetrahedron σ has been determined, the resulting smaller tetrahedra have to be correctly depth ordered. We execute this depth sort in two step: first we sort the two blocks and then we separately sort the tetrahedra of each block.
To depth sort the two blocks it is sufficient to look at the orientation of the isosurface plane w.r.t. the point of view. Similarly the correct depth ordering of tetrahedra composing each block depends only on the orientation of the facets internal to the block, as in the case of topological sort. For example, if we have an isosurface of class 1^+ (like the one in Figure 4.1) tetrahedra forming the triangular prism block are sorted on the base of the orientation of the two internal facets with respect to the viewpoint. The choice of the internal facets to be used to sort each block is table driven for efficiency reasons. Moreover we can exploit the smallness of our blocks (they are composed of at most three tetrahedra) to code in the table also the relation between the internal face orientation and the resulting depth order. Infact we have at most two internal facets so the possible orderings are[1] at

[1]We can consider all the internal facets that are perpendicular to the observer as directed in an arbitrary way without losing the resulting depth order correctness.

most 2^2, and it is reasonable to code these orderings into each row of the table. At rendering time we will check the internal facets orientation w.r.t. the viewpoint and we can immediately retrieve the corresponding ordering without storing or needing adjacency information between the tetrahedra inside each block.

For sake of completeness we should note that also the adjacency information could be stored into the table and convenientely exploited by a standard topological sor; the setup time to convert these information from the table coding format to the one necessary for the topological sorting is too high for an operation that should be done for each tetrahedron during the projection. Therefore we choose to add the following information to each block entry of our table:

- the number f_i of internal facets of the block;
- the indexes of the vertices of each internal facet;
- the 2^{f_i} depth orderings of the tetrahedra of the block.

The final table is composed of 81 entries and, by storing all the indexes in one byte, its size is less than two kilobytes. We can now summarize the whole splitting process executed at projection time for each tetrahedron σ that is crossed by an isosurface f:

- using the isosurface facet orientation, decide the depth ordering between the two blocks and the isosurface facet;
- following this order we process and render the two blocks and the isosurface facet. For each block:
  - recover from the table the internal facets, compute their orientation w.r.t. the viewpoint;
  - use the result to locate in the table the depth order of the tetrahedra composing each block;
  - following this order draw the tetrahedra out of the table and project them.

4.2 Multiple Isosurfaces

The technique proposed in the previous section works only if we have to render a single isosurface. If we need to extract and visualize many isosurfaces simultaneously, a tetrahedron can be cutted by more than one isosurface.

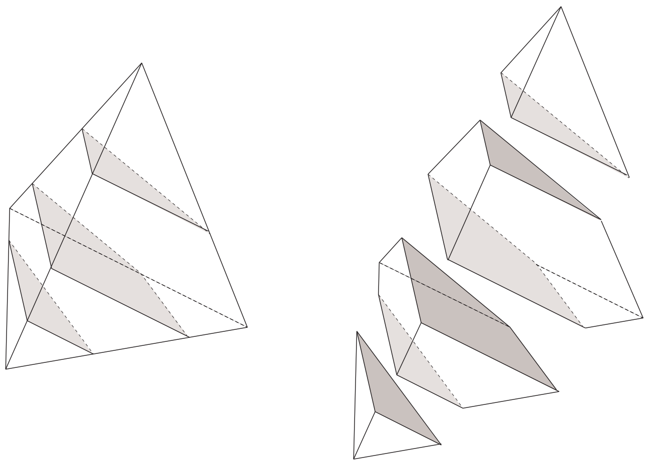


Figure 4.2: A tetrahedron crossed by multiple isosurfaces is splitted into convex blocks.

Fortunately, all the isosurface crossing the same tetrahedron are parallel (because of the linearity of the interpolation function inside the tetrahedron), therefore the possible different decompositions, resulting from multiple isosurface cuts, are simple enough to be classified and managed using a tabular approach as in the previous case.

To determine all the possible shapes of the resulting blocks we consider a tetrahedron cutted by k parallel planes (the isosurfaces) into $k+1$ convex blocks. The shape of each block depends only on the class $\mathsf{c_i}, \mathsf{c_j}$ of the two cutting isosurfaces that bound it. Let's denote with $i.j$ the block between two isosurfaces of class i and j. We consider the isosurfaces in order of increasing threshold value, in other words we step from vertices below the threshold δ to vertices above δ. The first and last block are usually denoted with $0.i$ and $i.0$, but we can focus our attention on the shape of the generic inner blocks $i.j$ because, for the blocks of class $0.i$ and $i.0$, bounded by a single isosurface, the table described in the previous section can be used. In the case of multiple isosurfaces our table should also give all the information about a generic $i.j$ block.

It is immediate to see that most of the 81×81 cases are not possible, infact if we consider two threshold $\delta_1 < \delta_2$ the vertices of the tetrahedron can be classified only in five different ways according their field value w.r.t. δ_1, δ_2. Therefore we use a five values[2] classification of the vertices of the tetrahedron that can give $5^4 = 625$ different classifications of σ w.r.t. two

[2]We adopt the following classification convention: '0' if the vertex field valued ϕ is less than δ_1, δ_2, '1' if $\phi = \delta_1$, '2' if $\delta_1 < \phi < \delta_1$, '3' if $\phi = \delta_2$ and '4' if $\phi > \delta_2$.

isosurfaces. Most[3] of these 625 configurations correspond to tetrahedra that are crossed by none or just a single isosurface, in this cases there are no inner blocks, so we leave the corresponding rows of the table empty.

The resulting table is therefore organized to manage the decomposition of the inner block only, and contains the following information:

- the number of the tetrahedra in which the block has to be decomposed;
- the indexes of the vertices of each tetrahedron composing the block.

The following simple mapping strategy can be used to address the vertices of the block using tetrahedron and isosurface vertices: the indexes of the vertices of the block are denoted with integers in the range [0..11] , indexes in the range [0..3] refer to the tetrahedron vertices, indexes in the range [4..7] refer to the first isosurface vertices and indexes in the range [8..11] refer to the second isosurface vertices.

4.2.1 Depth sorting

In this section we show how to sort simplicial complex generated by splitting a single tetrahedron with many isosurfaces. First of all we must be sure that such sorting always exists; for this reason we introduce the following theorem:

Theorem 4. *The simplicial complex generated by splitting a single tetrahedron with many isosurfaces is always acyclic.*

Proof. We prove this statement in two steps. As first step we note that the cell complex formed by the convex blocks $i.j$ is acyclic, because there exists a set of distinct parallel planes separating the blocks, so it is impossible to find a cyclic ordering without crossing twice one of these separating plane. The second step of this proof is that the tetrahedral complex obtained subdividing a block is still acyclic. This decomposition is generated, as explained in section 4.2.2, as the Delaunay triangulation of the block vertices calculated on a reference equilateral thetrahedron. If we consider the affine transformation that maps the reference equilateral tetrahedron into a generic one, we can easily see that it preserves the acyclicity property of the complex.

To effectively calculate the depth ordering of the decomposition of a tetrahedron crossed by k isosurfaces, we propose the following two-steps algorithm: first we sort the convex blocks $i.j$, then we sort the tetrahedra

[3]Only 198 configurations give blocks bounded by two isosurfaces

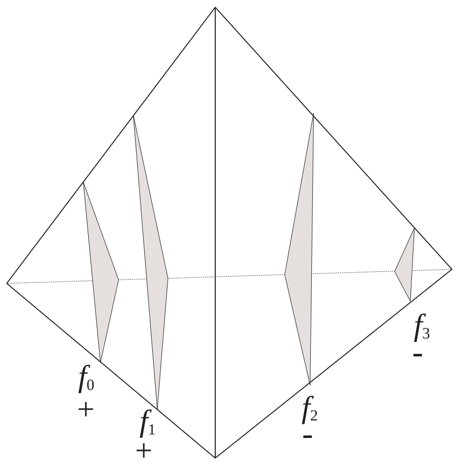


Figure 4.3: The blocks of a tetrahedron crossed by multiple isosurfaces are sorted using the orientations of isosurface facets.

composing each block independently. We assume that, for each tetrahedron σ, we can immediately retrieve the list of k isosurface facets f_i crossing σ, together with their classes c_i, ordered by increasing threshold and that the isosurface facets have the normal agreeing the field growing direction.

To find the correct depth ordering of the blocks a topological sort can be applied. In this case the topological relation between blocks is trivial (all the blocks are along a row) so this procedure can be furtherly simplified. As already stated we assume that the isosurface are sorted w.r.t. increasing threshold values and their normals agree with the field gradient inside the tetrahedron. To sort the blocks we classify isosurface facets f_i w.r.t. the viewpoint ('+' if they see the viewpoint, '-' otherwise) and store in an array the result. Then get all the blocks with agreeing '+' according to the order of the array, followed by the ones with agreeing '-' in inverse order and as last block (if it exists) the one with disagreeing signs. In figure 4.3 we show an example of this technique of sorting; the tetrahedron in figure, seen from the viewpoint, is crossed by four isosurfaces and splitted into five blocks, under each isosurface facets the result of the classification is shown; the resulting depth order of the blocks (back-to-front) is $0.f_0$, $f_0.f_1$, $f_3.0$, $f_2.f_3$, $f_1.f_2$.

Once the blocks are ordered we can independently depth sort the tetrahedra forming each block $i.j$ using the orientation of facets internal to $i.j$ and exploiting the same approach proposed for the single isosurface case: we store in each row of the table all the orderings resulting from all the 2^q possible orientations of the q internal facets of the block. Even if the number of tehtrahedra composing a block is larger than in the single isosurface

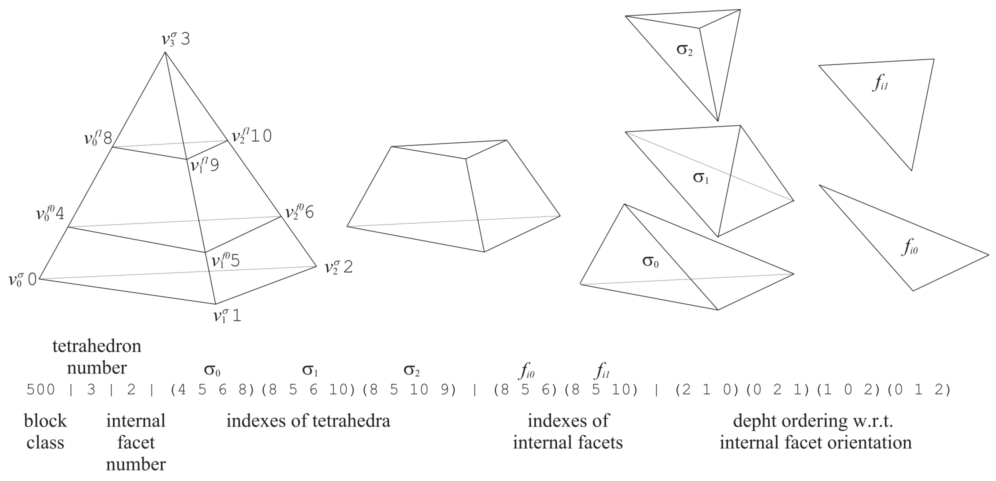


Figure 4.4: A row of the table indicating how to decompose and sort a block bounded by two isosurfaces.

case (a block can be composed up to six tetrahedra), this approach remains convenient and the table size is still manageable.

At the end of this ordering process we have a sorted list of tetrahedra, whose vertices are chosen between the ones of the original tetrahedron and the vertices of the isourface facets inside it, that can be classified and rendered using projective techniques. Figure 4.4 shows a row of our table. The first number (500) denote the configuration of the block, as resulting from our five values vertex classification. The tetrahedron in figure has vertices $v_0^\sigma, v_1^\sigma, v_2^\sigma$ under the first isosurface threshold and v_3^σ above both isosurface thresholds, giving a block configuration number of $0 \cdot 5^0 + 0 \cdot 5^1 + 0 \cdot 5^2 + 4 \cdot 5^3 = 500$. The second and third number denote the number of tetrahedra and internal facets resulting from the decomposition of the block. The following three quadruples indicate how to build the three tetrahedra $\sigma_0, \sigma_1, \sigma_2$ of the decomposition; similarly the next two triplets give the indexes of the two internal facets f_{i0}, f_{i1}. These indexes are given using the mapping strategy presented above; near to each vertex v_0^σ... is shown the corresponding table index. The last four triples codify the possible depth sorting of the tetrahedra w.r.t. the orientation of the internal facets f_{i0}, f_{i1}.

4.2.2 Building the table

Manually coding large tables, involving geometric properties, is a tedious and error prone process and its debugging is difficult. The choice of not

BuckyBall Dataset (176687 Tetrahedra)							
thr	Isosurf. size	Total Tetra	Splitted Tetra Overhead	Overhead Factor	Trivial Hybrid Times	Correct Hybrid Times	Trivial Times $\times$ Overhead
0.15	25,517	268,410	91,723	1.52	14.77	21.95	22.45
0.27	10,811	214,128	37,441	1.21	11.40	14.19	13.79
0.30	5,476	194,161	17,474	1.10	10.59	11.07	11.64

Table 4.1: Result of the integrating of isosurfaces and DVR using the splitting approach.

exploiting all the possible simmetries kept the table structure simple enough to permit the automatic filling of the table. To calculate the characteristics of each block we used an equilateral tetrahedron and generated on it all the 625 possible configurations of vertices values for two isosurface.

For each configuration we extracted the vertices of the corresponding block and built the Delaunay triangulation of them. For the convexity of the blocks, we obtained a decomposition of each block. Particular attention had to be payed to the geometric robustness of the Delaunay triangulation code adopted because most of the blocks presents degeneracies (5 or more cospherical points).

4.2.3 Experimental Results

The technique proposed has been implemented and tested; here we present some results of the first, rather unoptimized, implementation. The timings and the images refer to the integration of a the single isosurface with DVR and were executed on a SGI Indigo2 workstation (MIPS4400 200MHz). The case of multiple isosurfaces presents similar timings because in most cases the cells crossed by many isosurfaces are a very small portion of the dataset.

Table 4.1 reports results on the integration of three different isosurfaces with DVR. The experiment was run on a 32^3 subsampling of the Buckyball dataset (see chapter 5 for a more detailed description). The table reports the isosurface threshold, the number of facets of the extracted isosurface, the total number of the tetrahedra projected using the splitting procedure, the number of tetrahedra of overhead w.r.t. the original dataset, and the relative overhead. The last three columns reports respectively the rendering time for the trivial integration of isosurfaces and DVR (the isosurface facet is drawn before its tetrahedron), for the proposed approach, and the product between the relative overhead factor and the trivial integration rendering time.

It can be noted that the rendering time of our approach is very similar or lower than the time for a naive approach multiplicated for the relative increase of tetrahedra, therefore the time for splitting on-the-fly the tetrahedra crossed by the isosurface is a negligible portion of the total time of rendering.

4.3 Discontinuous Transfer Functions

In the framework presented in 2.1 both isosurface extraction and transfer function mapping belong the *modeling* part of the volume visualization process, the part in which a dataset becomes a geometric entity with well defined visual attributes. Here we try to unify these two visualization techniques by introducing a new single concept: the Discontinuous Transfer Functions (DTF). The main idea is to permit to the transfer function to have a number of discontinuity points where the mapping of field values into colors can sharply change.

We denote with $\mathcal{C}$ the color/opacity space; given a color $c \in \mathcal{C}$ we denote with $c^{\mathsf{r}}, c^{\mathsf{g}}, c^{\mathsf{b}}, c^{\alpha}$ the red, green, blue and opacity components of c. A DTF $\mathcal{T}$ is a piecewise linear function $\mathcal{T} : \mathbb{R} \to \mathcal{C}$ with a finite set of C^0 discontinuity values $D = d_1, \cdots, d_k$ in which for each $d_i \in D$, we can have:

$$\begin{array}{l} \lim_{v \to d_i^-} \mathcal{T}(v) = c_i^- \\ \mathcal{T}(d_i) = c_i \\ \lim_{v \to d_i^+} \mathcal{T}(v) = c_i^+ \end{array}$$

where $c_i^-, c_i, c_i^+ \in \mathcal{C}$ are, possibly different, color/opacity values. This class of functions permits the definition of TF's that are able to combine both the benefits of DVR and isosurface extraction in a single image:

- sharp discontinuities give, just like isosurfaces, a quantitative information permitting the exact localization of the regions in which the field assumes a given value;

- smooth color variations give a qualitative information about the field variations in space and the direction and intensity of the field gradient.

It is obvious that a DTF must be correctly rendered in order to be useful and, while this can be effectively done for some rendering approach (e.g. ray tracing) it can be difficult when projecting tetrahedral meshes; in Section 4.3.1 we deal with this problem.

It could be objected that DVR techniques based on ray-tracing of regular datasets have always been able to display both semi-transparent medium and isosurfaces [37, 66] and so the innovative contribution of our proposal could seem vague. Beside our original technique for table driven tetrahedra decomposition which permits, as shown in Section 4.3.1, the correct rendering of DTF to projective solutions, our main contribution is that we do not simply mix isosurface and DVR but we propose a single visualization modeling tool, the DTF's, that allow us to map the dataset into a geometric object with well defined visual attributes in such a way that the benefits of both DVR and isosurfaces can be exploited.

Other works have faced problems that are somewhat related with our approach, either in the direction of extending the isosurface technique towards DVR or, viceversa, using trying DVR to rendering not sharply defined surfaces. Here we briefly summarize present these related works, marking how DTF's cover all the presented solutions.

Shell Rendering Udupa and Odhner presented a data structure model for volume rendering called *shells* [99]; it roughly consists of a set of voxels in the vicinity of a surface structure that is not very sharply defined, and therefore it is difficult to extract with a classic isosurface algorithm. Using our DTF model case the corresponding DTF is the one completely transparent except for an opacity spike around the *interesting* value. While in regular datasets the uniform smallness of the cells permits us to ignore the problem of rendering voxels that are partially covered by the interesting fuzzy region, on tetrahedral domains we must resort to the technique presented in previous sections. This approach was also aimed to reduce the rendering time by proposing a data structure for traversing only the opaque part of a regular dataset. We will show how the opacity information of the DTF can be used to reduce the size of the rendered dataset by means of multiresolution techniques in Chapter 6.

Interval Volume A step in the direction of extending the isosurface concept was given by Fujishiro et al. [46]. Instead of a single isosurface, they propose to extract an *Interval Volume* $IV(\alpha, \beta)$, that is the polyhedral representation of the portion of the dataset with field values in the interval $[\alpha, \beta]$. The algorithm proposed is based on the extension of the Marching Cube algorithm [69]. For each cell, they separately retrieve the polyhedral representation of the portion of the cell below β and above α by means of an extended MC look-up table. These polyhedral blocks, one for each cell, are then merged togheter in a postprocessing phase and the face shared between blocks deleted.

Interval Set Another step in the search of an unified approach between DVR and isosurface rendering was the *Interval Set* concept proposed by

Guo [51]. His approach is to segment the volume into interval sets, similar to the interval volumes cited above, and render them as surfaces or directly as semitransparent clouds. The technique proposed to effectively extract an interval set from a regular dataset is based on the construction of the alpha shapes [40]. An α-shape complex is built over a point set which is the union of the dataset points with field value belonging to the interval and the isosurface vertices, and with a radius α equal to one half of the diagonal of a cubic cell. It should be noted that this approach, beside its high computational cost[4], does not ensure that the boundary of the interval set matches the original isosurfaces (it is possible to construct examples in which the isosurfaces has small features that cannot be captured by the α-shapes, so the reconstructed volume is an approximation of the desired interval set).

A third contribution has recentely be added by Nielson and Sung [77]. They propose an algorithm for computing a tetrahedralization of interval volume that it better than the one of Fujishiro et al.; their approach is based, similarly to our, on the use of the isosurface class to subdivide the the tetrahedron along the isosurface, but their approach is oriented to a preprocessing phase rather than an on the fly decomposition, does not handle degenerate situations, and it is designed for surface rendering rather DVR, so the sorting problem is not taken into account.

These two last approaches, Interval Volumes and Interval Sets, can be correctly and efficiently modeled by using DTF's. However their limit is that they focus their interest on a single interval of the field domain, trying to extract information about this set. The DTF model manages all these situations in a broader way, permitting the precise coloring of the intervals of the: infact it is easy to think of DTF's not manageable with previous approaches (like sawtooth transfer functions). See the figure 7.4 on page 136 for an example of a DTF that is not an interval volume.

A very recent technical report of Max et al. [115] describes a technique to accurately render unstructured volume data that is very similar to the one presented here: decompose cells crossed by isosurfaces and discontinuities into smaller tetrahedra after the sorting phase. The high level objectives of their work is to obtain the highest accuracy in rendering (infact they rely on a very accurate software scan conversion of each projected cell), while our aim is to obtain the best quality with the smallest overhead. One of the main differences between our approach and the one presented in [115] is that we exploit precomputed tables to recover both the decomposition and

[4]The Delaunay triangulation and α-shape generation can cost $O(n^2)$, with n number of vertex complex contained in the specified interval.

