Multiresolution Representation and Visualization of Volume Data

Paolo Cignoni, Claudio Montani, *Member, IEEE*,
Enrico Puppo, *Member, IEEE*, Roberto Scopigno, *Member, IEEE*

P. Cignoni, and C.Montani are with the Istituto di Elaborazione dell'Informazione - CNR, Via S. Maria 46, Pisa, Italy. E-mail: {cignoni,montani}@iei.pi.cnr.it.
E. Puppo is with the Istituto per la Matematica Applicata - CNR, Via De Marini 6, Genova, Italy. E-mail: puppo@ima.ge.cnr.it.
R. Scopigno is with the Istituto CNUCE - CNR, Via S. Maria 36, Pisa, Italy. E-mail: r.scopigno@cnuce.cnr.it.

Abstract

A system to represent and visualize scalar volume data at multiple resolution is presented. The system is built on a multiresolution model based on tetrahedral meshes with scattered vertices that can be obtained from any initial dataset. The model is built off-line through data simplification techniques, and stored in a compact data structure that supports fast on-line access. The system supports interactive visualization of a representation at an arbitrary level of resolution through isosurface and projective methods. The user can interactively adapt the quality of visualization to requirements of a specific application task, and to the performance of a specific hardware platform. Representations at different resolutions can be used together to enhance further interaction and performance through progressive and multiresolution rendering.

Index Terms – Volume data visualization, multiresolution representation, tetrahedral meshes.

I. Introduction

Volume datasets used in current applications have different characteristics, but a common problem: a huge size, which affects both storage requirements and visualization times. This issue is especially important with curvilinear and irregular datasets, where the mesh topology must be stored explicitly for visualization purposes [40]. Therefore, in some cases interactive image generation from very large datasets may not be feasible, even with the use of fast graphic hardware and parallelism.

In recent years, some efforts have been devoted in the literature towards improving performance of rendering algorithms, but few proposals are based on *data simplification*, which, on the other hand, has produced successful results in managing surface data complexity (e.g. free-form and topographic surfaces representation).

In this paper, we describe our experience in designing and developing a volume visualization system that can handle data at different resolutions, and that is based on a data simplification approach.

A. Related work

In the literature, *dataset complexity* has been carefully taken into account to reduce expected visualization times. Performance has been improved through different methods: *ad hoc data organizations* permit to speedup operations that visit the dataset during rendering [11], [23], [41], [6]; *simplification of the rendering process* can be achieved either by approximation techniques [43], [40], [45], or by reducing the size of the graphic output [37], [27], [10], [19].

On a different perspective, it is also possible to manage data complexity by adopting an *approximated representation* of the dataset. Such an approach is more general because, given a suitable strategy to reduce the size of the dataset, it remains totally independent of the rendering system. The methodology in this case is therefore to work on *data simplification* rather than on *graphics output simplification.*

A naive *subsampling* from a regular dataset has several drawbacks: there is no control on the accuracy of the simplified mesh; the technique is not adaptive, i.e. density of data cannot be variable over different regions of the domain; and it is not easily extensible to datasets that are not regular. In fact, an irregular distribution of samples makes the construction of a simplified dataset a non-trivial problem in general. Adaptive methods have been developed in 2D for the simplification of irregular meshes representing free-form and topographic surfaces: effective solutions have been obtained through incremental techniques, based on either refinement or simplification (see, e.g., [10], [14], [17], [21], [24], [37]). Some of such techniques can be extended to the 3D case to simplify volume data [5], [18], [34].

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, at different levels of accuracy, is called a *level of detail* (*LoD*) representation. LoD representations of surfaces and solid objects are widely used in a number of leading edge applications (e.g., virtual reality based on VRML). An evolution of a LoD representation is a *multiresolution* representation, which supports the compact storage of a number m (usually large) of representations at different levels of detail, where m is a monotone 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. For instance, 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, when simplification occurs. Many approaches have been recently proposed for the multiresolution management of surfaces (see, e.g., [33] for a survey), while multiresolution volume data management is still in a not sufficiently developed stage.

An approach to the representation of regular volume datasets based on the use of a hierarchical recursive partition (an octree-like scheme) has been proposed in [42]. Each node is obtained by recursive subdivision: it holds a basis function to reconstruct the field, as well as a measure of both error and importance factors, which are used for selective traversal of the tree. The method cannot be extended to irregularly distributed data. Using such a structure as a *LoD* representation, by considering each tree level as a separate layer, is equivalent to use subsampling. A *multiresolution* representation is also possible, by selecting nodes at different levels, but the field may result discontinuous across different levels, thus causing unpleasant effects (e.g., aliasing in direct volume rendering, and cracks in isosurfaces).

In a previous paper [5], we proposed a *LoD* representation based on tetrahedral decomposition: independent simplified representations of a volume dataset at different levels of approximation were built by a refinement technique. Such a work can be considered preliminary to that presented in this paper, and it is extended here in several aspects.

Finally, some approaches for the hierarchical representation of regular tetrahedral decompositons have been recently proposed [15], [29], [47].

Wavelet theory plays an important role in the multiresolution analysis of signals, and approaches based on wavelets have been proposed also to manage volume data [16], [28], [39]. The approach to data simplification based on wavelets is much different from the geometric approach we follow. Data are considered as samples from a signal that is decomposed into *wavelets* [26]: the coefficients of the wavelet decomposition represent the dataset at full resolution, while approximated (*LoD*-style) representations may be used in rendering by considering only subsets of the coefficients. The wavelet decomposition may also be used in a multiresolution manner by using higher resolution coefficients in limited locations of the 3D space only. Times for wavelet-based rendering are generally higher than those of standard cell/voxel-based techniques and, moreover, generality is limited because the wavelet approach has been applied to regular datasets only.

B. Summary

The paper consists essentially of two parts. In the first part (Sections II-IV) we show how a multiresolution model for volume data based on tetrahedral meshes can be built and stored. In the second part, (Sections V-VI) we describe a volume visualization system built on top of such a model, and we present experimental results.

Our approach to multiresolution, is based on *data simplification*, which is described in Section II: an approximated representation of volume data at reduced resolution is given by a tetrahedral mesh, having smaller size with respect to an initial mesh defined on the whole dataset. Data values are approximated by a linear function over each tetrahedron. Tetrahedral meshes are used because of their adaptivity (local refinement) and for the simplicity of linear interpolation.

In Section III, two methods for building approximated meshes are described: a top-down method that refines a coarse initial mesh by iteratively inserting vertices, and a bottom-up method that simplifies an initial mesh at the highest resolution by iteratively discarding vertices. The top-down method extends a previous result that we presented for convex datasets in [5], to handle also curvilinear (possibly non-convex) datasets. The bottom-up method extends simplification methods in 2D [19], [37], and it can be applied also to irregular non-convex datasets.

Since both methods are based on iterative local modifications of a mesh, each of them produces a fine-grained sequence of meshes at increasingly finer (respectively, coarser) resolution. In other words, a high number of different tetrahedral meshes at different resolutions are obtained on the basis of a moderate number of tetrahedra, namely all tetrahedra that appear during successive updates. Such tetrahedra can be stored in a compact representation of a multiresolution model, described in Section IV, which supports fast on-line extraction of a mesh at arbitrary resolution.

In Section V we describe the multiresolution visualization system *TAn (Tetrahedra Analyser)*, whose prototype is available in the public domain. Besides supporting the off-line construction of the multiresolution model, TAn has direct on-line access to the model itself: it allows the user to interactively

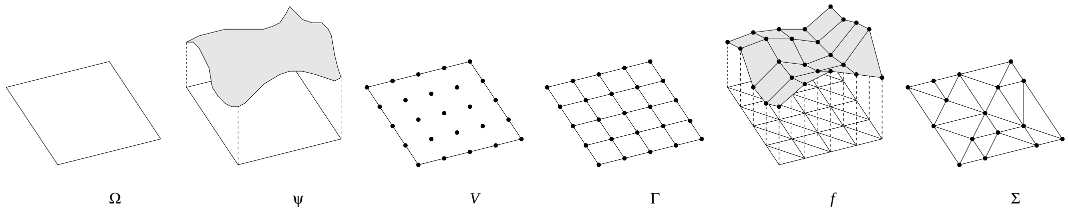

Fig. 1. A visualization of the terminology used, in a two-dimensional example.

select the resolution of representation, and the transfer function; it supports multiple isosurface fitting, direct volume rendering through projection, and approximated hybrid rendering; moreover, it supports interactive manipulation of huge volume data through progressive rendering, which is obtained by using representations at different resolutions from the multiresolution model.

Experimental results on the construction of the multiresolution model, on multiresolution visualization, and on the use of TAn are reported in Section VI.

In Section VII, concluding remarks are drawn, and current and future work on this subject is summarized.