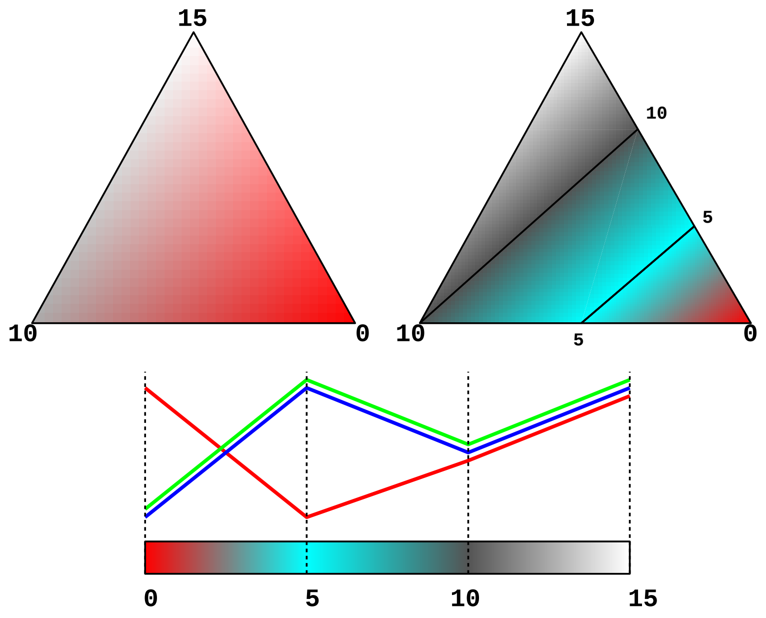


Figure 4.5: Aliasing due to the application the transfer function only to the vertex of a triangle.

depth ordering of the splitted cells in order to reduce the computation at projection time.

DTF and Optical Models In order to better exploit the possibility of DTF's, the volume shading model commmonly used, the density emitter model, must be enhanced. In particular, a non-realistic shading effect can be added to improve the shape comprehension of fuzzy surfaces defined by spike opacity in the DTF. To reach this result we follow the approach used in ray tracing DVR [66, 37] to add a diffuse shading contribution on the regions with high opacity variations, or proportional to the gradient of the opacity.

4.3.1 DTF Rendering

Transfer function transforms the dataset in a colored transparent geometric object and is part of the *modeling* part of the volume visualization process. This process is done with different accuracy depending on the DVR technique used: in ray tracing the TF is applied to all the points collected by the traversing ray in the volume, in projective and most of scan-line approaches the TF is applied only to the vertices of the dataset and then the result-

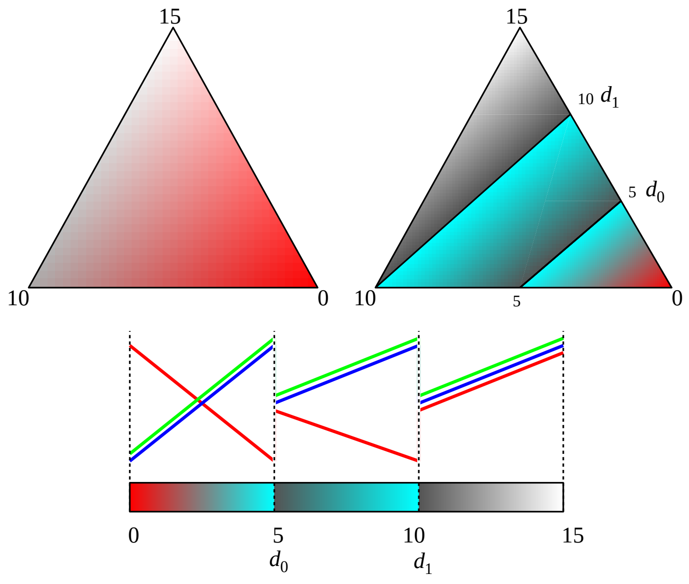


Figure 4.6: Aliasing due to the application of a transfer function with two C^0 discontinuity points d_0 and d_1 to the vertex of a triangle.

ing visual parameters are linearly interpolated across the space spanned by each tetrahedron. This linear interpolation of the visual attributes inside a tetrahedron can create some aliasing effects when the TF is not linear in the field range spanned by the tetrahedron. In figure 4.5 we show an example of this problem in the two-dimensional case. This aliasing problem is rarely considered in current rendering solutions. We propose the use of the splitting techniques developed in Section 4.2 to correctly render a DTF.In a DTF $\mathcal{T}$ the C^0 and C^1 discontinuities, can managed in the following way.

C^0 discontinuities are managed just like classical isosurfaces: we first extract the isosurface with threshold δ equal to the discontinuity value d_i; during the rendering we process each tetrahedron with the split-by-isosurface procedure explained in Section 4.2; when rendering each block b, bounded by two isosurfaces I_i, I_j generated by two discontinuity points $d_i < d_j$ of $\mathcal{T}$, we assign to the vertices of b belonging to I_i the color c_i^+ and to the one belonging to I_j the color c_j^-. The isosurface I_i is colored with c_i.

C^1 discontinuities appears where the DTF change its linear behaviour and can be managed in various ways. The simplest and most common approach is to ignore them with the possible aliasing effects shown in figure 4.5. A more sophisticated approach, presented by Max et al. [97], propose, during the rendering to calculate exactly the color integration only on the thickest point of the projected tetrahedron and linearly interpolate elsewere. Our splitting technique could be used to decompose each tetrahedron along the isosurfaces corresponding to the field values where $\mathcal{T}$ present C^1 discontinuity. While this approach guarantees the correctness of the rendering, an excessive number of tetrahedra could result from the splitting, and this could slow down the rendering process.

It is important to estimate and manage the error, hereafter named *color error* that we introduce calculating the DFT values only on the vertices and linearly interpolating these color values inside each tetrahedron σ. Consider a tetrahedron σ_i with vertices v_0, v_1, v_2, v_3, and let f_i be the linear function intepolating the field value inside σ_i. If we compute the colors of TF only onto the vertices of σ_i and we interpolate the result inside σ_i, we can define:

$$f_i^{\mathcal{T}} : \mathbb{R}^3 \to \mathcal{C}$$

as the function interpolating inside each tetrahedron the colors $\mathcal{T}(f_i(v_0))$ $\mathcal{T}(f_i(v_1))$ $\mathcal{T}(f_i(v_2))$ $\mathcal{T}(f_i(v_3))$ calculated on the field values on the vertices of σ_i.

Let $|| \cdot ||^{\mathcal{C}}$ be a suitable norm on the rgbα space, then the color error committed on point v can be denoted by the function:

$$Q(v) = ||\mathcal{T}(f_i(v)) - f_i^{\mathcal{C}}(v)||^{\mathcal{C}}$$

Obviously, Q is always null on the vertices of tetrahedra. For each tetrahedron σ it is easy to find, in a preprocessing step, the maximum error inside σ, hereafter denoted with a little abuse of notation with $Q(\sigma)$. If a tetrahedron σ is not crossed by any C^1 discontinuity of $\mathcal{T}$ then $Q(\sigma)$ is null; otherwise $Q(v)$ takes its maximum value on one of the C^1 discontinuities crossing σ. To take into account the fact that errors on opaque points are more visible than errors on transparent ones, the resulting error can be weighted with the maximum α value of $\mathcal{T}(f_i(v))$ and $f_i^{\mathcal{C}}(v)$.

Once we are able to measure the committed color error for each tetrahedron we can fix a threshold ϵ and, for each tetrahedron σ with $Q(\sigma) > \epsilon$, extract and store the isosurface passing through the C^1 discontinuity value that generate the maximum error inside σ, so that at rendering time we can decompose, and therefore correctly render, only the tetrahedra with greater error. Such an approach permits also to estabilish an a priori maximum percentage/number of tetrahedra that can be correctly rendered throug decomposition, in order to precisely bound the overhead due to the splitting in rendering time.

4.3.2 Experimental Results

In figure 7.4 and 7.5 on page 136 we show two examples of the wrong and correct rendering of a DTF for the buckyball dataset. In the lower part of Figure 7.4 the DTF applied is shown. On the left we show the result of a simple projective rendering with the DTF applied only to the vertices of the tetrahedra. On the right we show the result of splitting the tetrahedra along the C^0. The buckyball dataset, shown in these figures, is composed of 176,687 tetrahedra. The tetrahedra containing C^0 discontinuities, and therefore incorrectly rendered, were 48,439. The splitting of these tetrahedra along the disconituities has brought the total number of tetrahedra to 360,200. The rendering time, like the case of correct rendering of isosurfaces and DVR, depends only on the total number or tetrahedra and is therefore roughly doubled.

4.4 Conclusions

In this chapter we have presented two main results. The first one is the splitting technique, that allows to integrate correctly and in a efficient manner isosurfaces with the direct volume rendering through tetrahedra projection. The second contribution is the new concept of Discountinuous Transfer Function, that allows the unified management of isosurface, interval volumes and direct volume rendering in a unique framework. The splitting technique presented in the first part of this chapter is then used to correctly render a DTF.

It should be remarked that the modeling/rendering framework, introduced in Chapter 2 for Visualization, led us to a better focusing of the problem faced in this Chapter: we have assumed the tetrahedron with linear interpolation of per-vertex color attributes as the basic rendering primitive for volume visualization. Then the problem of correctly render a DTF is a *modeling* one: we don't need new rendering primitives for each new visualization technique, but we should find visualization *modeling* strategies such that we can *correctly* transform the dataset in, for example, tetrahedra with linear interpolation of per-vertex color attributes.

Chapter 5

Size Reduction of Tetrahedral Meshes

Very often datasets are so large that they cannot be rendered interactively. Simplification techniques can build smaller datasets ensuring a limited/controlled degradation in the represented data. Two original simplification algorithms for tetrahedral meshes are described in detail.

The real usability of a system for the visualization of volume data is strictly connected to the level of interactivity the system performs. This is because the user–system interaction is enhanced and the understanding of the results is improved through motion and interactive modifying of the visualization parameters (e.g. transfer function, isosurface threshold). The efficiency of the visualization algorithm is therefore crucial.

Direct projection of tetrahedral cells, using the hardware capabilities of current state-of-the-art graphics workstations, is an efficient process (nearly of the order of $10K \div 100K$ tetrahedral cells per second). Nevertheless, the performance required for the interactive use of these techniques is still far beyond current speeds, especially in the case of low or medium power workstations.

A data simplification approach can be applied to produce significant speedups while maintaining good approximations in the images produced. Therefore, we prove William's intuition [112] that real-time interactive projection can be only obtained through data reduction.

The Chapter has the following structure: in section 5.1 we survey techniques to perform data simplification of tetrahedral volume datasets, in Section 5.2 we introduce the notation used in Section 5.3 and 5.4 to describe two original simplification algorithms; finally in Section 5.5 the empirical results of the application of the first algorithm are presented .

5.1 Related Work

The main approach to build an approximate representation of a tetrahedral dataset is choosing a subset of the original vertices and building a new triangulation of (almost) the same domain.

Many different adaptive methods, which try to select the smallest set of points approximating a dataset within a given error, have been developed in 2D for the simplification of irregular meshes and topographic surfaces; a detailed review of these algorithm is beyond the scope of this chapter, for a complete survey on this subject see [26].

Very concisely we can summarize by saying that effective solutions to the simplification problem have been obtained through incremental techniques, based on what we can call either *refinement* (refine a coarse representation by adding points [44, 32]) or *decimation* (simplify the dataset by removing points [89, 18, 9]) strategies. Most of these techniques can be extended to the 3D case to simplify volume data, but only few experiments have been carried out [21, 52]. In the following we review the specific results regarding tetrahedral meshes.

A first attempt in this direction was proposed by Williams in [114]; he suggest to choose a random subset of the vertices of the mesh and retriangulate them using a Delaunay triangulation conformed in order to approximate the original domain. This proposal was neither implemented nor specified in details and presents two serious drawbacks: there is no control on the accuracy of the simplified mesh and the technique is not adaptive, i.e. the density of the data cannot vary over different regions of the domain.

A more detailed description of a very similar approach is given by Renze and Oliver in [85]; they propose a volume decimation algorithm in which, given a volume dataset described by a tetrahedral complex Σ, they try to remove, without any specified order, the internal vertices of the mesh; the retriangulation of the hole left by the removal of a vertex v is done by building the Delaunay triangulation Σ_v of the vertices adjacent to v, and searching, if it exists, a subset of the tetrahedra of Σ_v whose (d-1)-faces match with the faces of Σ. If such a subset does not exists the vertex is not removed; it should be noted that such condition may very ofter occur if the original complex is not a Delaunay one. The Renze and Oliver's approach, as the idea sketched by Williams, neither measures the approximation error introduced in the reduced dataset, nor tries to select the vertex subset in order to minimize this error.

In [52], Hamann and Chen introduce a refinement strategy for the simplification of tetrahedral convex complexes. Their method is based on the selection of most important points and Their insertion into the convex hull of the domain of the dataset. Significant data are identified by large absolute curvatures obtained by a local least square approximation method. When a point is inserted into the triangulation, local modifications (by face/edge swapping) are performed in order to minimize a local approximation error. This process leads to a data dependent triangulation.

In a recent paper Popovic and Hoppe [81] have extended the Progressive Meshes algorithm [54], a simplification strategy for three-dimensional surfaces based on edge-collapse operations, to generic simplicial complexes. However their approach is more oriented towards the simplification and management of surfaces in the most comprehensive way, than towards the topology preserving simplification of tetrahedral complexes for scientific visualization.

5.2 Approximated meshes

Let V be a volume dataset, and let Γ be a given mesh over V, covering a domain Ω, and having all points of V as vertices. The pair (V, Γ) is called a *reference model* for the volume dataset. An *approximated model* of such volume data is given by a pair (V', Σ), with Σ a tetrahedral mesh having as vertices the points in $V' \subset V$, and covering a domain $\tilde{\Omega}$ that approximates Ω. A linear function is given for each tetrahedron of Σ to interpolate the field inside *sigma*. The *accuracy* of approximation is given by the difference between the reference model and the approximated model, and it depends essentially on:

- the *warping* of the domain, i.e., the difference between Ω and its approximation $\tilde{\Omega}$;

- the *error* made in approximating values at the points of V through the piecewise-linear function defined on Σ.

Hereafter we will denote the accuracy of an approximation with the pair $\mu = (\delta, \varepsilon)$ where δ and ϵ denote the warping and the error of the approximated dataset, respectively.

Some considerations on the computation of warping and error can be made for the following classes of volume datasets.

Convex Datasets. This is the simplest case: we can assume that $\tilde{\Omega} \equiv \Omega$, i.e., there is no warping error because convex datasets usually have a small number of vertices on their convex hull; in particular, regular datasets have

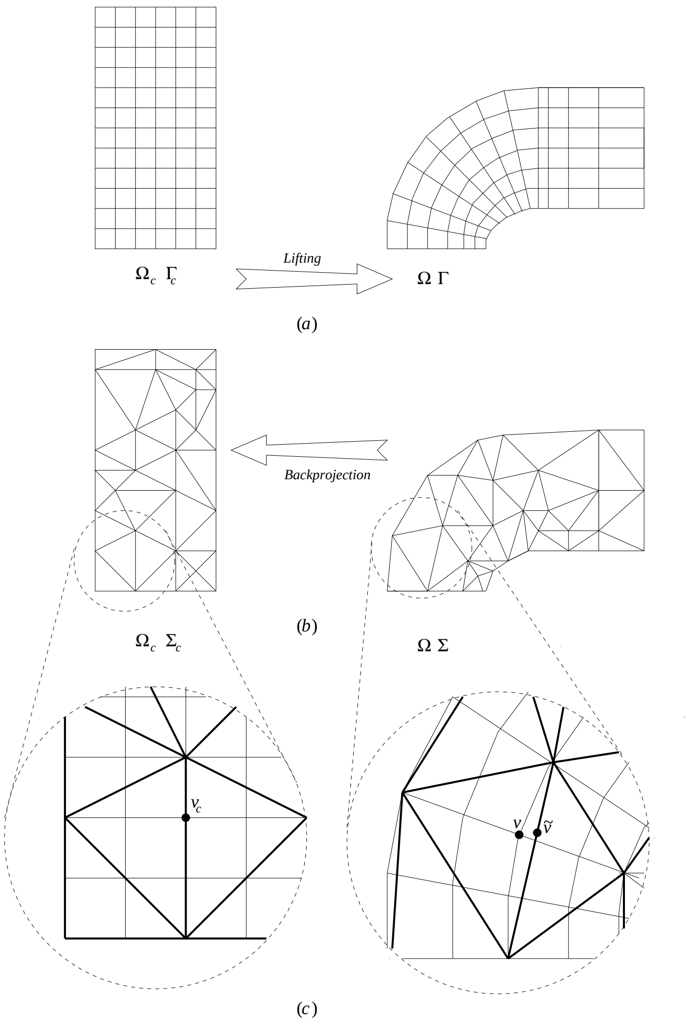


Figure 5.1: Lifting and warping for curvilinear datasets (example in 2D): (a) the lifting maps a regular mesh Γ_c into a curvilinear mesh Γ; (b) the triangular mesh Σ approximating Γ is back-projected in computational space into mesh Σ_c; (c) the warping at a point v is equal to the distance from v to the warped point $\tilde{v}$.

a convex hexahedral domain that can be defined with just six vertices. In a *convex* dataset, the error on a point v contained in a tetrahedron σ is given by the absolute value of the difference between the field value at v, and the value of the linear function associated to σ computed at v.

Non-convex curvilinear datasets. This class of datasets is a common result of simulations: the domain is represented by a deformed hexahedral lattice. We consider a parallelepiped Ω_c, that we call the *computational domain*, and a regular hexahedral mesh Γ_c covering Ω_c, and isomorphic to Γ. The one-to-one correspondence (isomorphism) between vertices of Γ_c and Γ will be called a *lifting* from computational to physical domain (see Figure 5.1a). Since Σ has vertices on a subset of vertices of Γ, we can use lifting to back-project Σ into a corresponding tetrahedral mesh Σ_c in computational domain (see Figure 5.1b). Meshes Γ_c and Σ_c both cover Ω_c, provided that Σ_c has at least the eight corners of Ω_c as vertices. Therefore, each vertex v_c of Γ_c is contained into some tetrahedron σ_c of Σ_c. We express the position of v_c in baricentric coordinates with respect to σ_c, and we consider the point $\tilde{v}$ in physical space having the same baricentric coordinates as v_c with respect to the tetrahedron σ, image of σ_c in the physical space. Point $\tilde{v}$ is called the *warped image* of v (where v is the image of v_c through lifting). The warping at v is the distance between v and $\tilde{v}$ (see Figure 5.1c). The maximum distance over all the vertices of Γ whose back-projection lies inside σ_c provides an estimation of the warping of its lifted image σ; the maximum warping over all tetrahedra of Σ defines the warping of the whole approximated model.

The field value error E is measured by computing the difference between the field value at v, and the value of the linear function in the computational domain: this is equivalent to measure the difference between the field at a datum v and the estimated value at its corresponding warped point $\tilde{v}$ defined above.

Non-convex irregular datasets. This is the most general case; for the estimation of the warping errors we follow the approach used in the field of the simplification of 3D surfaces and we estimate the actual difference between the boundaries of Ω and $\tilde{\Omega}$. Such a difference is measured by computing the Hausdorff distance between the two domains, defined as follows: the Euclidean distance between a point p and a set $P \subset \mathbb{E}^{\mathrm{d}}$ is defined by

$$d(p, P) = \min_{x \in P} d(p, x)$$

where $d()$ is the Euclidean distance between two points in $\mathbb{E}^{\mathrm{d}}$. The one-sided Hausdorff distance $d_E(P, Q)$ from a set $P \subset \mathbb{E}^{\mathrm{d}}$ to a set $Q \subset \mathbb{E}^{\mathrm{d}}$ is defined

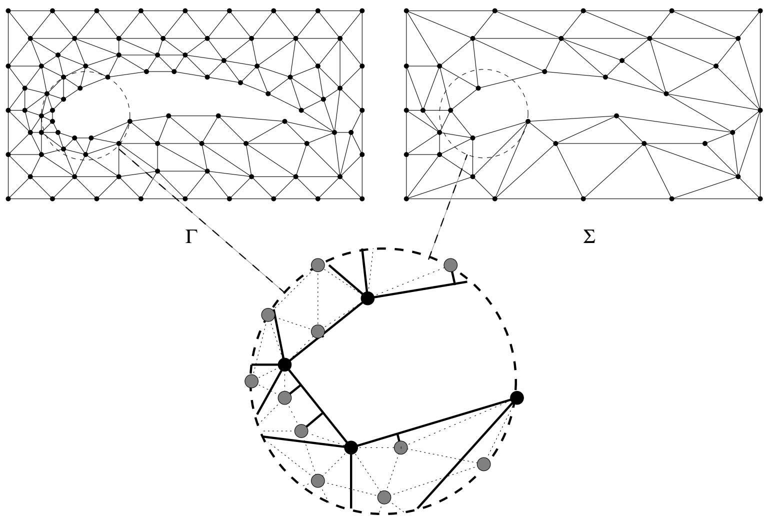


Figure 5.2: For *non-convex irregular* datasets, we estimate the actual difference between the boundaries by computing at each boundary vertex of Γ its minimum distance from the boundary of Σ.

by

$$d(P, Q) = \max_{x \in P} d(p, Q)$$

An approximation of this distance can be computed as follows.The warping of a boundary face σ of Σ is the maximum among all the distances corresponding to the boundary vertices of Γ that are projected onto σ; the warping of Σ is the maximum warping of all its boundary faces [18].

To estimate the error in approximating the field on a *non-convex irregular* datasets we need to consider two possible cases: if v is inside $\tilde{\Omega}$, then we compute the field difference as in the convex case; if v lies outside $\tilde{\Omega}$, we compute first the projection v_p of v on the boundary of $\tilde{\Omega}$, then we measure the difference between the field at v and the linear interpolation at v_p. In this case, v is said *related to* the tetrahedron σ having v_p on its boundary (see Figure 5.2).
The error of a tetrahedron σ is the maximum error of all the vertices v_i such that: for the convex case, v_i lies inside σ; for the non-convex curvilinear case, the point corresponding to v_i in computational space lies inside σ_c; v_i is either inside σ, or related to σ. The error of the mesh Σ is the maximum among all errors of its tetrahedra.

Hereafter, warping and error will be denoted by functions $W()$ and $E()$, respectively; they can be evaluated at a point v, at a tetrahedron σ, or at a mesh Σ. Warping and error at data points can also be weighted by suitable functions that may vary over Ω. Weights can be useful to obtain a *space-based* measure of accuracy. For example, if we assume that for applicative needs accuracy is important in the proximity of a selected point p, then we can select weights decreasing with distance from p. Similarly, *range-based error* can be used to require more accuracy where data assume a given value q: in this case, a weight for error can be obtained by composing the value function ψ with a real univariate function decreasing with distance from q.

5.2.1 Building an approximated model

Given a reference model (V, Γ), and a threshold pair $\mu = (\delta, \varepsilon)$, we deal with the problem of building an approximated model (V', Σ) that represents the volume dataset with accuracy μ, i.e. with a warping smaller than δ and an error smaller than ε. A key issue is that the size of Σ should be as small as possible. A result in 2D [3] suggests that the problem of minimising the size of the mesh for a given accuracy is intractable (NP-hard); moreover, approximated algorithms that guarantee a bound on the size of the solution with respect to the optimal one are difficult to find, and hardly applicable in practice [3]. Hence, heuristics can be adopted to obtain a mesh of reduced size by following data simplification strategies. There are two basic classes of strategies for simplifying a mesh:

- *Refinement strategies* start from a mesh whose vertices are a very small subset of vertices of Γ. The mesh is iteratively refined by inserting other vertices of Γ into it. Refinement continues until the accuracy of the mesh satisfies the required threshold. The vertex to be inserted can be selected on the basis of the best improvement of the mesh accuracy.

- *Decimation strategies* start from the reference model Γ and iteratively modify it by eliminating vertices. As many vertices as possible are discarded, while maintaining the required accuracy. Also in this case, points are selected at each iteration in order to cause the least possible increase in warping and error.

In the following Sections, we present two algorithms for the simplification of a tetrahedral mesh: the first method is based on refinement and Delaunay tetrahedralization; it can be applied to convex datasets, and to non-convex curvilinear datasets; the second method is based on decimation, and it can be applied to any dataset, provided that the reference mesh Γ is a tetrahedral mesh, but it is especially well suited to non-convex irregular meshes.

5.3 A method based on refinement

In [19] we have proposed the extension of an existing refinement for convex datasets and we have further extended it in [21] to deal also with non-convex curvilinear datasets. In this section we describe the basic technique and our proposed extention. The basic idea comes from an early technique developed in the two-dimensional case and widely used for approximating natural terrains [44, 33]. An on-line algorithm for Delaunay tetrahedralization is used together with a selection criterion to refine an existing Delaunay mesh by inserting one vertex at a time. In the case of curvilinear datasets, a Delaunay tetrahedralization is computed in the computational domain, while its image through lifting gives the corresponding mesh in the physical domain. In both cases, the selection strategy at each iteration is aimed to refine the tetrahedron that causes the maximum warping/error in the current approximation: this is obtained by selecting the datum v_{max} corresponding to the maximum warping/error as a new vertex. The description of the algorithm is general, while specific aspects of either the convex or the curvilinear case are explained when necessary.

Given a dataset V, an initial mesh Σ is created. If V is a convex dataset, then Σ is a tetrahedralization of the convex hull of V. If V is a non-convex curvilinear dataset, then a tetrahedralization Σ_c of the computational domain Ω_c is considered: since Ω_c is a block, Σ_c has only the eight corners of Ω_c as vertices, and it subdivides Ω_c into five tetrahedra; Σ is obtained by lifting Σ_c into the physical domain. Given a threshold μ for the accuracy, the usual refinement strategy [33] is applied:

```
procedure REFINEMENT($V, \Sigma, \mu$);
    while not ($\Sigma$ satisfies $\mu$) do
        $v_{max} \leftarrow$ SELECT_POINT($V, \Sigma, \mu$);
        $\Sigma \leftarrow$ ADD_VERTEX($\Sigma, v_{max}$)
    end while ;
    return ($\Sigma$)
end ;
```

This refinement procedure always converges since the number of points in V is finite; total accuracy is warranted when all of them are inserted as vertices of Σ. In summary, three tasks are accomplished at each iteration of the refinement procedure:

1. test the accuracy of Σ against μ: this requires evaluating $E(\Sigma)$ and, in the curvilinear case, $W(\Sigma)$, and comparing them with ε and δ, respectively;

2. select a new vertex v_{max} from the points of V by means of the procedure SELECT_POINT: for the convex case, the point of V that

maximises $E()$ is selected; for the curvilinear case, the point of V that either maximises $W()$ or maximises $E()$ is selected, depending on whether $W(\Sigma)/E(\Sigma)$ is larger or smaller than δ/ε.

3. update Σ by inserting v_{max} by ADD_VERTEX: this is done by using an algorithm for on-line Delaunay triangulation that was proposed in [57]: in the curvilinear case, update is always made on the tetrahedral mesh in computational domain, and Σ is obtained through lifting.

In order to implement the algorithm, we have used a data structure that can achieve efficiency, while remaining as simple as possible. A tetrahedral mesh is encoded as a set of vertices plus a set of tetrahedra. For each vertex v, only its coordinates and its field value are stored; in the case of curvilinear datasets, the coordinates of v are maintained both in computational and in physical domain. For each tetrahedron σ, four pointers to its vertices, and four pointers to its adjacent tetrahedra are maintained.

The relations among tetrahedra and points of V that are not vertices of Σ are maintained by means of a bucketing technique similar to that proposed in [59, 33] for dynamic triangulation in 2D: for each tetrahedron σ we maintain a list of data points of V it contains; for the curvilinear case, containment is intended in computational domain. For every such point, also its accuracy values (error and warping) are stored; the vertices with the maximum error and maximum warping are stored at the head of the list. This data structure is initialised by locating each point of V with respect to the tetrahedra of the initial mesh Σ. Then, each time Σ is updated, all points that lie within the modified volume are relocated with respect to the new tetrahedra generated during updating.

Tetrahedra of Σ are stored in a priority queue that supports efficient retrieval of tetrahedra maximising error and warping, hence evaluation of $E(\Sigma)$ and $W(\Sigma)$, respectively. Such a query actually provides also the point v_{max} corresponding to the maximum error or warping: therefore, point selection is obtained as a side effect of the test of accuracy.

The procedure ADD_VERTEX updates the tetrahedral mesh, while maintaining the whole data structure consistent. The on-line algorithm proposed in [57] updates the mesh at each vertex insertion by using a sequence of actions called *face flips*. A face flip modifies the mesh only locally: each face flip replaces one, two, or three tetrahedra with four, three, or two new tetrahedra, respectively (see Figure 5.3).

Beside modifying the mesh, we must also update the bucketing structure, and the priority queue after each face flip. The following operations must be performed:

- eliminate from the priority queue all tetrahedra replaced by face flips;

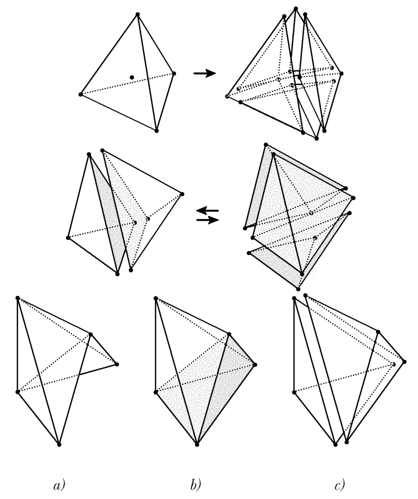


Figure 5.3: A tetrahedra *split*, due to a new vertex selection and insertion in the mesh (top image); the two classes of tetrahedra *flip* actions: the 2_*to*_3 flip, which produces three cells out of two (center image); the 3_*to*_2 flip, needed when the two tetrahedra present a non convex union (a), and a third cell (b) has to be included in the flip action.

- relocate all data points that were contained inside such tetrahedra with respect to the new tetrahedra;
- insert the new tetrahedra into the priority queue.

Relocation of points is simply done by scanning the lists of points attached to the "old" tetrahedra, and, for each point v in a list, deciding which of the "new" tetrahedra contains v. Dynamic update of the priority queue is performed efficiently by standard methods.

A further remark is necessary, though, about the case of curvilinear datasets. During the initial stages of refinement, mesh Σ might result geometrically inconsistent because of the warping caused by lifting. Indeed, while mesh Σ_c is a Delaunay tetrahedralization of the computational domain, hence consistent, some tetrahedra might "flip over" during lifting, hence changing their orientation and causing geometric inconsistencies in Σ. See Figure 5.4 for a two-dimensional example. Consistency can be tested by verifying whether each tetrahedron maintains its orientation both in computational and in physical domain.
We assign infinite warping to each tetrahedron presenting an inconsistent lifting. In this way, inconsistent tetrahedra are refined first, and the mesh rapidly converges to a consistent one.

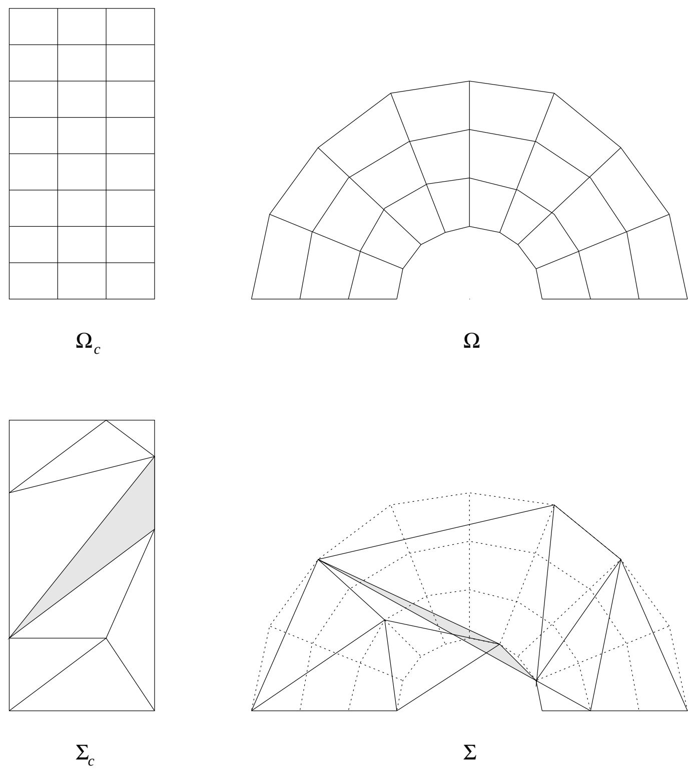


Figure 5.4: Inconsistency in curvilinear mesh (2D example): mesh Σ_c is geometrically consistent, while its lifted image Σ is not.

Complexity The time complexity of the refinement procedure is not crucial to our application, as long as it remains within reasonable bounds, because the algorithm is applied off-line to the volume dataset in order to build a multiresolution model (see Chapter 6). However, time analysis when all the n points of V have to be inserted into Σ shows a bound of $O(n^3)$ in the worst case [19], while experiments show a subquadratic behaviour in practice. On the other hand, the space occupancy of this algorithm is quite high, due to the need to maintain both a bucketing structure and a priority queue (see empirical evaluations in Section 5.5, Tables 5.1 and 5.2).

5.3.1 Refinement of large datasets by block-decomposition

For datasets having a regular structure (either in physical or in computational domain) it is possible to bring the size of the structure into more manageable bounds, by splitting the dataset into blocks, and running the algorithm separately on each block. Assume, for instance, that a regular

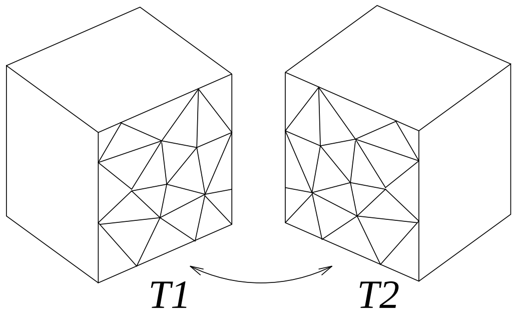


Figure 5.5: Two adjacent blocks Σ_1 and Σ_2, and the coincident triangulations T_1 and T_2 of their common face.

dataset of size $m \times n \times p$ is given: we can subdivide it, e.g., into k^3 blocks of size $(m/k+1) \times (n/k+1) \times (p/k+1)$ and process them separately, with the same threshold μ in all cases. Then, the resulting meshes are joined to form a mesh of the whole domain.

In order to warrant the correctness of such a procedure, we must be sure that the structure obtained by joining all results is a tetrahedralization of the whole domain. A similar approach is presented in the two dimensional case in [33]. This can be proved by showing that given two blocks sharing a common face, the refinement algorithm will triangulate such a face in the same way while refining each block (see Figure 5.5). Let Σ_1 and Σ_2 be the meshes of the two blocks, and let T_1 and T_2 be the triangulations of the face r common to both blocks in Σ_1 and Σ_2, respectively. We may assume that, upon suitable initialization of the meshes, T_1 and T_2 are initially coincident. Let us consider a generic step of the algorithm that refines Σ_1: if the inserted vertex does not lie on r, the update change will not change either T_1 or the error/warping of data points lying on r; on the contrary, if the vertex inserted lies on r, it must be the point which maximizes error/warping among all data points lying on r. This means that the sequence of vertices refining T_1 is independent of the refinement that occurs in the rest of Σ_1. Since the same situation occurs for the refinement of Σ_2, we can conclude that the same sequence of vertices will be selected for T_2, hence the two triangulations for a given accuracy will be coincident. However it should be noted that the result will not be the same that we would obtain by running the refinement algorithm on the whole dataset, since the resulting tetrahedralization might not be globally Delaunay: the Delaunay property is verified only locally to each block.

5.4 A method based on decimation

The refinement method described above is difficult to adapt to the case of non-convex irregular datasets. Major difficulties arise in finding an initial coarse mesh to approximate the domain Ω and in the estimation of warping. Delaunay triangulation is not applicable to non-convex polyhedra; moreover even if we have an approximation of the boundary of the starting domain finding a tetrahedralization of this polyhedron, without adding new points, is an NP-complete problem [87].

Experience in the approximation of non-convex surfaces through 2D triangular meshes suggests that a decimation technique might be more appropriate to the case of non-convex irregular datasets (see, for example, [89, 54, 18]). In the following, we describe an algorithm that extends such heuristics to volume data: starting from the reference mesh Γ, vertices are iteratively discarded as long as it is possible. Given a threshold μ for the accuracy, the following decimation procedure is applied:

```
procedure DECIMATION($V, \Gamma, \mu$);
    $\Sigma \leftarrow \Gamma$;
    while $\Sigma$ satisfies $\mu$ do
        $v_{min} \leftarrow$ SELECT_MIN_VERTEX($V, \Sigma, \mu$);
        $\Sigma \leftarrow$ REMOVE_VERTEX($\Sigma, v_{min}$)
    end while ;
    return ($\Sigma$)
end ;
```

The two procedures SELECT_MIN_VERTEX and REMOVE_VERTEX select the vertex to be removed and effectively remove it, respectively. They are somehow more delicate than their respective counterparts in the refinement approach SELECT_MAX_POINT and ADD_VERTEX. In the following subsection we give some details about them.

5.4.1 Selecting a vertex to be removed

Selecting a vertex to be removed involves an estimation of the amount of error and warping due to the removal: the criterion adopted is that the vertex causing the smallest increase in error/warping should be selected at each iteration. An exact estimation of the change in error and warping can be obtained by simulating deletion of all vertices in the current mesh. This would be computationally expensive, since each vertex has 24 incident tetrahedra on average. This may involve relocating many points lying inside such tetrahedra. We prefer to use heuristics to estimate apriori how a vertex removal affects error and warping. Such an estimation is computed for all vertices before decimation starts, and it is updated for a vertex each time

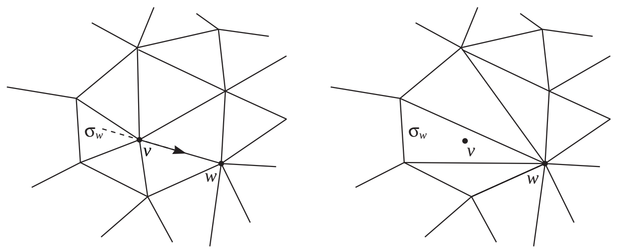


Figure 5.6: Estimating the error due to the collapsing of v on w.

one or more of its incident tetrahedra change.

In order to estimate error increase, we pre-compute the field gradient ∇_v at each vertex v of the reference model: this can be done by calculating the weighted average of gradients in all tetrahedra incident at v. The weight for the contribution of a tetrahedron σ is given by the solid angle of σ at v. Then, for each vertex v in the mesh, we search the vertex w, among those adjacent to v, such that the difference $\Delta\nabla_{v,w}$ between ∇_v and ∇_w is minimum. Value $\Delta\nabla_{v,w}$ gives a rough estimate of how far from linear is the field in the neighbourhood of v: the smaller $\Delta\nabla_{v,w}$, the smaller the expected error increase if v is removed. Value $\Delta\nabla_{v,w}$, and a pointer to w are stored together with v.

Another estimation can be given by the *local* error introduced with the collapsing of v on w. Let σ_w be the tetrahedron containing the removed vertex v after the collapsing. We note that σ_w is tetrahedron incident on v and containing the extension of the edge (v, w) on the v side. Once determined σ_w we evaluate the error as the difference between the field value on v and the field obtained by interpolating inside σ_w on v position. Figure 5.6 shows the tetrahedron σ_w before and after the edge collapsing operation and, as a dashed line, the extension of the edge (v, w).

Warping changes only if a vertex lying on the boundary of Σ is removed. Therefore, for each boundary vertex v, we estimate apriori warping increase caused by removing v on the basis of the local geometry of the boundary of Σ in the neighbourhood of v. We adopt a criterion proposed in [89]; it is essentially based on the distance d_v between v and a plane that best fits all vertices lying around v on the boundary of Σ (see Figure 5.7): the smaller d_v, the smaller the expected warping increase if v is removed. Therefore, d_v is stored together with v.
Vertices of Σ are maintained in a priority queue supporting efficient selection. In this framework, the selection criterion adopted in procedure SELECT_MIN_VERTEX is symmetrical to the one used in the refinement algorithm: we select the vertex of Σ which is expected to produce the smallest

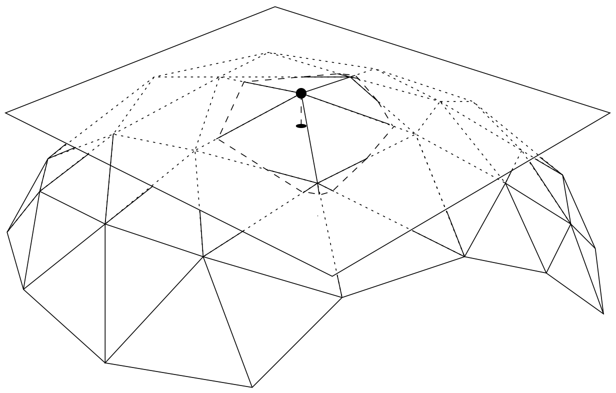


Figure 5.7: An apriori estimate of warping increase caused by removing a boundary vertex v is obtained by measuring the distance of v from an average plane fitting its adjacent vertices on the boundary of Σ.

increase in either warping or error, depending on whether $W(\Sigma)/E(\Sigma)$ is larger or smaller than δ/ε.