II. Volume data approximation

A scalar volume dataset is given by the values of a scalar field ψ taken at a finite set of sample points V in $\mathbb{R}^3$. A volume $\Omega \subset \mathbb{R}^3$ spanned by the points of V is called the *domain* of the dataset: Ω is usually a polyhedron, it can be either convex or non-convex, possibly with cavities. In most cases, a three-dimensional mesh Γ is also given, which covers the domain Ω, and has its vertices at the points of V: the scalar field ψ is estimated over Ω by a function f that interpolates all data values at points of V, and is defined piecewise on the cells of Γ. The terminology introduced is visually represented in Figure 1 where we present, for the sake of simplicity, a 2D example: in this case, Ω is a square region, Γ is a triangulation, V is the set of vertices of Γ, the graph of ψ is a surface in 3D, and the graph of f is a corresponding triangulated approximation.

A. Volume data classification

Volume data can be classified through the characteristic structure of the underlying grid.

- In *regular datasets*, sample points are distributed regularly in 3D space: Ω is a block (parallelepiped) and Γ is a regular hexahedral mesh.
- In *curvilinear datasets*, sample points lie on a regular grid in a computational space, while the grid is warped to become curvilinear in physical space: Ω is a polyhedron (usually non-convex), and Γ has the connective topology of a hexahedral mesh, while its cells are irregular convex hexahedra.
- In *irregular datasets*, sample points are irregularly distributed in 3D space: Ω can be either convex or non-convex, and Γ is usually a tetrahedral mesh, or a hybrid mesh made of tetrahedra and irregular hexahedra.

- In *scattered datasets* (sometimes also said *unstructured*), only sample points of V are known, which are irregularly distributed in 3D space, while Γ must be reconstructed. In the simplest case, Ω can be assumed coincident with the convex hull of V, therefore Γ may be obtained as a tetrahedrization of the points of V. A more general non-convex situation may require specific reconstruction techniques that are beyond the scope of this work.

Hereafter, we will always assume that Γ is given, and we will use the following (non-standard) classification of datasets, which is suitable to our purpose: *convex* (i.e., having a convex domain, disregarding any further classification of data distribution and type of mesh); *non-convex curvilinear;* and *non-convex irregular.*

B. Tetrahedral meshes

A tetrahedral mesh is a collection of tetrahedra such that for any pair of tetrahedra either they are disjoint, or they meet at a common vertex, or edge, or triangular face. This establishes topological relationships, essentially incidences and adjacencies, among the vertices, edges, triangular faces, and tetrahedra that form the mesh. As a convention, a tetrahedral mesh will be usually denoted by Σ, and a generic tetrahedron by σ.

Given a set of points V, a tetrahedral mesh Σ having its vertices at the points of V and covering the convex hull of V is called a *tetrahedrization* of V. Many different tetrahedrizations of V exist. In particular, the *Delaunay tetrahedrization* has the property that the circumsphere of each tetrahedron does not contain any point of V in its interior. The Delaunay tetrahedrization has some nice properties ("fat" cells, acyclicity in depth sort [13]), which make it a suitable mesh in the applications [44].

Given a polyhedron Ω, a tetrahedral mesh Σ covering it is also called a *tetrahedrization* of Ω. If Ω is non-convex, a tetrahedrization of Ω having vertices only at the vertices of Ω does not necessarily exist. Moreover, deciding whether such a tetrahedrization exists or not is NP-complete [35]. This suggests how the non-convex case is more difficult to handle, and it justifies the application of heuristics.

Given a tetrahedral mesh Σ with data values at its vertices, it is easy to interpolate such data by using a linear function within each tetrahedron. Therefore, piecewise-linear interpolation is most commonly used on tetrahedral meshes. Higher order interpolation would be necessary to achieve smoothness across different tetrahedra, but this involves high numerical effort which makes it hardly applicable to volume data. Discontinuities of the field represented by a tetrahedral mesh may be modeled by assigning different values to the same vertex for different tetrahedra incident into it.

C. 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 vertices at the subset V' of V, and covering a domain $\tilde{\Omega}$ that approximates Ω. A linear function is given for each tetrahedron of Σ. The *accuracy* of approximation is given by the difference between the reference model and the approximated model, which depends essentially on two factors:

- 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.

For *convex* datasets, we assume that $\tilde{\Omega} \equiv \Omega$, i.e., there is no warping, because convex datasets usually have a small number of vertices on their convex hull (e.g., the domain of a regular dataset is defined by six vertices).