5.4.2 Removing a vertex

Once a vertex v has been selected, we need to tetrahedralize the polyhedron resulting from the elimination of all the tetrahedra incident on v. Unfortunately the removal of this vertex from the mesh is not always possible: this difficulty is related to the fact that it may be not possible to tetrahedralize a non-convex polyhedron. Since deciding whether this is possible or not is NP-complete, we use heuristics to try to remove a vertex by collapsing one of its incident edges to its other endpoint. Given a vertex v, we try to remove it by collapsing the edge e that joins v to vertex w having the smallest difference $\Delta\nabla_{v,w}$ from v in its surface normal; recall that w had been selected while estimating the cost of removing v in terms of error.
Edge collapse is a simple operation: all tetrahedra incident at e are deleted, while all other tetrahedra that have a vertex at v are modified by moving such a vertex at w. All adjacencies are updated accordingly: if two tetrahedra σ_1 and σ_2 were both adjacent to a tetrahedron σ_0 that is deleted, then σ_1 and σ_2 become mutually adjacent (see Figure 5.8a for an example in 2D).

In order to be correct a collapse must pass some topological and geometric consistency tests. We desire that the edge collapse operation does not topologically modify our complex, that is that it neither it change its genus neither it introduce some not 3-manifold. A collapse is topologically consistent if the following conditions are satisfied:

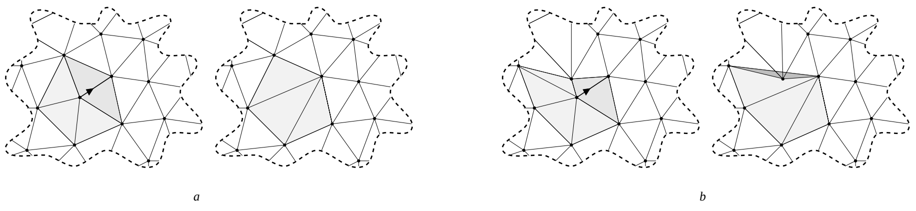


Figure 5.8: Edge collapse in 2D: (a) a valid collapse; (b) an inconsistent collapse.

- if v and w are boundary vertices then the edge (v, w) is a boundary edge;
- if v and w are boundary vertices then for each boundary vertex i adjacent to both v and w, the face (w, v, i) exists and is a boundary facet.
- if v and w are boundary vertices then the complex formed by all the boundary facets reachable through adjacency from v and w has at least 5 vertices.

Geometric consistency of the mesh may be violated if some tetrahedron "flips over", i.e., it changes its orientation, because of edge collapsing (see Figure 5.8b for an example in 2D). Consistency can be tested simply by checking the orientation of each tetrahedron incident at v before and after collapse. If collapse is impossible, then no mesh update occurs and v is temporary tagged as non-removable, by setting its error and warping estimate at infinity. In this way, a different vertex will be selected at the next cycle.

After a successful edge collapse, a precise evaluation of the current accuracy must be obtained. As in the refinement method, we adopt a bucketing structure to maintain the relations between tetrahedra and data points they contain. Updating this structure involves only the portion of mesh covered by the "old" tetrahedra that were adjacent to v. All removed points (including v) that belong to such a volume are relocated with respect to the "new" tetrahedra. Note that, if v is a boundary vertex, some points may fall outside the mesh: such points (including v) are assigned to tetrahedra by considering their projections on the "new" boundary faces of the mesh (see Figure 5.9). Changes in accuracy are computed for each point on the basis of its new location. Finally, the apriori estimate of error and warping increase is recomputed for each vertex that was adjacent to v, and the priority queue is updated accordingly.

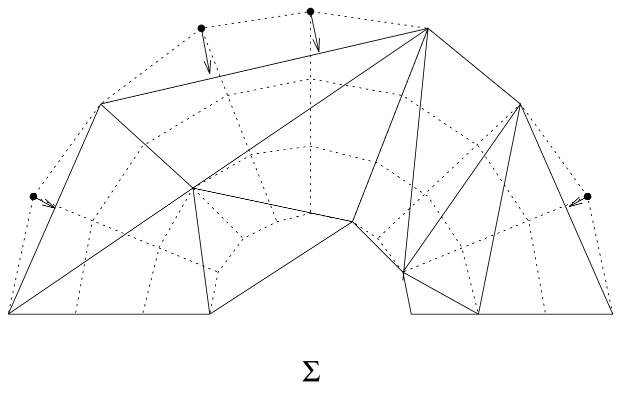


Figure 5.9: Points that fall outside the mesh are assigned to tetrahedra by projecting them on the boundary faces.

Complexity. The complexity of this algorithm can be calculated as follows. The vertex collapse operation has a cost depending on the number of incident tetrahedra on v, that can be, in the worst case, $O(n)$ where n is the number of vertices of the mesh. Similarly, the vertex relocation operation in the bucketing structure, needed for the error evaluation, can cost $O(n)$. The resulting overall worst case complexity of the decimation algorithm is therefore $O(n^2)$.

5.5 Experimental Results

The performance of the refinement based simplification algorithm were evaluated on four datasets, representative of the two classes of regular and non-convex curvilinear datasets. The implementation of the decimation based algorithm was still under development at the time of writing. Datasets commonly used in the volume rendering field were chosen in order to facilitate comparisons with other proposals:

- **BluntFin**, a $40 \times 32 \times 32$ curvilinear dataset, was built by running a fluid-flow simulation of an air flow over a blunt fin and a plate[1];
- **Post**, a $38 \times 76 \times 38$ curvilinear dataset which represents the result of a numerical study of a 3D incompressible flow around multiple posts;

[1]Both BluntFin and Post are produced and distributed by NASA–Ames Research Center.

- **SOD**, a subset $32 \times 32 \times 32$ (not a subsampling) of a regular rectilinear dataset which represents the electron density map of an enzyme[2];
- **BuckyBall**, a $128 \times 128 \times 128$ regular rectilinear dataset which represents the electron density around a molecule of C_{60}. Some experiments are presented on either $32 \times 32 \times 32$ or $64 \times 64 \times 64$ subsampling of such a dataset[3].

Tables 5.1 and 5.2 shows results of the simplification of curvilinear and regular datasets, respectively. Each table reports: computation times required to refine the whole model, maximal RAM space occupancy during construction and some information on a number of approximated meshes extracted from it. The accuracy of each approximation is measured as follows: warping is a percentage of the length of the diagonal of a minimum axis-aligned bounding box containing the dataset, while error is a percentage of the range spanned by data values. Times are CPU seconds on a SGI Indigo workstation (MIPS R4000).

The graph of Figure 5.10 shows the number of vertices of the mesh through refinement, depicted as a function of approximation error. Note how rapidly the size of the mesh decreases when the error increase. These results give a quantitative estimate of the advantage of using approximated representation of volume datasets.

Figure 5.11 shows the spatial distribution of sites of the BluntFin dataset, compared with the spatial distribution of vertices of an approximated model at accuracy $\mu = (2.\%, 2.\%)$

As it can be noted the experiments presented in Table 5.2 for the BuckyBall dataset were run on a subsampling, because of limitations in the available RAM. A multiresolution model of the whole dataset, and of two subsampled datasets, were also obtained by using the *block-decomposition refinement* described in Section 5.3.1. Results are presented in Table 5.3. By adopting this method we can overcome the intrinsic RAM limitations of a specific platform, because for any dataset we can always have a partition such that the refinement of each block becomes a tractable problem with the available resources.

In particular, we can compare the results obtained for the 32^3 subsampled dataset refined as a whole (lower part of Table 5.2) and refined as 64 independent blocks (upper part of Table 5.3). Note that, with the block decomposition refinement, total computation time decreases from 1,318 sec. to 532 sec., while we have only a small increase in the number of vertices

[2]SOD was produced by D. McRee, Scripps Clinic, La Jolla (CA), and kindly distributed by the University of North Carolina at Chapel Hill.

[3]BuckyBall is available courtesy of AVS International Center.

Curvilinear Datasets	no. tetra.	no. vertices	% of vertices
BluntFin (40x32x32)		40,960	
Construction Time: 1,704 sec. RAM = 35,300 Kb			
$\delta \leq 4.0\%$, $\varepsilon \leq 4.0\%$	20,324	3,612	8 %
$\delta \leq 3.0\%$, $\varepsilon \leq 3.0\%$	30,116	5,296	12 %
$\delta \leq 2.0\%$, $\varepsilon \leq 2.0\%$	47,189	8,263	20 %
$\delta \leq 1.0\%$, $\varepsilon \leq 1.0\%$	80,883	14,162	34 %
$\delta \leq 0.5\%$, $\varepsilon \leq 0.5\%$	111,251	19,620	47 %
$\delta \leq 0.2\%$, $\varepsilon \leq 0.2\%$	152,927	27,351	66 %
$\delta \leq 0.1\%$, $\varepsilon \leq 0.1\%$	182,660	32,945	80 %
$\delta \leq 0.0\%$, $\varepsilon \leq 0.0\%$	222,528	40,960	100 %
Post (38x76x38)		109,744	
Construction Time: 7,794 sec. RAM = 95,240 Kb			
$\delta \leq 4.0\%$, $\varepsilon \leq 4.0\%$	47,691	8,282	7 %
$\delta \leq 3.0\%$, $\varepsilon \leq 3.0\%$	76,893	13,177	12 %
$\delta \leq 2.0\%$, $\varepsilon \leq 2.0\%$	121,181	20,773	18 %
$\delta \leq 1.0\%$, $\varepsilon \leq 1.0\%$	193,971	33,681	30 %
$\delta \leq 0.5\%$, $\varepsilon \leq 0.5\%$	277,822	48,418	44 %
$\delta \leq 0.2\%$, $\varepsilon \leq 0.2\%$	395,299	69,568	63 %
$\delta \leq 0.1\%$, $\varepsilon \leq 0.1\%$	490,337	87,085	79 %
$\delta \leq 0.0\%$, $\varepsilon \leq 0.0\%$	609,245	109,744	100%

Table 5.1: Measures on multiresolution models built from curvilinear datasets

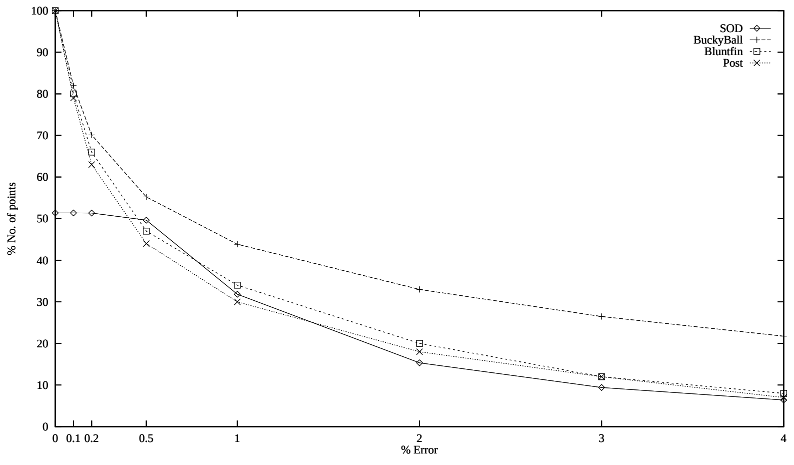


Figure 5.10: Number of points in the simplicial model expressed as a function of the approximation error.

Regular Datasets	no. tetra.	no. vertices	% of vertices
SOD (32x32x32)		32,768	
Construction Time: 1,491 sec. RAM = 45,134 Kb			
Levels of Detail:			
ε =4.0 (%)	11,485	2,094	6 %
ε =3.0 (%)	17,178	3,082	9 %
ε =2.0 (%)	28,521	5,026	15 %
ε =1.0 (%)	59,718	10,443	31 %
ε =0.5 (%)	91,963	16,269	49 %
ε =0.2 (%)	95,314	16,825	51 %
ε =0.1 (%)	95,349	16,831	51 %
ε =0. (%)	95,349	16,831	51 %
BuckyBall (32x32x32)		32,768	
Construction Time: 1,318 sec. RAM = 25,860 Kb			
ε =4.0 (%)	42,468	7,125	21 %
ε =3.0 (%)	51,490	8,680	26 %
ε =2.0 (%)	63,649	10,808	32 %
ε =1.0 (%)	83,667	14,372	48 %
ε =0.5 (%)	104,113	18,090	55 %
ε =0.2 (%)	130,152	22,982	70 %
ε =0.1 (%)	150,249	26,854	81 %
ε =0. (%)	176,687	32,768	100 %

Table 5.2: Measures on multiresolution models built on two regular datasets.

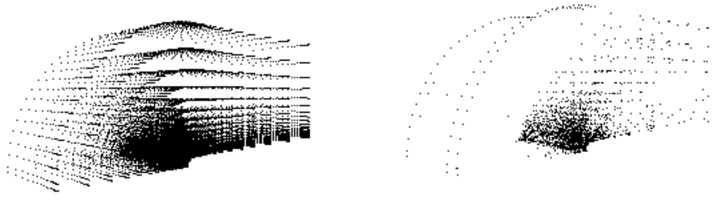


Figure 5.11: Distribution of vertices of the BluntFin dataset: original dataset (40,960 sites) on the left, approximated mesh with $\delta \leq 2\%$ and $\varepsilon \leq 2\%$ (8,263 sites) on the right.

necessary to achieve a given accuracy. Such an increase is due to the spatial constraints introduced by the block boundaries.

Note also how the performance of data simplification, in terms of data needed to achieve a given accuracy, improves with the resolution of the input dataset. If we consider, for example, the *LoD* meshes at accuracy 1.0 % from the 32^3, 64^3 and 128^3 multiresolution models of BuckyBall, the percentage of sites needed to build each approximated mesh decreases respectively from 45.2% to 22.1% down to 6.8% of the total number of sites of the dataset. In absolute values, the ratio between the 128^3 and the 32^3 datasets is 64:1 at full resolution, while it decreases to 10:1 at accuracy 1.0%.

5.5.1 Rendering features evaluation

Figure 7.6 on page 137 presents visual results related to isosurface and direct volume rendering of three representations of the BluntFin dataset. The images at the top refer to the mesh at full resolution, the images in the middle refer to an approximated mesh at accuracy $\mu = (1.0\%, 1.0\%)$, while the images at the bottom refer to an approximated mesh at accuracy $\mu = (4.0\%, 4.0\%)$.

Numerical results regarding the size of the meshes of the extracted isosurfaces, as well as times for Direct Volume Rendering, are summarized in Table 5.4. The images provide evidence that the image degradation is almost inperceptable when passing from full accuracy to $\mu = (1.0\%, 1.0\%)$ accuracy, while it is still small at $\mu = (4.0\%, 4.0\%)$. On the contrary the output sizes (and times) are considerably reduced.

The visualization results obtained, which are essentially based on the concept of *data simplification*, can be also compared with results obtained with approximation methods based on *graphics output simplification*. In the case of isosurface rendering, the size and number of the facets extracted from a simplified mesh depend essentially on the variation of the field function (namely, few large facets are fitted on subvolumes where the gradient is constant or nearly constant). On the other hand, a *geometry-based simplification* of an isosurface extracted from the mesh at full resolution would be driven by isosurface curvature ([89, 54]). An obvious computational advantage of the approach based on data simplification is that the most strenuous part is made in a preprocessing stage (i.e., when the simplified or multiresolution model is built), while standard simplification approaches are implemented as a post-processing phase, therefore reducing throughput in interactive applications.

Moreover, standard geometry-based methods may produce anomalies if the surface has variations in curvature which are small in size, but reflect significant variations of the field (e.g., a sinusoidal function, having ampli-

	no. tetra.	no. sites	% of sites
BuckyBall (32x32x32)	32,768		
Levels of Detail:			
ε =4.0 (%)	43.929	7,426	22.6 %
ε =3.0 (%)	53,025	8,974	27.3 %
ε =2.0 (%)	65,409	11,133	33.9 %
ε =1.0 (%)	86,130	14,839	45.2 %
ε =0.5 (%)	106,695	18,584	56.7 %
ε =0.2 (%)	131,967	23,340	71.2 %
ε =0.1 (%)	151,345	27,073	86.6 %
ε =0.0 (%)	176,641	32,768	100 %
BuckyBall (64x64x64)	262,144		
Levels of Detail:			
ε =4.0 (%)	105,422	17,164	6.5 %
ε =3.0 (%)	140,183	22,833	8.7 %
ε =2.0 (%)	203,885	33,202	12.6 %
ε =1.0 (%)	353,652	58,014	22.1 %
ε =0.5 (%)	522,764	86,633	33.0 %
ε =0.2 (%)	749,259	125,711	47.9 %
ε =0.1 (%)	954,551	161,378	61.5 %
ε =0.0 (%)	1,483,742	262,144	100 %
BuckyBall (128x128x128)	2,097,152		
Levels of Detail:			
ε =4.0 (%)	178,138	28,272	1.3 %
ε =3.0 (%)	257,390	41,262	1.9 %
ε =2.0 (%)	424,283	67,878	3.2 %
ε =1.0 (%)	897,994	143,936	6.8 %
ε =0.5 (%)	1,672,207	269,195	12.8 %
ε =0.2 (%)	3,301,742	537,843	25.6 %
ε =0.1 (%)	4,748,306	780,509	37.2 %
ε =0.0 (%)	12,152,055	2,097,151	100 %

Table 5.3: Tetrahedralization of the BuckyBall dataset using the *block-decomposition refinement*: 128^3 dataset is the original one, while 64^3 and 32^3 datasets are obtained by subsampling. Decompositions: 32^3 divided into 64 blocks of size 8^3; 64^3 divided into 64 blocks of size 16^3; 128^3 is divided into 512 blocks of size 16^3.

Accuracy	no. vertices	no. tetra	no. iso. triangles	DVR time
(0.0%,0.0%)	40,960	222,528	19,499	44.1
(1.0%,1.0%)	14,162	80,883	9,143	16.1
(4.0%,4.0%)	3,612	20,324	3,442	3.9

Table 5.4: Isosurface rendering (with threshold value 1.244), and direct volume rendering of the Bluntfin dataset at different accuracies shown in Figure 7.6 on page 137. Times are in seconds on an SGI Indigo XS24 R4000.

tude lower than the simplification threshold), and, even worse than this, intersections between surfaces at different isovalues may occur because of simplification. These problems do not arise with methods based on data simplification.

In a recent paper [24], we have extensively compared the performance of the standard projected tetrahedra (PT) algorithm applied to a simplified mesh, to the performance of approximated versions of the PT algorithm [112] applied to a mesh at full resolution. Experiments provided evidence that images with visual degradations similar to those obtained using the approximated PT are produced using highly simplified datasets, thus achieving much shorter processing times (about five times shorter).

The large difference in speedups is due to the fact that standard approximated PT techniques only act on the pure rendering phase, thus achieving a reduction in overall time of up to a maximum of 50%. On the contrary, the speedup in overall time achieved by using a data simplification approach is linearly proportional to the simplification operated on data (this means that not only pure rendering is affected, but depth sorting, cell classification and splitting as well).

Chapter 6

Tetrahedral Multiresolution Models

A collection of simplified models can be managed in a single unified framework adopting a multiresolution representation; in this way the resolution and the size of the dataset can be adapted to the user's need. We show how these methods permit the compact representation of many different approximations of the dataset. Multiresolution allows, for example, the use of low resolution models for interactive phases or the extraction of variable resolution representations according to viewing parameters and/or to the user specification of a particular region of interest.

The iterative application of a simplification technique with different approximation parameters produces a collection of representations at different accuracies. A data structure that holds a *constant* (and usually small) number of different representations of the dataset, with various accuracies, is called a representation at different *levels of detail* (*LoD*). LoD representations of surfaces are widely used in a number of important applications (e.g., virtual reality based on VRML). An evolution of a LoD representation is a *multiresolution* representation, which supports (with the greatest flexibiltity) the compact storage of a number m (usually large) of representations at different levels of detail, where m is a monotonic function of the size of the input dataset (i.e., the more data, the more representations). Multiresolution or LoD can greatly improve the efficiency of data rendering, e.g., through suitable progressive visualization algorithms. The multiresolution approach improves over the LoD one with valuable characteristics:

- The user or the application have much more flexibility in selecting the "best" level of detail, depending on their specific needs in terms of accuracy, memory, and time performance: in many cases, it is better to leave that choice at run time, instead to force it in the preprocessing.

- Multiresolution schemes may be more compact in space than an *LoD* one, since they encode in a single structure a large number of different representations.

- Multiresolution representations can permit extracting models in which the resolution varies over the domain of the dataset; in this way the user, or the visualization system itself, may choose to render/manage with the highest detail only some parts of the dataset, for example the ones chosen by the user or the most *important* ones.

The chapter is organized as follows: in Section 6.1 we survey other approach to the multiresolution management of tetrahedral datasets; then in Section 6.2 we introduce a simple technique oriented towards the management of a large collection of levels of detail with a compact data structure; in Section 6.3 we adopt the Multiresolution Simplicial Model (MSM), introduced by De Floriani et al [34], as a unified framework to model multiresolution. In Section 6.3.3 we specialize this model for the handling of volume datasets describing how to extract variable resolution models. In Section 6.4 we face the problem of extracting a variable resolution model together with the face-adjacency topology and we propose an original data structure for the management of a MSM based on the concept of a MSM as a complex in a higher dimension.

6.1 Related Work

While many different approaches have been proposed for the multiresolution management of surfaces, (see, e.g., [32] for a survey), the multiresolution volume data management is still in a developing stage. Some of the proposed multiresolution models, like methods based on wavelets [104, 75] or on Multi-Dimensional Trees [109], work only on regular volume datasets. Following our tetrahedral framework we focus our interest on techniques for multiresolution management of tetrahedral meshes.

In [12] De Floriani et al. proposes the use of hierarchical simplicial complexes as a model for the multiresolution representation of a volume scalar field. This model is based on the recursive refinement of a tetrahedral complex: a tetrahedron σ of the complex can be refined by replacing it with tetrahedralization Σ_σ whose domain cover σ. The recursive application of this refinement process results in a hierarchy of tetrahedralizations. Algorithms for extracting models with a given accuracy or directly extracting isosurfaces from a hierarchical tetrahedral complex are presented.

Zhou, Chen and Kaufman have recently presented a multiresolution framework, called MultiTetra [118] for regular datasets based on tetrahedral subdivision. This is a 3D extension of the Lindstrom's subdivision strategy [67] for regular two-dimensional grids. The MultiTetra model is built by the recursive subdivision of tetrahedra along an edge; the starting model is a tetrahedralized cube. The subdivision process follows a regular pattern so the authors can represent all the resulting multiresolution model as a binary tree (each tetrahedron is always subdivided in two along an edge) whose leaves represents the tetrahedral cells at a given resolution. Using this approach the authors are able to support efficent extraction of variable resolution models and store the multiresolution model in a very compact way.

A multiresolution model for simplicial complexes, called Progressive Simplicial Complex (PSC) has been proposed by Popovic and Hoppe [81], as an extension of the Progressive Meshes (PM) model [54]. The PM models are based on the simplification of a mesh with a sequence of edge-collapse transformations, this sequence of operations is encoded in the PM structure. Given a PM, a mesh can be reconstructed by applying, in the right order, a series of vertex-split transformations (the reverse of edge-collapse). The PSC codifies in a similar manner the sequence of more general edge-collapse transformations. It should be noted that while the PSC are quite general, they has been conceived for the management of 2D surfaces rather than simplicial complexes, so the conditions of legality of a sequence of vertex-split operation of a generic complex are not specified.

A comprehensive multiresolution model for simplicial complexes has been introduced in [34]. This framework is the one we have chosen and is described in detail in Section 6.3.

6.2 The Historical Model for Multiresolution

Each one of the algorithms described in the Chapter 5 can be regarded as producing an "historical" sequence of tetrahedra, namely all the tetrahedra that appear in the current mesh Σ during its construction. Based on such an observation, we have extended in [21] to the three-dimensional case a simple idea to manage multiresolution, which we proposed in [27] in the two-dimensional case, for the multiresolution representation of terrains. A very similar approach has been independently described in [13] for the management of pyramidal simplicial complexes and called Sequence of List of Simplices (SLS). This structure encodes a simplicial complex represented by a collection (a pyramid) of complexes with different accuracies and covering the same domain. The main idea of the SLS is the same of the historical model (giving to each cell a accuracy interval), but the objectives are different: we need just a compact way of storing and retrieving a large number of LOD's, while SLS's are a powerful tool that permits also to solve interference queries.

The structure presented here, though it does not support sophisticated features like variable resolution extraction, it is well suitable to store, in a compact way, a large number of different levels of details. It can be effectively used, for example, in visualization systems where the choice of the size of the level of detail depends both on user needs and hardware characteristics and therefore it cannot be defined apriori.

The main idea of the *historical* approach is that each tetrahedron of the sequence is marked with two accuracies μ_b and μ_d, called *birth* and *death* accuracy, and corresponding to the worst and best accuracy of a mesh containing it, respectively. The intuitive meaning of the names refers to the fact that, adopting a *refinement* strategy of simplification μ_b and μ_d are the accuracies of the mesh when σ appears and disappears respectively. Tetrahedra belonging to the final mesh are assigned a death of $(0, 0)$. The two birth/death values are swapped in case the historical sequence is built by a *decimation* strategy.

For each site in the dataset, we store its coordinates and field value, while for each tetrahedron in the historical sequence, we store its vertex indexes and the *birth* and *death* accuracies. Space occupancy obviously depends only on the number of sites and on the number of tetrahedra in the historical sequence.

Querying the model Given a query accuracy μ, the mesh at accuracy μ will be formed by all the tetrahedra that are μ-*alive*, i.e., such that $\mu_d \leq \mu \leq \mu_b$. Based on this fact, we use μ_b and μ_d as filters to retrieve tetrahedra that either form a given mesh, or cover a given range of accuracies, from the historical sequence. Such a filter can also be combined with a spatial filter to perform windowing operations, i.e., to retrieve only tetrahedra belonging to a given query region.

The query time can be optimalized by using an *interval tree* [38, 83]. In our case, the interval tree consists of a binary search tree with $O(h)$ nodes, where h is the total number of accuracies spanned by the multiresolution model: each leaf corresponds to an accuracy, and leaves are sorted in order of increasing accuracy. Each internal node n has a discriminant value μ_n associated with, and two sorted lists of tetrahedra μ_n-alive are appended to n, sorted according to their birth and death accuracies, respectively. With such a data structure, a mesh of k tetrahedra can be retrieved in optimal time $\theta(\log h + k)$. Note that this data structure is more expensive than the previous one, since every tetrahedron appears in two lists, and the whole structure must be maintained in the main memory, in order to achieve a true speedup with respect to the data structure described in the previous paragraph.

A data structure for sequential files. If the multiresolution model must be maintained in a sequential file, tetrahedra forming the historical sequence can be simply saved into the file in the order they appear during the refinement (in the inverse order if the model is built through decimation).

In this case, the sequence of tetrahedra belonging to a model at a given resolution μ is obtained by sequentially scanning the file, and selecting tetrahedra according to their birth and death accuracies: only tetrahedra that are μ-alive are accepted, and the search stops as soon as a tetrahedron having a birth accuracy better than μ is found. This search may take a time linear in the total number of tetrahedra of the model in the worst case.

6.2.1 Transmitting the model through the network

If a multiresolution model must be transferred from a server to a client over the network, it is important to compress information further.
Conciseness can be achieved by avoiding the explicit transmission of tetrahedra forming the historical sequence, but providing an implicit encoding that allows the client to make the structure explicit efficiently. To this aim, we directly extend techniques recently proposed for surface simplification [62, 54].

If the model is built through a refinement strategy, by exploiting the properties of Delaunay tetrahedralizations, we can transmit only the vertices of the final mesh Σ in the order they were inserted during refinement (i.e., in the order we store them on file). For each vertex, we send to the client its coordinates, its field value, and the accuracy of the mesh just after its insertion. This allows the client to reconstruct the whole historical sequence in the right order, by applying a procedure for on-line Delaunay tetrahedralization [57] while vertices are received. Note that this is a task much cheaper than rebuilding the model from the initial dataset, since the selection of vertices now comes free from the sequence. Moreover, the on-line construction performed by the client directly results in a progressive representation (and, possibly, rendering) of the mesh at the highest resolution.

If the model is built adopting a decimation strategy, a similar technique may be adopted, following Hoppe [54]. In this case, the coarsest mesh is transmitted explicitly, while the remaining vertices are listed in inverse order of decimation (i.e., in the order we store them in the file). For each vertex, we send to the client its coordinates, its field value, the accuracy of the mesh just before its deletion, and the vertex it was collapsed on. This last information permits us to perform a *vertex-split* operation that inverts the *edge-collapse* performed by the decimation algorithm [54]. Therefore, the client can generate the whole historical sequence in the right order, by using a sequence of vertex splits. Similarly to the previous case, mesh reconstruction is performed by the client efficiently, and progressive transmission and rendering are supported. Note that, in this case, operations performed by the client at each vertex split are much simpler than those required by a Delaunay procedure, but, on the other hand, the amount of information transmitted is slightly larger.

In both cases, we transmit only the list of vertices (with for each vertex its coordinates, field value, and birth accuracy). The size of data transmitted can be further reduced by using geometric compression [35].

6.3 A Framework for Multiresolution

The multiresolution model presented in the previous section presents some limitations, the most important one is its inability of extracting models with varying resolution. In this section we introduce the framework of multiresolution simplicial models (MSM) introduced by De Floriani, Puppo and Magillo [34] as a multidimensional extension of the two dimensional structure described by Puppo in [84].

In this framework we will propose in Section 6.4 an original data structure to manage a MSM together with all the face-adjacency topological information. This structure, called Hyper Simplicial Complex (HySC), codifies a MSM in $\mathbb{E}^{\mathrm{d}}$ through its embedding as complex in $\mathbb{E}^{\mathrm{d+1}}$.

6.3.1 Definitions

Let S be a finite set. A partial order on S is a antisymmetric and transitive relation $<$ on its elements. A pair $(S,<)$ is called a *partially ordered set (poset).* For every $s, s' \in S$ with $s <' s$ we mean that $s < s'$ and $\nexists s''$ such that $s < s'' < s'$. A subset $S' \subseteq S$ is called a *lower set* if $\forall s' \in S', s < s' \Rightarrow s \in S'$. Intuitively S' is a lower set if it contains all the elements that precede each of its elements. The algebraic structure of a poset $(S,<)$ can be described by a DAG, where nodes represent elements of S and arcs encode the $<'$ relation. For any $s \in S$ the set $S_s = \{s' \in S | s' \leq s\}$ is the smallest lower set containing s and it is called the *down-closure* of s. The *sub-closure* of s is defined as $S_s^- = S_s \backslash \{s\}$.

We call any finite set of d-simplexes in $\mathbb{E}^{\mathrm{n}}$ a *d-set*; a regular d-simplicial complex Σ is completely characterized by the collection of its d-simplexes i.e. by the d-set associated with Σ.

When managing a collection of representation of the same complex at different resolutions, as done in the previous section, we need a measure of the error we commit. With $\mu(\sigma)$ we denote a function $\mu : \Sigma \rightarrow [0, 1]$ measuring this error, $\mu(0)$ means exact representation. With $\mu(\Sigma)$ we denote the maximum error among all the tetrahedra of Σ: $\max_{\sigma\in\Sigma} \mu(\sigma)$.

Operators: interference and combination We define two operators on d-sets: the interference operator $\otimes$ and the combination operator $\oplus$. Both operators take two d-sets as arguments and produce a d-set. The *interference* operator of two d-sets is defined as:

$$\Sigma_i \otimes \Sigma_j = \{\sigma \in \Sigma_i | \exists \sigma' \in \Sigma_j, i(\sigma) \cap \sigma' \neq \emptyset\}$$

In other words, the interfernce of two d-sets Σ_i, Σ_j is the set of the simplexes of Σ_i that have a proper intersection with some simplexes of Σ_j.
The *combination* operator of two d-set is defined as:

$$\Sigma_i \oplus \Sigma_j = \Sigma_i \backslash (\Sigma_i \otimes \Sigma_j) \cup \Sigma_j$$

In other words, the combination of two d sets Σ_i, Σ_j is the set of the simplexes of that can be obtained by adding to Σ_j the simplexes of Σ_i not intefering with Σ_j.

If $\Sigma_i \oplus \Sigma_j$ is a d-simplicial complex and $\Delta(\Sigma_i \oplus \Sigma_j) = \Delta(\Sigma_i) \cup \Delta(\Sigma_j)$, then the complex Σ_j it is said to be *compatible* over Σ_i.

Let $[\Sigma_0, \ldots, \Sigma_k]$ be a sequence of d-complexes. the *combination* $\oplus_{i=0}^{k} \Sigma_k$ is defined as:

- if $k = 0$ then $\oplus_{i=0}^{0} \Sigma_k = \Sigma_0$
- if $k > 1$ then $\oplus_{i=0}^{k} \Sigma_k = (\oplus_{i=0}^{k-1} \Sigma_k) \oplus \Sigma_k$

6.3.2 Multiresolution Simplicial Model

A Multiresolution Simplicial Model (MSM) on Ω is a poset $(\mathcal{S}, <)$, where $S = \{\Sigma_0, \ldots, \Sigma_h\}$ is a set of d-complexes and $<$ is a partial order on $\mathcal{S}$ satisfying the following conditions:

1. $\Delta(\Sigma_0) = \Omega$,
2. $\forall i, j = 0..h, i \neq j,,$
   (a) $\Sigma_i <' \Sigma_j \Rightarrow \Sigma_i \otimes \Sigma_j \neq \emptyset$
   (b) $\Sigma_i \otimes \Sigma_j \neq \emptyset \Rightarrow \Sigma_i$ is in relation with Σ_j

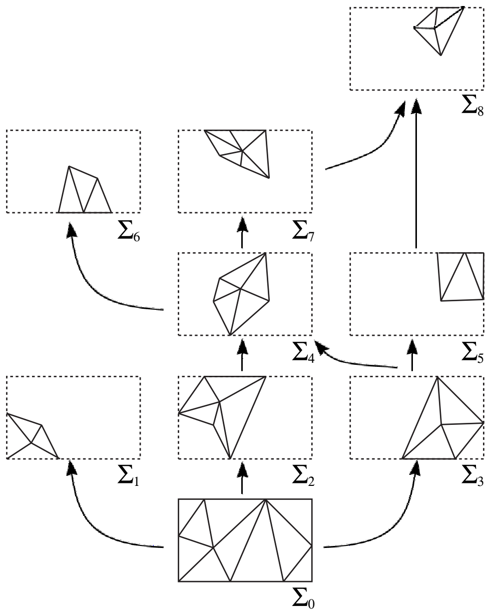


Figure 6.1: The DAG describing a simple two-dimensional MSM.

3. the sequence $[\Sigma_0, \ldots, \Sigma_h]$ of all complexes of $\mathcal{S}$ defines a consistent order w.r.t. relation $<$ and $[\Sigma_0, \ldots, \Sigma_h]$ is a compatible sequence.

The meaning of the second condition becomes clearer if we consider the DAG encoding relation $<'$ for the set $\mathcal{S}$:

- if two complexes Σ_i and Σ_j are connected by an arc, they are interfering;
- if two complexes Σ_i and Σ_j interfere then they are connected throug a path.

The elements of $\mathcal{S}$ are called components or fragments; intuitively the fragments describe a portion of the domain Ω at a certain resolution. For example, if we think to an iterative refinement procedure on a simplicial mesh, the set of simplexes, derived from the substitution of a complex with a more refined one, can be considered as a fragment that is combined over the existing complex. Combining a lower set of $\mathcal{S}$ give us a complete description of Ω at various resolutions.

The following properties holds for MSM's:

Lemma 1. Σ_0 *is unique minimum element of* $(\mathcal{S}, <)$.

In other words Σ_0 is the starting coarsest complex. Given any subset $\mathcal{S}' \subseteq \mathcal{S}$ the total order of its elements consistent with the sequence $[\Sigma_0, \dots, \Sigma_h]$ is called the *default order* of $\mathcal{S}'$.

Lemma 2. *The default order of any lower set $\mathcal{S}' \subseteq \mathcal{S}$ is a consistent order.*

Lemma 3. *In a MSM $(\mathcal{S}, <)$, the combination of a lower set $\mathcal{S}' \subseteq \mathcal{S}$ is independent of the specific consistent order.*

Since the combination obtained from any consistent order is unique, it will simply called the combination of $\mathcal{S}'$ and denoted with $\oplus\mathcal{S}$. Let Σ_i be a component of $(\mathcal{S}, <)$; the combination of the subclosure $\mathcal{S}_\Sigma^-$ is called the *support* of Σ_i; the set of the simplices of the support that are interfering with Σ_i, $(\oplus\mathcal{S}_\Sigma^-) \otimes \Sigma_i$is called the *floor* of Σ_i.

The following definitions permit us to consider a particular class of MSM's where the order relations provide control over the size in terms of number of simplexes. A MSM $(\mathcal{S}, <)$ is *increasing* if and only if for every pair of lower sets $\mathcal{S}'$,$\mathcal{S}''$ holds: $(\mathcal{S}' \subset \mathcal{S}'' \Rightarrow |\oplus \mathcal{S}'| < |\oplus \mathcal{S}''|$. Similarly is defined the concept of *decreasing* MSM; an increasing or decreasing MSM is said *monotone*. In other words a MSM is increasing (decreasing) if and only if the size of each fragment is larger (smaller) than the size of its floor.

In Figure 6.1 is shown a simple multiresolution simplicial model for the two dimensional case; the arrows in the figure correspond to the relation $<'$. The fragments of a MSM can be used to build different triangulations of the domain Ω through the paste operator. The intuitive meaning of the $<$ relation is of dependence between the pasting of the fragments: if we use a fragment Σ_i in a triangulation then all the fragments $\Sigma_j < \Sigma_i$ must also be used.

In Figure 6.2 is shown the triangulation resulting from the pasting/combination of a subset $\mathcal{S}'$ of fragments in a consistent order $\Sigma_0, \Sigma_2, \Sigma_3, \Sigma_4, \Sigma_7$; note that any other consistent order of pasting of $\mathcal{S}'$ (like $\Sigma_0, \Sigma_3, \Sigma_2, \Sigma_4, \Sigma_7$) builds the same triangulations.

Topological Information The topological information of a MSM, which encodes the adjacency, boundary and coboundary relations among the simplexes of the model, can be distincted in local topology and global topology; local topology is concerned with topological relations between simplexes within the same component, while global topology considers relations between simplexes not necessarily belonging to the same component. In the next subsections we will describe the standard data structure for MSM that does not codify any topological information and in section 6.4 we will introduce the Hyper Simplicial Complexes, an original data structure for mantaining the global topology in a MSM.

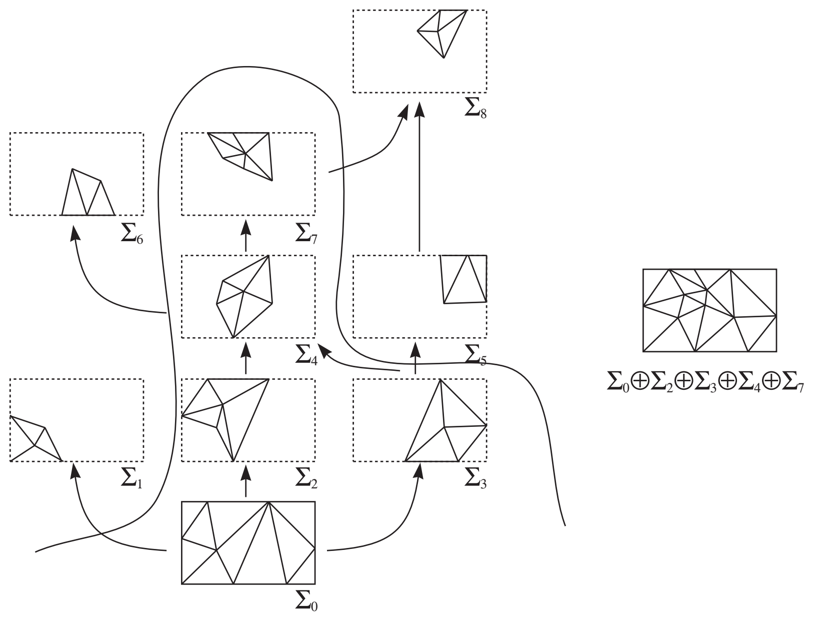


Figure 6.2: A subset $\mathcal{S}' \subset \mathcal{S}$ combined in a consistent order builds a triangulation

6.3.3 MSM for Volume Visualization

Each algorithm described in Chapter 5 can be used to build a MSM. Both a decimation or a refinement algorithm for simplifying a tetrahedral complex can be regarded as producing an "historical" sequence of tetrahedra, namely all tetrahedra that appear in the current mesh Σ during its construction. An historical sequence can be also viewed as the sequence of all subdivisions of the whole domain that are obtained through changes, or as an initial subdivision plus a sequence of fragments reflecting the local changes iteratively done to the mesh, i.e. subdivisions of portions of the domain, which can be partially overlapping and are pasted one above the other to update the existing structure.