For *non-convex curvilinear* datasets, we consider a parallelepiped Ω_c, called the *computational domain*, and a regular hexahedral mesh Γ_c covering Ω_c, and isomorphic to Γ. This is always possible because Γ is a deformed hexahedral mesh. 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 2a). Since Σ has vertices at 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 2b). 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 tetrahedron σ, image of σ_c through lifting. 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 2c). The maximum distance over all vertices of Γ whose back-projection lies inside σ_c estimates the warping of its lifted image σ; the maximum warping over all tetrahedra of Σ defines the warping of the whole approximated model.

For *non-convex irregular* datasets, we estimate the actual difference between the boundaries of Ω and $\tilde{\Omega}$. Such a difference is measured by computing at each boundary vertex of Γ its minimum (Hausdorff) distance from the boundary of Σ (see Figure 3).

The warping of a boundary face σ of Σ is the maximum among all distances corresponding to boundary vertices of Γ that are projected onto σ; the warping of Σ is the maximum among warping of its boundary faces [4].

Error is measured similarly. In a *convex* dataset, the error at a datum 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.

For a *non-convex curvilinear* dataset, the error is measured by computing the same difference in computational domain: this is equivalent to measuring the difference between the field at a datum v and the estimated value at its corresponding warped point $\tilde{v}$ defined above.

For *non-convex irregular* datasets there are two possible situations: if v is inside $\tilde{\Omega}$, then we compute the 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.