For example, if we follow the *refinement* heuristics, the minimum of the poset is the starting coarse triangulation; when we insert a new point v_i in the complex the new tetrahedra that are built form a new fragment Σ_i; the floor of this fragment is constituted by the tetrahedra that were destroyed by the insertion of v_i.

Following the MSM all these fragments, represented by a 3-simplicial complex covering a small part of the whole domain Ω, are arranged in a poset where the order relation between fragments is dependent on their interferences in 3D space. The minimum fragment Σ_0, the coarsest representation of our mesh, has an empty floor. Similarly all the triangles on the top of $\mathcal{S}$, representing the dataset at its full resolution, have no upper fragments.

In the next subsections we will describe how to effectively manage a MSM for tetrahedral complexes, describing the data structure that we need and an algorithm for the extraction of a variable resolution model. The algorithms and data structures here presented are a straightforward extension to tetrahedral complexes of the one presented in [30, 84]. They are needed for an easier introduction and for a comparision with our structure, the hyper simplicial complexes, that we present in Section 6.4.

6.3.4 A data structure for encoding a tetrahedral MSM

In this section we describe the data stuctures presented in [30] to encode a MSM for the specific case of tetrahedral complexes. We need to maintain the set of all the vertices of $\mathcal{S}$, the set of all tetrahedra $\Sigma_{\mathcal{S}}$, the set of all fragments $\mathcal{S}$, and for each tetrahedron, the reference to the vertices forming it and its accuracy. To codify the $<'$ relation we note that each tetrahedron σ is referenced by two fragments:

- the fragment Σ_i containing σ and called *lower* fragment;
- the fragment containing σ in its floor, called *upper* fragment.

We also note that, for each fragment, we need to recover the tetrahedra composing Σ_i and the floor of Σ_i. Expliciting all these needs, our data structure will contain:

- a vector `V` containing the coordinates of all the vertices in $\mathcal{S}$;
- a vector `T` containing all the tetrahedra description;
- a vector `F` containing the description of all the fragments;

Each element of vector `T` describes a single tetrahedron σ and contains the indexes of the vertices of σ in `V`, the accuracy $\mu(\sigma)$ and two pointers to the lower and upper fragments. Each element of vector `F` describes a single fragment Σ_i and contains a vector with the pointers to the tetrahedra composing Σ_i and another vector with the pointers to tetrahedra composing the floor of Σ_i.

With this data structure it is possible to implement the following operations with a complexity that is linear in the size of the output:

- Floor(Σ_i): returns the tetrahedra forming the floor of Σ_i,

- Lower(σ),Upper(σ): return the lower and upper fragment of σ, respectively.

6.3.5 Extracting a variable resolution model

We suppose that our MSM is monotone, to extract a variable resolution model we need a boolean acceptance function $c(\sigma)$ in order to decide, for a given tetrahedron, if its accuracy is sufficient or if we need to further refine that part of the domain. The choice of an acceptance function suitable for Volume Visualization will be the topic of Section 6.5. The algorithm to build a variable resolution model is shown in figure 6.3; it tries to incrementally build the desired solution by adding new fragments to the current solution. The algorithm is based on a breadth-first traversal of the DAG representing the MSM. The traversal starts from the coarsest fragment Σ_0, root of the DAG, and fragments above the current solution are progressively traversed and marked. The current solution is maintained as a list of tetrahedra Σ_{Out}. For each fragment Σ that we encounter in the traversal of the tree, the following two loops are executed:

- we search for fragments before Σ, still not visited and, if found, they are added to the traversal queue Q. All the fragments before Σ can be found by checking, for each tetrahedron $\sigma \in \Sigma$, if the corresponding lower fragment Lower(σ), has been marked.

- for each tetrahedron $\sigma \in \Sigma$, if it satisfies the acceptance function $c(\sigma)$ then σ is added to the current solution, else we add the upper fragment of σ to the traversal queue Q and mark it to be removed from the solution.

The correctness of this algorithm has been proved in [31].

6.4 Hyper Simplicial Complex

Though the MSM represents a valid framework for the modeling of multiresolution, the data structure presented in the previous Section cannot manage global topology, that is the (d-1)-face adjacency relation between simplexes. In other words the extracted models are just a collection of simplexes, in our case tetrahedra, without explicit representation of adjacency between tetrahedra. The adjacency relations can be very useful in many situations in Volume Visualization, for example to exploit locality in isosurface extraction (cfr. Section 2.2.1) or to use a topological sort for Direct Volume Rendering (cfr. Section 3.2).

```
    procedure EXTRACT($\mathcal{S}, c(), \Sigma_{Out}$);
    local var $Q$: queue;
    local var $\Sigma$, $\Sigma_2$ : Fragment;
    local var $\sigma$ : tetrahedron;
    $\Sigma$ = Least($\mathcal{S}$)
    Mark($\Sigma$)
    Add($Q$,$\Sigma$)
    while $Q \neq \emptyset$
        $\Sigma$ = First($Q$)
        foreach $\sigma \in$ Floor($\Sigma$)          *check if we have visited all fragments before $\Sigma$*
            Mark($\sigma$)
            $\Sigma_2$=Lower($\sigma$);
            if $\Sigma_2 \neq \emptyset$ and not Marked($\Sigma_2$)
                Mark($\Sigma_2$)
                Add($Q$,$\Sigma_2$)
        foreach $\sigma \in \Sigma$
            if $c(\sigma)$
                then
                    Add($sigma, \Sigma_{Out}$)
                else
                    $\Sigma_2$=upper($\sigma$);
                    Mark($\sigma$);             *tetrahedron t is not good*
                    Mark($\Sigma_2$);
                    Add($\Sigma_2, Q$)
    foreach $\sigma \in \Sigma_{Out}$                  *Remove marked tetrahedra from solution*
        if Marked($\sigma$) then Remove($\sigma, \Sigma_{Out}$)
end ;
```

Figure 6.3: The algorithm for extracting a variable resolution tetrahedral complex in the MSM framework.

In this Section we will follow the *Hypertriangulation* (HyT) approach to interpret a MSM with all the face-adjacency information as a complex in $\mathbb{E}^{d+1}$. We proposed this approach in two dimension [27, 28] to manage the representation of discrete topographic surfaces at variable resolution. The main idea of HyT consists in the interpretation the model containing all the triangles of a *historical* sequence of a simplification as a cell complex embedded in 3D space. Intuitively, we look at the simplification process as a sequence of local modifications in which we substitute a subset of triangles with a (possibly) larger one covering the same domain and sharing the same boundary: we store all these modifications by *sewing* the patch formed by the new triangles over the old ones.

In Figure 6.4 we show this process for a pair of refinement operations: by sewing the patch of new triangles over the existing mesh we codify the multiresolution model as a cell complex in 3D.

Following the terminology introduced in Sections 6.3 and 6.3.3 for MSM's, we can give the following interpretation of a hypertriangulation:

- the upper part of each patch encodes a fragment Σ_i of a MSM;
- the lower part of each patch, formed by the triangles covered by the patch, is the floor of Σ_i.
- the $<'$ relation of MSM corresponds to the concept of *above* between a patch/fragment Σ_i and the patches under Σ_i.

The main idea of the HyT is that the $<$ relation can be encoded by the topology itself. It can be showed that, if we do not permit unconnected fragments, i.e. each fragment is a single connected component, any two-dimensional MSM can be codified by a hypertriangulations: the third condition of the definition of MSM guarantees that the sewing is always possible, infact the existence of a compatible sequence $[\Sigma_0, \ldots, \Sigma_k]$ means that all the partial pastings of a fragment over the previous ones gives a simplicial complex, so the boundary of a fragment and its floor must coincide.

In the case of the fragments made by more than one single connected component the case MSM express stronger constraint for dependences between fragments. Let Σ_{i1}, Σ_{i2} be the two connected components of a single fragment, with floor $\Sigma_{i1}^F, \Sigma_{i2}^F$; in HySC we model this situation as the sewing of Σ_{i1} and Σ_{i2} over Σ_{i1}^F and Σ_{i2}^F, respectively. The $<$ relation is implicitly stored in the topology of the HySC, so we have codified $\Sigma_{i1}^F < \Sigma_{i1}$ and $\Sigma_{i2}^F < \Sigma_{i2}$ but not, for example, $\Sigma_{i1}^F < \Sigma_{i2}$. We think that such constraints are not very frequent or useful.

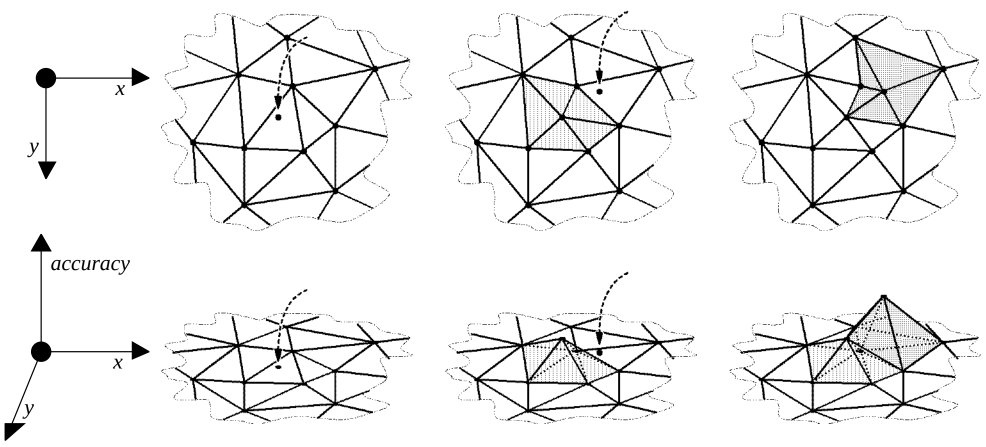


Figure 6.4: We can codify a fragment Σ_i and its floor Σ_j as the sewing of the *patch* made with triangles of Σ_i over its floor Σ_j.

6.4.1 Managing HyT and HySC

The HyperTriangulation approach can be extended to the most general of multiresolution simplicial complexes in $\mathbb{E}^d$ building what we call a *Hyper Simplicial Complex* (HySC). For sake of clarity we will continue to show examples only in two dimensions. When managing a HyT we are interested in efficiently traverseing the two-dimensional skeleton of a particular class of a complex. Here we define some traversal operation to navigate a HyT and their direct extention to HySC. In the next subsection we will show how to efficiently implement the operations for extracting a variable resolution model from a MSM using the HySC as underlying structure. In this way we will be able to extract also all the face-adjacency topologic information for the variable resolution model.

As shown in figure 6.4, when sewing a fragment over an existing mesh the sewed edges share more than two facets; in the general case it means that the (d-1)-facets on the boundary of the new fragment are shared by more than two d-simplexes. Given the structure of the complex representing a HyT we can imagine to subdivide all the facets incident on a single edge in two groups, namely those formed by the faces whose projection on the original space of the complex (the xy plane of figure 6.4) lie on the left or on the right of the projection of the edge. The same subdivision can be done in higher dimensions, partitioning the d-simplexes using the hyperplane passing through the common (d-1)-face. After this partitioning all the facets lying on one side of a given edge can be linked and ordered together according their accuracy.

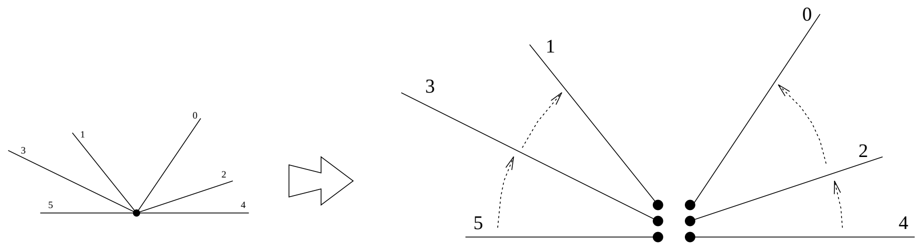


Figure 6.5: Facet-edge ordering in the half-facet-rings follows the error ordering (on the right: the lines represent faces, the bullets represent coincident edges, the arrows represent the linking between facets on the same side

Figure 6.5 shows a side view of the facets incident on a single edge, the arrows represent the linking between the facets and the numbers the error (the lower the error the higher the accuracy) for the faces.

Now we can introduce the operators for navigating a HySC. We base our navigation function on pairs (σ, f) formed by a d-simplex σ and a (d-1)-facet f of σ. In two dimensions this pair is the same that is the base of the *facet-edge* data structure proposed by Dobkin and Laszlo [36]; here we use an extension of generic d-simplexes. We assume that the (d-1)-facets of a d-simplex of our complex can be consistently ordered. Moreover we can associate to each pair (σ, f) the vertex of σ that is not contained in f. Our representation of a HySC allows the following navigation operators:

- $Up(\sigma, f)$, $Down(\sigma, f)$ returns, if exists, the pair (σ', f), formed by the facet f and the simplex σ' with lower (higher) accuracy, immediately above (under) σ;

- $Other(\sigma, f)$ returns the pair (σ', f) formed by the same facet f and the simplex σ' on the other side w.r.t. σ and with accuracy lower or equal to the one of σ.

- $Next(\sigma, f)$ returns the pair (σ, f') formed by the same simplex σ and the next face f' of σ. It holds $Next^{d+1}(\sigma, f) = (\sigma, f)$.

Intuitively the first two functions provide the means to navigate the HySC moving in the accuracy space, while the last two permit to move in the domain of the original complex. The arrows of Figure 6.5 represent the application of the *Up* operator, while the arrows of Figure 6.6 denote the application of the *Other* function. Remember that the numbers represent the error (the lower the error the higher the accuracy) for the faces.

The data structure codifying a pair (σ, f) can be described with a C-like syntax, as follows:

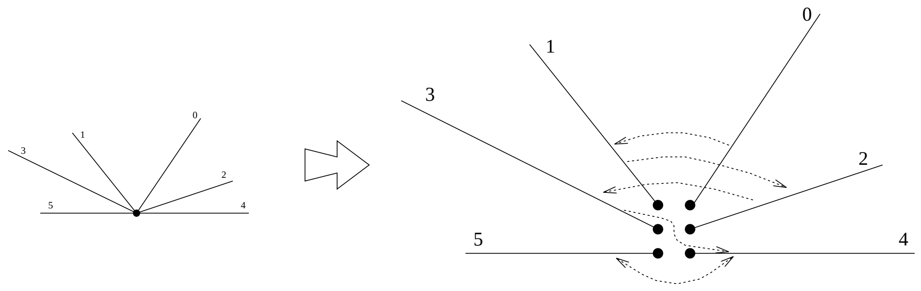


Figure 6.6: The result of the *Other* function.

```
struct SimplexFace
{
 Vertex *v;
 SimplexFace *Other;
 SimplexFace *Next;
 SimplexFace *Up;
 SimplexFace *Down;
}
```

This structure can be codified in a more compact manner using the data structure called Packed Facet-Edge (PFE) that we have proposed in [23] for the two dimensional case of HyT: instead of having a single facet-edge for each edge-face pair, the PFE representation encodes into a single record all the facet-edges incident on a given oriented edge from one of its sides. This packing strategy can be applied also to our SimplexFace structure.

6.4.2 MSM operation with HySC

The first observation is that in HySC there is not an explicit representation of fragments, but all the information that we need can be easily reconstructed from any simplex belonging to it, so we can use one of the simplexes that belong to Lower(Σ) to represent the fragment Σ. The functions Lower, Upper and Floor presented in previous sections must be adapted as follows in order to manage simplexes in HySC, instead of fragments.

- *Lower*(σ): we can retrieve all the simplexes belonging to the same fragment of σ by visiting, through face adjacencies, all the simplexes that can be reached through faces f such that $Down(\sigma, f) = \emptyset$;

- *Upper*(σ): we move, through adjacencies, until a simplex with $Up(\sigma, f) \neq \emptyset$ is found, and then we move up.

- *Floor*(σ): we can retrieve all the simplices belonging to the floor, by reaching the boundary of our patch, moving down and visiting all the simplexes $Up(\sigma, f) = \emptyset$;

The first and last functions return a set of simplexes, the second one returns a simplex representant of a fragment. The algorithm presented in figure 6.3 can therefore be modified as shown in Figure 6.7 in order to extract from a HySC $\mathcal{H}$ a variable resolution model. The main idea is to represent each fragment Σ with one of the simplexes that belong to Lower(Σ). For this reason we keep in our stack just a set of simplexes, when we extract one of them we can think we have extracted the fragment having it in Lower(Σ). We must pay a little of attention with the marking strategy: we distinguish two kind of marking for simplexes: the first, called MarkV, to denote that we have already visited the fragment represented by σ, the second one, called MarkE, to denote that σ cannot be part of the solution. This distinction between marks is due to the fact that we have no way of explicitly marking a fragment. In the presented code we denote with σ_f, σ'_f the simplexes that are interpeted as representant of fragments. When marking a simplex σ_f representant of fragment σ we must pay attention to mark all the simplexes in Lower(Σ).

The correctness of this algorithm strictly depends on the correctness of the Lower, Upper and Floor functions, because the main structure of this algorithm is the one for general MSM.

It can be noted that the cumulative time spent in traversing the HySC for executing the Floor, Lower, Upper functions during a variable resolution extraction is linear with the number of simplexes of the visited fragments. Infact the Floor operation, whose cost is linear in the number of returned simplexes, is applied at most once for each fragment, similarly the Lower operation, is executed at most twice for each simplex. The most complicated part of the analysis regards the Upper function. It could seem that the search for the boundary of the floor, needed to retrieve a representant of the upper fragment, could be done once for each simplex belonging to the floor of the upper fragment; this can be simply avoided by marking (with a third mark!) the simplexes traversed in this search and abort the search returning nothing when we encounter a marked simplex. Infact if we found a marked simplex it means that we have already executed an Upper operation on the simplex beloging to that floor of a fragment Σ and therefore we have already inserted that fragment into Q.

Once we have collected all the simplexes of our variable resolution complex, it is easy to reconstruct the face-adjacency topology using the HySC; for each simplex σ, if simplex σ' such that $(\sigma', f) = Other(\sigma, f)$ belongs to the solution (i.e. is not marked) we can link σ and σ' together through face

```
    procedure EXTRACT(HySC $\mathcal{H}, c(), \Sigma_{Out}$);
    local var $Q$: queue;
    local var $\sigma, \sigma', \sigma_f, \sigma'_f,$ : tetrahedron;
    $\Sigma = \text{Least}(\mathcal{H})$
    foreach $\sigma \in \Sigma$
            MarkV($\sigma$)
    choose a tetrahedron $\sigma \in \Sigma$
    Add($Q$,$\sigma$)
    while $Q \neq \emptyset$
        $\sigma_f = \text{First}(Q)$
        foreach $\sigma \in \text{Floor}(\sigma_f)$          *check if we have visited all fragments before $\Sigma$*
            MarkE($\sigma$)
            if not MarkedV($\sigma$)
                Add($Q$, $\sigma$)
                foreach $\sigma' \in \text{Lower}(\sigma)$
                    MarkV($\sigma'$)
        foreach $\sigma \in Lower(\sigma_f)$
            if $c(\sigma)$
                then
                    Add($\sigma, \Sigma_{Out}$)
                else
                    $\sigma'_f$=Upper($\sigma$);
                    MarkE($\sigma$);             *tetrahedron $t$ is not good*
                    MarkV($\sigma'_f$);
                    Add($\sigma'_f, Q$)
    foreach $\sigma \in \Sigma_{Out}$                   *Remove marked tetrahedra from solution*
        if MarkedE($\sigma$) then Remove($\sigma, \Sigma_{Out}$)
end ;
```

Figure 6.7: The algorithm for extracting a variable resolution tetrahedral complex in the HySC framework.

f; otherwise we search for the first simplex σ'' such that $(\sigma'', f) = Up^*(\sigma', f)$, where $Up^*()$ is the iterative application of the Up operator.

In [23] we have also described an algorithm for extracting a variable resolution model form a 2D HySC that works with a completely different strategy: instead of traversing bottom-up the MSM we walk orthogonaly over the domain of the complex incrementally building the variable resolution model. Although its worst case computational complexity is higher than the algorithms here presented, it shows good empirical perfomances.

6.5 Acceptance Functions for Variable Resolution

In the previous section we have shown how to manage multiresolution tetrahedral models and discussed how to extract models where the representation error is not constant. In this section we show how effectively use multiresolution in Volume Visualization, why variable resolution models are important and what kind of acceptance function we can use to define our variable resolution model.

The definition of an acceptance function implies the definition of what should be considered *important.* We can identify two different strategies for considering a portion of the domain more or less important:

- **range based**: the user, or the system itself, considers more important a particular range of the domain of the field value;
- **space based**: the user, or the system itself, consider more important a particular portion of the spatial domain of the dataset.

Range Based Acceptance Function In this case the user specifies a value or a range of values and the dataset is extracted with varying resolution: the highest resolution is reserved to the portions of the domain containing the desired values. In the case of volume rendering if the Discontinuous Transfer Functions (DTF) introduced in section 4.3 are used, this definition can be implicitly done by the user; the *important* values are the ones where DTF has a C^0 discontinuity; moreover we can assign the lowest importance to the values of the field that are mapped by the DTF in colors with a high transparency. These regions have a minimal contribution to the final image.

If we link the acceptance function to the DTF, the varying resolution model of the dataset must be modified each time we change the DTF, so a multiresolution model that suppors efficient updating of the current model-

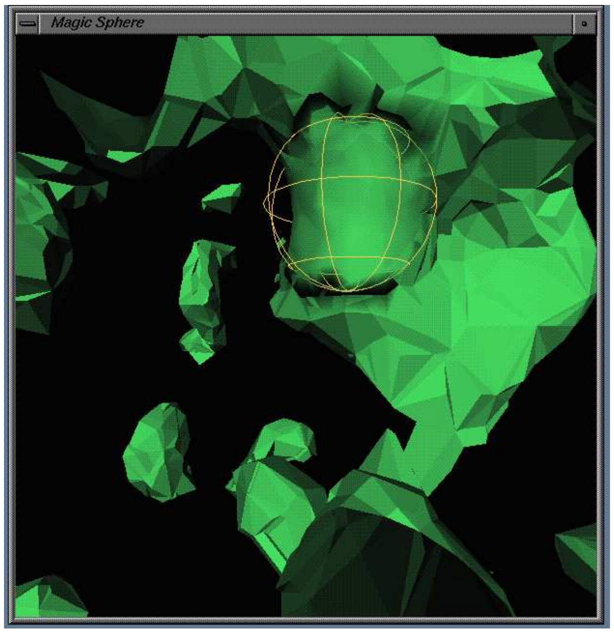


Figure 6.8: The MagicSphere tool used to specify a high resolution area on an isosurface on the SOD dataset.

has to be used. The MSM and their implementations based on the HySC support this feature.

Space Based Acceptance Functions In addition to the range based methods, we should also allow the user to specify a region of interest where he want the higher resolution. To perform this kind of specification interactively we have proposed 3D widget called MagicSphere [25]. This tool, that is an extension of the ToolGlass^{TM} paradigm [14] is presented as a transparent or wireframe sphere whose position and size in 3D space are interactively controlled by the user. The center of the MagicSphere specifies the center of the region of interest where the user want the dataset at the highest resolution, and the its radius specify how quickly the resolution must decrease. The MagicSphere tool is shown in Figure 6.8; the isosurface inside the MagicSphere has a higher resolution. Note that the cracks on the boundary of the MagicSphere are due to the lackness, at implementation time (1994) of a multiresolution model able to extract a variable resolution representation of the dataset.

6.6 Conclusions

In this chapter we discussed the use of multiresolution models in tetrahedral volume visualization. In the framework of the Multiresolution Simplicial Model we have introduced the original concept of Hyper Simplicial Complex (HySC) that codifies a MSM in $\mathbb{E}^{d}$ as a simplicial complex in $\mathbb{E}^{d+1}$. This approach permits us to define data structures and algorithm for the management of the global topology of a MSM. In particular we have proposed an algorithm for the extraction of a variable resolution model from a HySC with the full face-adjacency topological relations. We think that the HySC interpretation still needs further work, in particular we think to give better proofs of correctness and worst-case complexity analysis of the presented algorithms; we will also investigate on the definition of algorithms for answering spatial interference queries using HySC.

In the last section we have given some details on the effective use of multiresolution, and in particular on the use of variable resolution representation in volume visualization.

The implementation of the presented Multiresolution techniques is still in an early stage of development. We think to complete this work and compare the implementation with other multiresolution techniques.

Chapter 7

Concluding Remarks

In this Thesis we have proposed the use of simplicial complexes as a framework for representing the geometric structure generated by the *modeling* step of the visualization process (cfr. Chapter 2). In particular we have focused our attention on the three-dimensional case by providing techniques, strategies and algorithms for the visualization modeling and rendering of volumetric scalar datasets represented by tetrahedral complexes. In summary, we can outline the main contributions of this work as follows:

- An original intepretation of the SciViz process as a two-step mapping problem: a *modeling* step in which data are mapped into geometries with visual attributes, and a *rendering* step where geometries are transformed into images (Section 2.1).

- The optimal solution to problem of the search of the cells crossed by an isosurface by the use of the interval tree data structure (Section 2.2.1).

- A new technique for sorting a complex belonging to the projective class. The approach is based on the preprocessing construction of the *lifted* complex and on its representation as a power diagram (Chapter 3).

- A run-time splitting technique for the decomposition of projected tetrahedra along isosurfaces; this technique is based on a tabular approach, is very efficient and allows the correct integration of DVR and isosurface (Section 4.2).

- The introduction of the concept of Discontinuous Transfer Function (DTF) which unifies the management of isosurfaces, interval volumes and direct volme rendering (Section 4.3). The problem of correctly rendering a DTF has been also addressed, by exploiting the splitting technique descrbed in Section 4.2.

- Two algorithms for the simplification of volume datasets represented by tetrahedral meshes with accurate control of the introduced error (Chapter 5). The two algorithm are based on the refinement and decimation strategies, respectively.

- Hyper Simplicial Complexes (HySC), an original interpretation of a Multiresolution Simplicial Complex in $\mathbb{E}^{d}$ as a complex in $\mathbb{E}^{d+1}$ (Section 6.4); a data structure for HySC and an algorithm for extracting a variable resolution representation from a HySC have been also presented.

Some of the ideas here presented were integrated into a tetrahedral volume visualization system called TAn (**T**etrahedra **An**alyzer) available in the public domain[1]. We are now developing the 2.0 version of our system that will include all the solution here presented.

In spite of the intensive research done on volume visualization and, in particular, on simplicial volume visualization, some issues still need further investigation. Now we sketch some of the areas in which we want to continue our work.

Power Diagram Sorting We think that the sorting approach proposed in Chapter 3 needs some more work in order to improve its practical relevance. In particular the lifting problem should have a solution better than the one we presented in order to be applied on common datasets. We are working in two directions to solve this problem. There is some evidence that we can substitute the Simplex algorithm with a specialized algorithm like the network simplex. The pivot operations of the simplex algorithm can be interpreted in terms of geometric modifications on the original simplicial complex. Another approach can be the definition of heuristic algorithms that build a lifted complex by lifting one vertex after the other, by checking the convexity of the resulting complex only locally.

Moreover the Power Diagram sorting approach also suggest a strategy for the management of non-projective, and perhaps cyclic, complexes. We plan to investigate into heuristics which transform a partially-convex lifted complex, resulting for example by the failure of the simplex algorithm, into a convex one. Such a process can be performed by adding new points onto the non-convex (d-1)-facets of the partially lifted complex, splitting them and trying again to lift the new complex.

[1]SGI executables of the TAn system are freely downloadable on our web site`http://miles.cnuce.cnr.it/cg/swOnTheWeb.html`.

Tetrahedral Mesh Simplification The implementation of the decimation algorithm, proposed in Chapter 5, is still under developement at writing time. We think that better strategies for evaluating the *future* error resulting from a vertex removal could be suggested from the result of this implementation.

The memory requirements of these algorithms can be very high, so we are planning to design a decimation algorithm working on very large meshes by applying *windowing* strategies to advance (and decimate) only a portion of the dataset at a time, while mantaining the mesh correctness.

Multiresolution Models We plan to implement and compare the two multiresolution models for tetrahedral complexes described in Chapter 6. One of the two multiresolution models will be integrated in the new release of the TAn visualization system. We will use the MagicSphere tool for specifying space-based focus areas and Discontinuous Transfer Functions for range-based focus vaules in order to extract variable-resolution models which fit the user needs.

The memory requirements for multiresolution models are generally very high, therefore we are also interested in studying the use of I/O optimal techniques and data structures (like the Buffer Tree [5]) for the management of the multiresolution model on secondary memory.

Using these techniques we hope to succeed in efficiently mantaining only the low resolution part of the multiresolution model in main memory, and accessing to multiresolution model stored on the secondary memory only for the high resolution parts of the specified by the focus area.

Flow Visualization Our thesis dealt only with scalar field visualization, but many applications manage and require the visualization of vectorial data. The inclusion of this kind of data in simplification algorithms and multiresolution model is straightforward: it is sufficient to change the error evaluation functions.

An interesting problem is the definition of algorithms for the construction of streamlines working on multiresolution models, using for example variable resolution models where the resolution follows the *turbolence* of the flow.

We are also investigating the possibility of including some visualization technique, like illuminated field lines [96] in the next release of TAn.

Tetrahedron as a Graphics Primitive Finally, we wonder if the state of Volume Visualization is stable enough to include volume primitives into graphics systems, either software libraries and toolkits or hardware subsystems. In the case of regular volume datasets, this has been recently made possible by a slight modification of the graphics subsystems. The adoption of new rendering approaches, based on hardware texture mapping and trilinear interpolation, produced impressive speedups to voxel-based applications (e.g. medical imaging). Will it be possible in the near future to have a hardware support for the efficient rendering of tetrahedral volume primitives and its inclusion in standard graphics libraryΓ

Bibliography

[1] P. K. Agarwal, M. J. Katz, and M. Sharir. Computing depth orders and related problems. *Comp. Geom. Theory and Appl.*, 5:187–206, 1995.

[2] P.K. Agarwal and J. Matousek. Ray shooting and parametric search. *Siam J. Computing*, 22(4):794–806, 1993.

[3] P.K. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings 5th ACM-SIAM Symposium On Discrete Algorithms*, pages 24–33, 1994.

[4] Guy Albertelli and Roger A. Crawfis. Efficient subdivision of finite-element datasets into consistent tetrahedra. In *Volume Visualization 97*. IEEE, October 1997. ISBN 0-89791-741-3.

[5] Lars Arge. The buffer tree. a new technique for optimal I/O-algorithms. In *Proc. European Symp. Algorithms*, 1995.

[6] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *Siam J. Comput.*, 16(1):78–96, February 1987.

[7] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. Fast isocontouring for improved interactivity. In *1996 Volume Visualization Symposium*, pages 39–46. IEEE, October 1996. ISBN 0-89791-741-3.

[8] Chandrajit L. Bajaj and Daniel R. Schikore. Visualization of scalar topology for structural enhancement. Technical Report 96-006, Purdue University, Jan 1996.

[9] Chandrajit L. Bajaj and Daniel R. Schikore. Topology preserving data simplification with error bounds. *Computers & Graphics*, 22(Γ):(in press), 1998.

[10] Thomas F. Banchoff. *Beyond the Third Dimension.* W. H. Freeman and Co., New York, 1990.

[11] David Banks. Interactive manipulation and display of two-dimensional surfaces in four-dimensional space. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):197–207, March 1992.

[12] M. Bertolotto, E. Bruzzone, L. De Floriani, and E. Puppo. Multiresolution Representation of Volume Data through Herarchical Simplicial Complexes. In *Aspects of Visual Form Processing*, pages 73–82. World Scientific, Singapore, 1994.

[13] M. Bertolotto, L. De Floriani, and P. Marzano. Pyramidal simplicial complexes. In *Proceedings 4th International Symposium on Solid Modeling*, pages 153–162, Salt Lake City, Utah, U.S.A., May 17-19 1995. ACM Press.

[14] E.A. Bier, M.C. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses: the see–through interface. In *Proceedings of SIGGRAPH '93(Anaheim, CA, August 1-6). In Computer Graphics Proceedings, Annual Conference series, ACM SIGGRAPH*, pages 73–80, 1993.

[15] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (SIGGRAPH '82 Proceedings)*, 16(3):21–29, July 1982.

[16] B. Chazelle, H. Edelsbrunner, L.J. Guibas, R. Pollak, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. *Computational Geometry: Theory and Applications*, 1:305–323, 1991.

[17] Y. Chiang and C. T. Silva. I/o optimal isosurface extraction. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97 Proceedings*, page (in press). IEEE Press, 1997.

[18] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, June 1997.

[19] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and rendering of volume data based on simplicial complexes. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 19–26. ACM Press, October 17-18 1994.

[20] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *Proceedings of IEEE/ACM 1996 Symposium on Volume Visualization*, pages 31–38. ACM Press, October 28-29 1996.

[21] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data. *IEEE Trans. on Visualization and Comp. Graph.*, 3(4):(in press), 1997.

[22] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Trans. on Visualization and Comp. Graph.*, 3(2):158–170, 1997.

[23] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. Resolution modeling. Technical Report C97-02, CNUCE – C.N.R., Pisa, Italy, January 1997.

[24] P. Cignoni, C. Montani, D. Sarti, and R. Scopigno. On the optimization of projective volume rendering. In P.Zanarini R. Scateni, J.J. van Wijk, editor, *Visualization in Scientific Computing 1995*, pages 58–71. Springer KG, Wien, 1995.

[25] P. Cignoni, C. Montani, and R. Scopigno. MagicSphere: an insight tool for 3D data visualization. *Computer Graphics Forum*, 13(3):317–328, 1994. (Eurographics '94 Conf. Proc.).

[26] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(Γ):(in press), 1998.

[27] P. Cignoni, E. Puppo, and R. Scopigno. Representation and visualization of terrain surfaces at variable resolution. In R. Scateni, editor, *Scientific Visualization '95 (Proc. Inter. Symposium on)*, pages 50–68. World Scientific, 1995.

[28] P. Cignoni, E. Puppo, and R. Scopigno. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer*, 13(5):199–217, 1997. (A preliminary version appeared on "Scientific Visualization '95", Proc. Inter. Symposium, World Scientific, pp.50-68).

[29] Mark de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal.* Number 703 in Lecture Notes in Computer Science. Springer-Verlag, 1993.

[30] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97*, Phoenix, AZ (USA), October 1997. (to appear).

[31] L. De Floriani, P. Magillo, and E. Puppo. Efficient encoding and retrieval of triangle meshes at variable resolution. Technical Report DISI-TR-97-01, Department of Computer and Information Sciences, University of Genova, Genova, Italy, January 1997.

[32] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, 1996.

[33] L. De Floriani and E. Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, October 1995.

[34] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In R. Klein, W. Straßer, and R. Rau, editors, *Theory and Practice of Geometric Modeling*. Springer-Velrag, 1997. (to appear).

[35] M. Deering. Geometry compression. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '95), ACM Press*, pages 13–20, 1995.

[36] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three–dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[37] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 65–74, August 1988.

[38] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Report F59, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1980.

[39] H. Edelsbrunner. An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10:251–260, 1990.

[40] Herbert Edelsbrunner and Ernst P. Mücke. Three-Dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994. ISSN 0730-0301.

[41] J. Foley, A. van Dam, S. Feiner, J. Hugues, and R. Phillips. *Introduction to Computer Graphics.* Addison Wesley, 1993.

[42] Thomas A. Foley, David A. Lane, Gregory M. Nielson, Richard Franke, and Hans Hagen. Interpolation of scattered data on closed surfaces. *Computer Aided Geometric Design*, 7(1-4):303–312, June 1990.

[43] Thomas A. Foley and Gergory M. Nielsen. Modeling of scattered multivariate data. In C. Giertsen and P. A. Fevang, editors, *State of the Art Reports of Eurographics '94*, pages 38–59, 1994.

[44] R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *ACM Computer Graphics (Siggraph '79 Proc.)*, 13(3):199–207, Aug. 1979.

[45] H. Fuchs, Z.M. Kedem, and B.F. Naylor. On visible surface generartion by a priori tree structures. *ACM Computer Graphics*, 14:124–133, Aug. 1980.

[46] I. Fujishiro, Y. Maeda, and H. Sato. Interval volume: a solid fitting technique for volumetric data display and analysis. In *Visualization '95*, pages 151–158. IEEE Com. Soc. Press, 1995.

[47] R. S. Gallagher. Span filter:an optimization scheme for volume visualization of large finite element models. In *IEEE Visualization'91 Proc.*, pages 68–75, 1991.

[48] Michael P. Garrity. Raytracing irregular volume data. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):35–40, November 1990.

[49] Christopher Giertsen. Volume visualization on sparse irregular meshes. *IEEE Computer Graphics & Applications*, pages 40–48, March 1992.

[50] M. Giles and R. Haimes. Advanced interactive visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[51] B. Guo. Interval sets: a volume rendering technique generalizing isosurface extraction. In *Visualization '95 Proceedings*, pages 3–10. IEEE Computer Society Press, 1995.

[52] B. Hamann and J.L. Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11:477–489, 1994.

[53] Andrew J. Hanson and Pheng A. Heng. Illuminating the fourth dimensions. *Computer Graphics & Applications*, 12(4):54–62, 1992.

[54] H. Hoppe. Progressive meshes. In *ACM Computer Graphics Proc., Annual Conference Series, (Siggraph '96)*, pages 99–108, 1996.

[55] Victoria Interrante. Illustrating surface shape in volume data via principal direction-drive 3d line integral convolution. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '97), ACM Press*, 1997. (to appear).

[56] T. Itoh and K. Koyamada. Automatic isosurface propagation using an Extrema Graph and sorted boundary cell lists. *IEEE Trans. on Vis. and Comp. Graph.*, 1(4):319–327, 1995.

[57] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[58] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 165–174, July 1984.

[59] T. Kao, D. Mount, and A. Saalfeld. Dynamic maintenance of delaunay triangulations. In *Proceedings Auto-Carto 10*, pages 219–233, 1991.

[60] A. Kaufman. *Volume Visualization.* IEEE Computer Society Press, Los Alamitos, CA, 1990.

[61] Arie Kaufman, Daniel Cohen, and Roni Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, July 1993.

[62] R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In R. Yagel and G. Nielson, editors, *Proceedings of Visualization '96*, pages 311–318, 1996.

[63] Michael J. Laszlo. Fast generation and display of isosurface wireframes. *CVGIP: Graphical Models an Image Processing*, 54(6):473–483, November 1992.

[64] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In Thomas W. Sederberg, editor, *Computer Graphics 25(4) (SIGGRAPH '91 Proceedings)*, pages 285–288, July 1991.

[65] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[66] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[67] 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. In *Comp. Graph. Proc., Annual Conf. Series (Siggraph '96), ACM Press*, pages 109–118, New Orleans, LA, USA, Aug. 6-8 1996.

[68] Y. Livnat, H.V. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm for structured and unstructured grids. *IEEE Trans. on Vis. and Comp. Graph.*, 2(1):73–84, 1996.

[69] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–170, July 1987.

[70] Xiaoyang Mao, Lichan Hong, and Arie Kaufman. Splatting of curvilinear volumes. In *IEEE Visualization '95 Proc.*, pages 61–68, 1995.

[71] Nelson Max. Optical models for direct volume rendering. *IEEE Trans. on Vis. and Comp. Graph.*, 1(2):99–108, 1995.

[72] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3D scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, November 1990.

[73] C. Montani, R. Perego, and R. Scopigno. Parallel rendering of volumetric dataset on distributed memory architectures. *Concurrency: Practice and Experience*, 5(2):153–167, 1993.

[74] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. In R.D. Bergeron and A.E. Kaufman, editors, *Visualization '94 Proceedings*, pages 281–287. IEEE Computer Society Press, 1994.

[75] S. Muraki. Multiscale volume representation by a dog wavelet. *IEEE Trans. on Vis. and Comp. Graph.*, 1(2):109–116, June 1995.

[76] M. E. Newell, R. G. Newell, and T. L. Sancha. A new approach to the shaded picture problem. In *Proc. ACM Natl. Conf.*, pages 443–450, 1972.

[77] Gregory M. Nielson and Juewon Sung. Interval volume tetrahedrization. In *Volume Visualization 97.* IEEE, October 1997. ISBN 0-89791-741-3.

[78] O. Nurmi. On translating a set of objects in two- and three-dimensional space. *Computer Vision, Graphics and Image Processing*, 24:42–52, 1985.

[79] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993.

[80] M. S. Paterson and F.F. Yao. Efficient binary partitions for hidden surfac removal and solid modeling. *Discrete and Computational Geometry*, 1990.

[81] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *ACM Computer Graphics Proc., Annual Conference Series, (Siggraph '97)*, pages 217–224, 1997.

[82] T. Porter and T. Duff. Compositing digital images. *ACM Computer Graphics (Proc. of SIGGRAPH '84)*, 18:253–259, 1984.

[83] F.P. Preparata and M.I. Shamos. *Computational Geometry - An Introduction*. Springer-Verlag, 1985.

[84] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry, Ottawa, Canada*, pages 202–210, August 12-15 1996.

[85] K.J. Renze and J.H. Oliver. Generalized unstructured decimation. *IEEE C.G.&A.*, 16(6):24–32, 1996.

[86] Michael L. Rhodes. Computer graphics in medicine. *IEEE Computer Graphics & Applications*, 13(6), 1993.

[87] J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete Comput. Geom.*, 7:227–253, 1992.

[88] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):51–58, 1988.

[89] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *ACM Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[90] H. Shen and C.R. Johnson. Sweeping simplices: a fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization '95 Proc.*, pages 143–150, 1995.

[91] J. R. Shewehuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, May 1996.

[92] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):63–70, November 1990.

[93] Claudio T. Silva and Joseph S. B. Mitchell. The lazy sweep ray casting a.lgorithm for rendering irregular grids. Technical Report 11794/3600, State University of New York, Stony Brook, 1997.

[94] C.T. Silva, J. Mitchell, and A. Kaufman. Fast rendering of irregular grids. In R. Crawfis and C. Hansen, editors, *Proceedings 1996 Symp. on Volume Visualization (Oct. 28-29)*, pages 15–22, 1996.

[95] Don Speray and Steve Kennon. Volume probes: Interactive data exploration on arbitrary grids. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):5–12, November 1990.

[96] Detlev Stalling, Malte Zockler, and Hans-Christian Hege. Fast display of illuminated field lines. *IEEE Trans. on Visualization and Computer Graphics*, 3(2):118–128, 1997.

[97] C. Stein, B. Becker, and N. Max. Sorting and Hardware Assisted Rendering for Volume Visualization. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 83–90. ACM Press, October 17-18 1994.

[98] P. Su and R. L. Scot Drysdale. A comparison of sequential delaunay triangulation algorithms. In *11th ACM Computational Geometry Conf. Proc. (Vancouver, Canada)*, pages 61–70. ACM Press, 1995.

[99] K. Udupa, Jayaram and Dewey Odhner. Shell rendering. *IEEE C.G.&A.*, 13:58–67, November 1993.

[100] S. M. Pizer V. Interrante, H. Fuchs. Conveying the 3d shape of smoothly curving transparent surfaces via texture. *IEEE Trans. on Visualization and Comp. Graph.*, 3(2):98–117, 1997.

[101] A. van Gelder and J. Wilhelms. Rapid exploration of curvilinear grids using direct volume rendering. In *IEEE Visualization '93 Proceedings (Oct 25-29, San Jose CA)*, pages 70–77, 1993.

[102] M. van Kreveld. Efficient methods for isoline extraction from a digital elevation model based on triangulated irregular networks. Technical Report UU-CS-1994-21, Dep. Computer Science, Utrecht University, November 1994.

[103] M. van Kreveld, R van Oostrum, Chandrajit L. Bajaj, Daniel R. Schikore, and V. Pascucci. Contour trees and small seed sets for isosurface traversal. In *Proceedings of Thirteen ACM Symposium on Computational Geometry*, pages 212–219, Nice, France, june 4-9 1997. ACM press.

[104] R. Westermann. A Multiresolution Framework for Volume Rendering. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 51–58. ACM Press, October 17-18 1994.

[105] R. Westermann and T. Ertl. The vsbuffer: visibility ordering of unstructured volume primitives by polygon drawing. In editors ???, editor, *Proceedings of '97 Visualization*, page ???, 1997.

[106] L. Westover. Interactive volume rendering. In *Chapel Hill Workshop on Volume Visualization*, pages 9–16, May 1989.

[107] L. Westover. Footprint evaluation for volume rendering. *ACM Computer Graphics*, 24(4):367–376, July 1990.

[108] J. Wilhelms and A. van Gelder. A coherent projection approach for direct volume rendering. *ACM Computer Graphics*, 25(4):275–284, July 1991.

[109] J. Wilhelms and A. van Gelder. Multi-dimensional Trees for Controlled Volume Rendering and Compression. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 27–34. ACM Press, October 17-18 1994.

[110] J. Wilhelms, A. Van Gelder, P. Tarantino, and J. Gibbs. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. In R. Yagel G. Nielson, editor, *Visualization '96 Proceedings*, pages 57–64. IEEE Press, 1996.

[111] Jane Wilhelms and Allen van Gelder. Octrees for faster isosurface generation. *ACM Transaction on Graphics*, 11(3):201–227, July 1992.

[112] Peter L. Williams. Visibility ordering of meshed polyhedra. *ACM Transaction on Graphics*, 11(2):103–126, April 1992.

[113] Peter L. Williams and Nelson Max. A volume density optical model. *1992 Workshop on Volume Visualization*, pages 61–68, 1992.

[114] P.L. Williams. *Interactive Direct Volume Rendering of Curvilinear and Unstructured Data.* PhD thesis, University of Illinois at Urbana–Champaign, 1993.

[115] Peter L. Willias, Nelson L. Max, and Clifford M. Stein. A high accuracy volume renderer for unstructured data. Technical Report UCRL-JC-126942, L.L.N.L. Livermore, CA, September 1997.

[116] R. Yagel, D.M. Reed, A. Law, Shi P.W., and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. In R. Crawfis and C. Hansen, editors, *Proceedings 1996 Symp. on Volume Visualization (Oct. 28-29)*, pages 55–62, 1996.

[117] J. Zhou, N. M. Patrikalakis, S. T. Tuohy, and X. Ye. Scattered data fitting with simplex splines in two and three dimensional spaces. *The Visual Computer*, 13:295–315, 1997.

[118] Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing volume data. In *IEEE Visualization '97 Proceedings*, page (in press). IEEE Press, 1997.

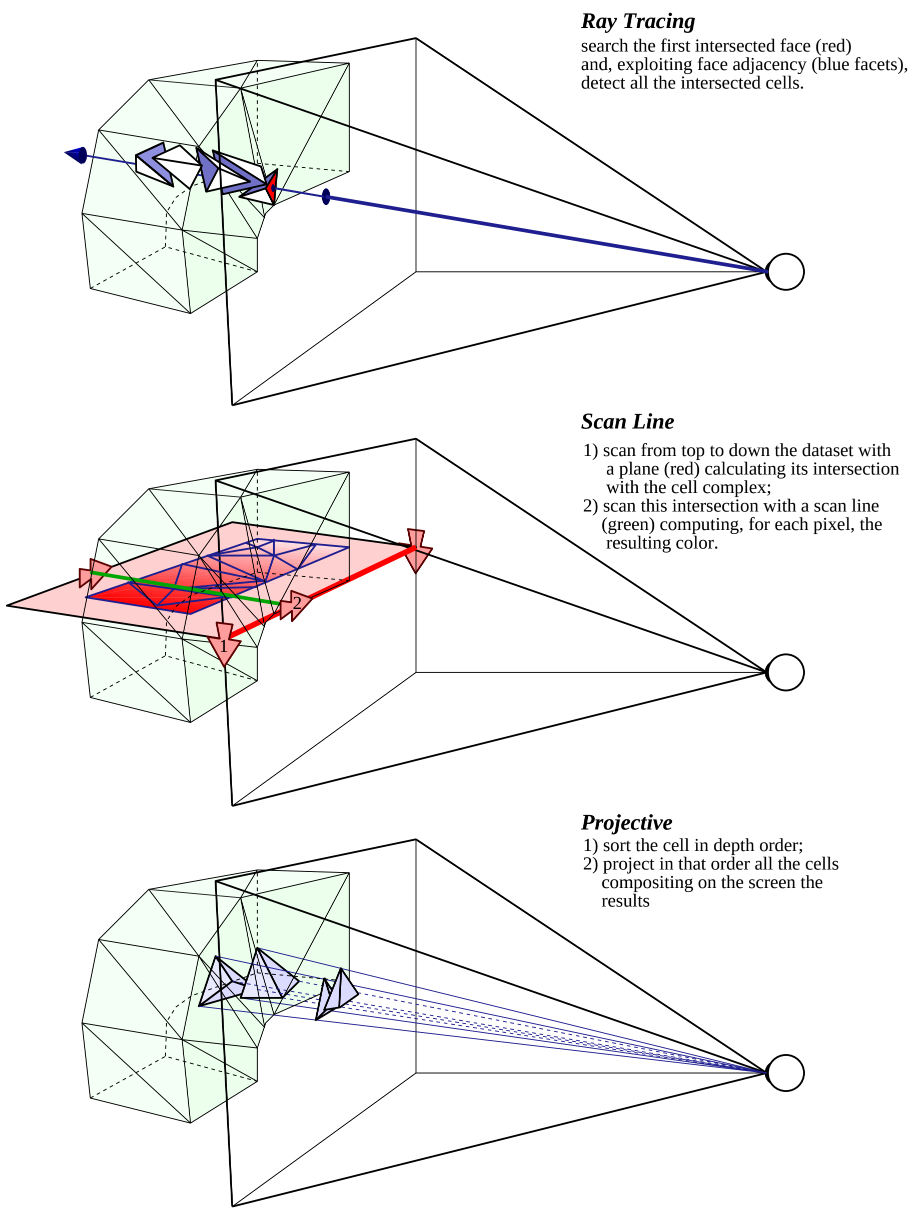

Ray Tracing
search the first intersected face (red)
and, exploiting face adjacency (blue facets),
detect all the intersected cells.
Scan Line
1) scan from top to down the dataset with
a plane (red) calculating its intersection
with the cell complex;
2) scan this intersection with a scan line
(green) computing, for each pixel, the
resulting color.
1
2
Projective
1) sort the cell in depth order;
2) project in that order all the cells
compositing on the screen the
results

Figure 7.1: Tetrahedral direct volume rendering approaches (Section 2.3).

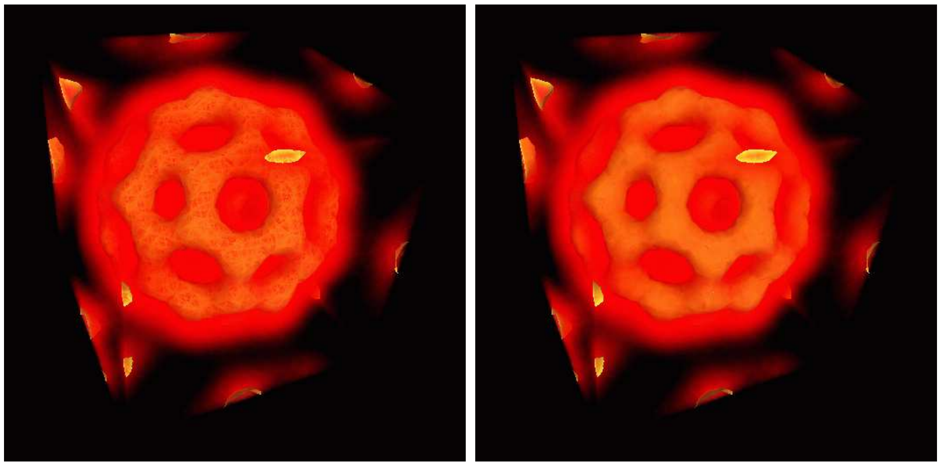


Figure 7.2: Trivial Integration of DVR and Isosurfaces (left) and the correct one (right). See Chapter 4.

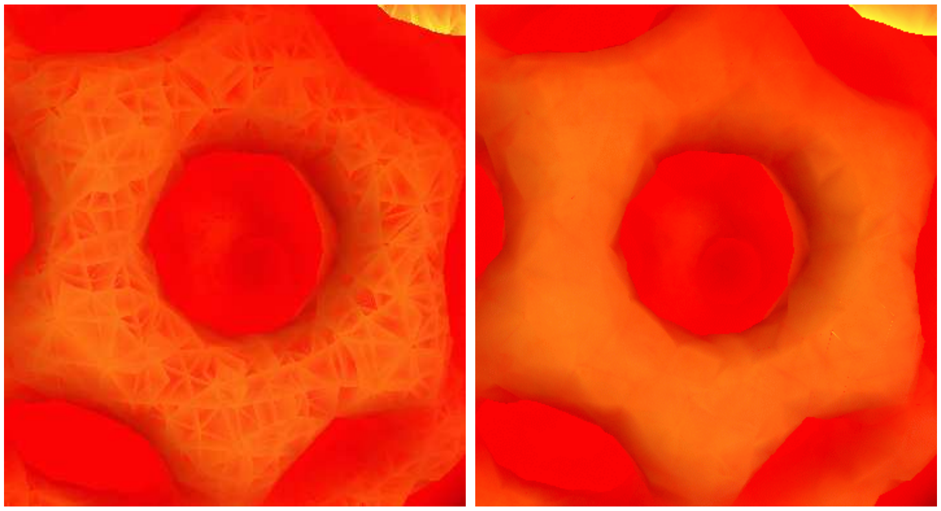


Figure 7.3: A zoomed in portion of the comparision between trivial integration of DVR and Isosurfaces (left) and the correct one (right)

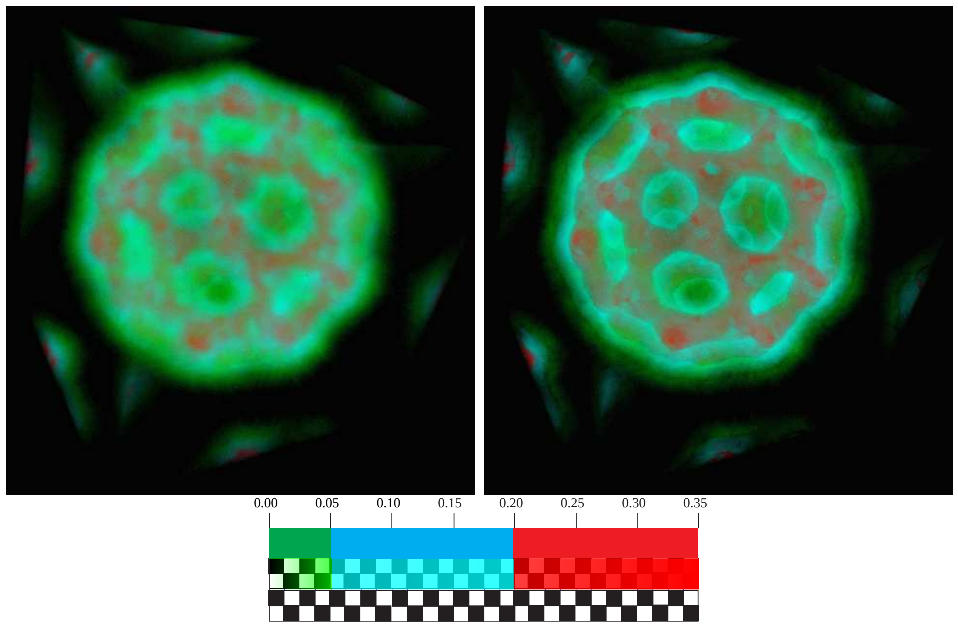


Figure 7.4: DTF rendering: without splitting on C^0 discontinuities (left) and using our approach (right). The DTF is shown in the lower part of the figure. See Section 4.3.

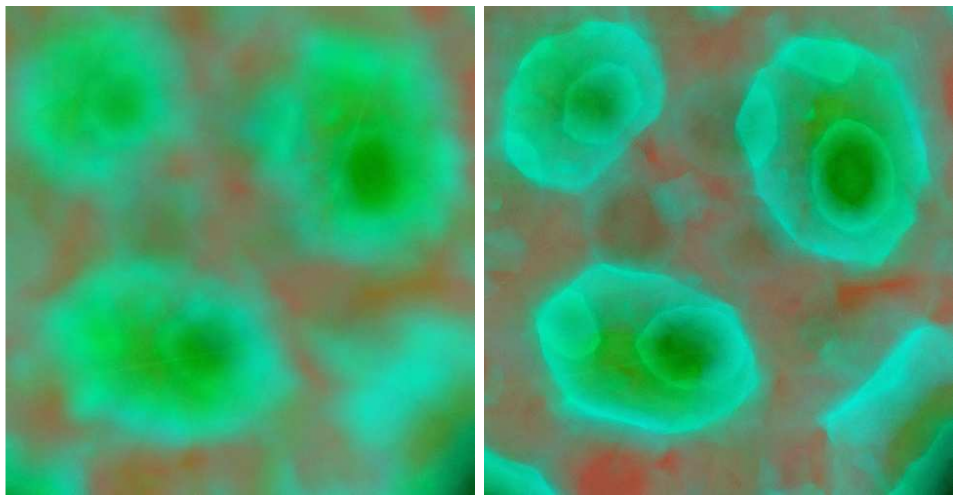


Figure 7.5: A zoomed in portion of the DTF rendering of figure 7.4. On the left the approximate rendering and on the right the correct one.

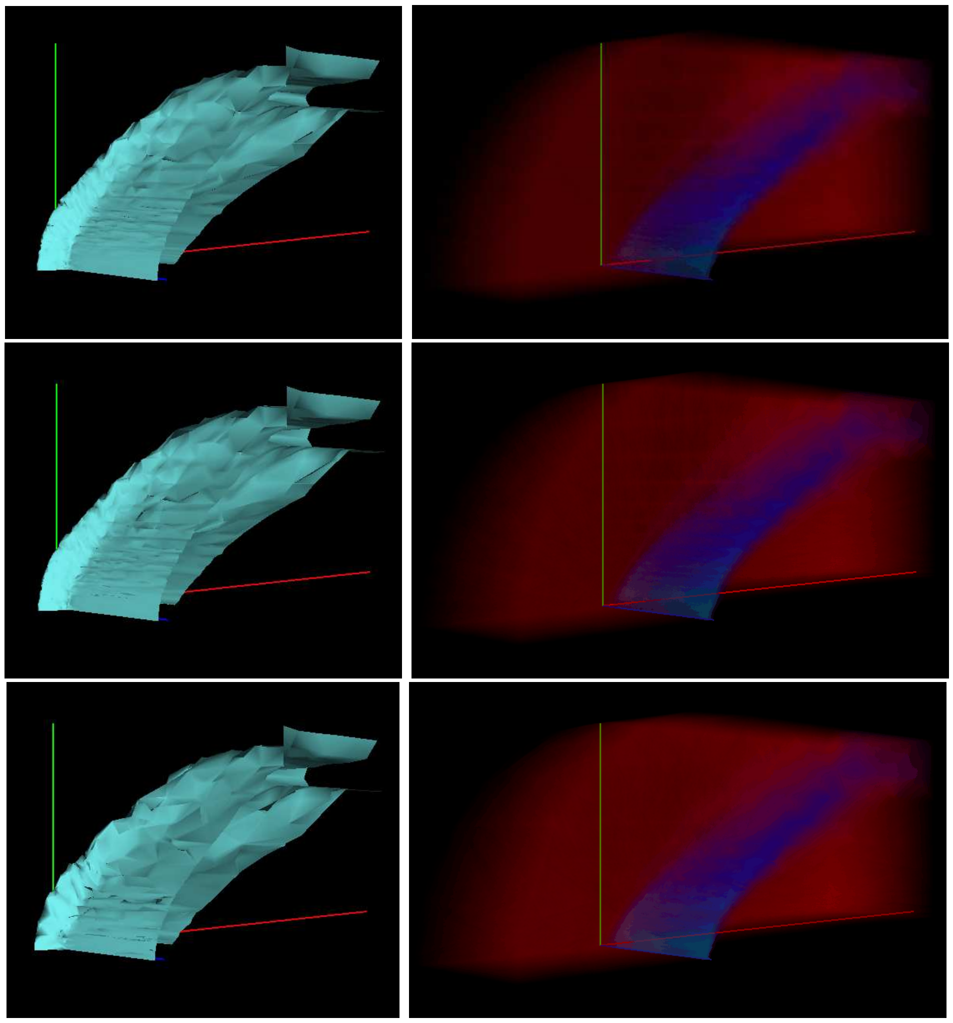


Figure 7.6: Isosurface visualization and direct volume rendering of the BluntFin dataset simplified with the refinement approach. The dataset is shown at three different resolutions, from top to bottom: $(\delta, \varepsilon) = (0,0)$,$(\delta, \varepsilon) = (1.0\%, 1.0\%)$ and $(\delta, \varepsilon) = (4.0\%, 4.0\%)$. See Section 5.

Accuracy	no. vertices	no. tetra	no. iso. triangles	DVR time
(0.0%,0.0%)	40,960	222,528	19,499	44.1
(1.0%,1.0%)	14,162	80,883	9,143	16.1
(4.0%,4.0%)	3,612	20,324	3,442	3.9

Figure 7.7: Numerical values for isosurface (thr.= 1.244), and direct volume renderings of Figure 7.6. Times are in seconds on an SGI Indigo R4000.