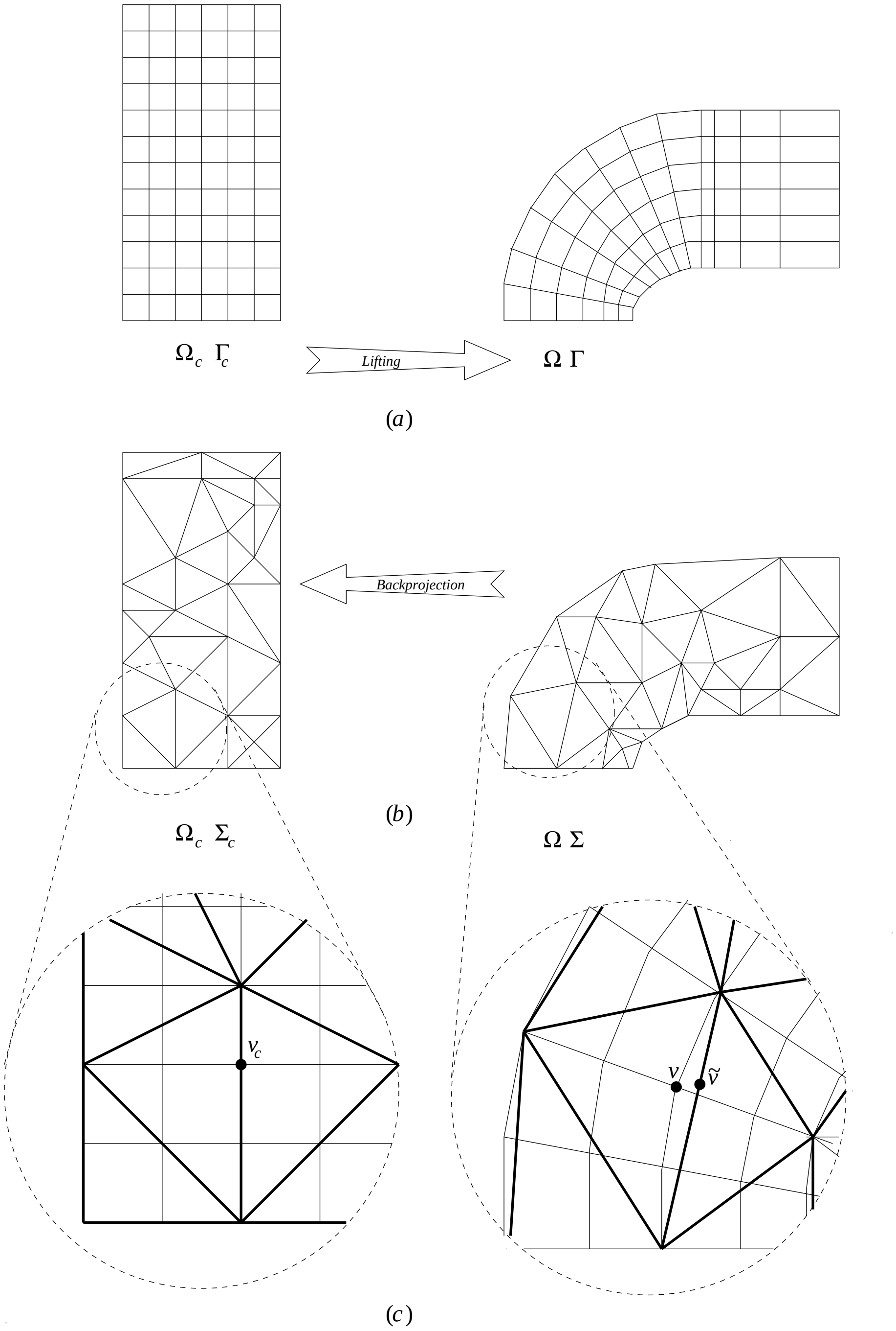

Fig. 2. Lifting and warping for curvilinear datasets (example in 2D): (a) lifting maps a regular mesh Γ_c into a curvilinear mesh Γ; (b) the triangular mesh Σ approximating Γ is back-projected in computational space into mesh Σ_c; (c) warping at a point v is equal to the distance from v to the warped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

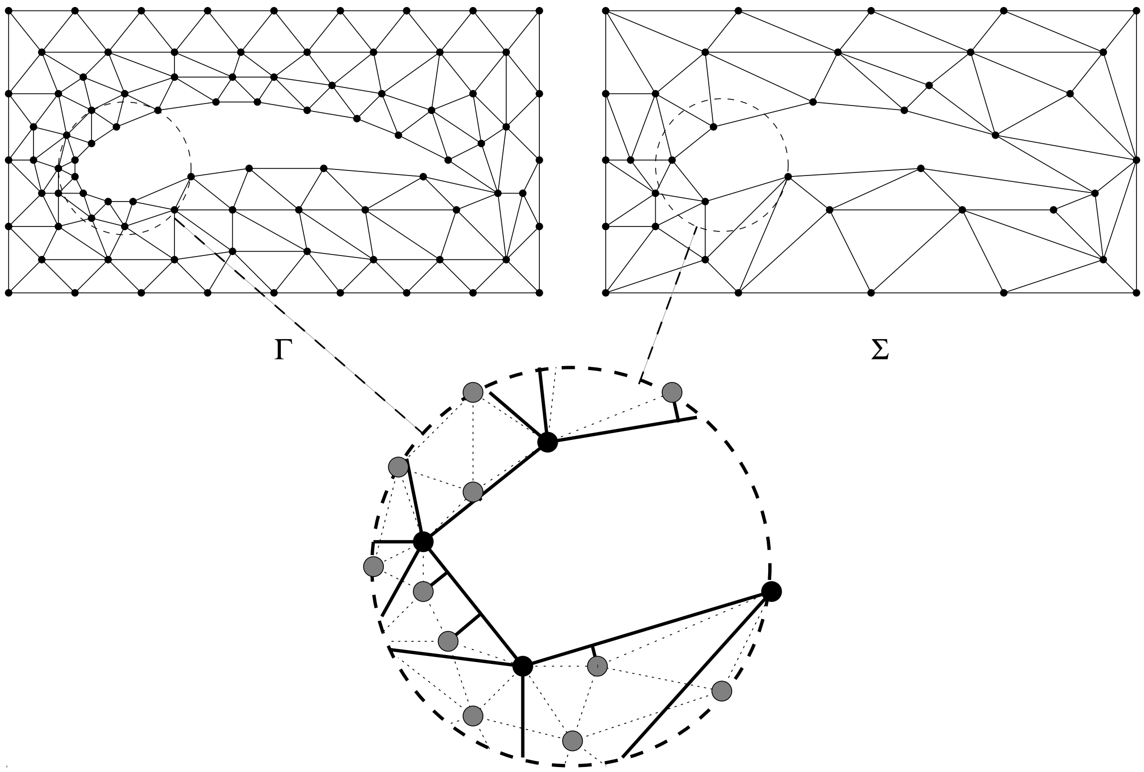

Fig. 3. For *non-convex irregular* datasets, we estimate the actual difference between the boundaries by computing at each boundary vertex of Γ its minimum distance from the boundary of Σ.

In this case, v is said *related to* the tetrahedron σ having v_p on its boundary (see Figure 3).
The error of a tetrahedron σ is the maximum among the error of all 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; for the non-convex irregular case, 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, which 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, let us assume that for applicative needs accuracy is relevant in the proximity of a selected point p. We can select weights that decrease 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.

III. Building an approximated model

Given a reference model (V, Γ), and a threshold pair $\mu = (\delta, \varepsilon)$, we face 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 suggests that the problem of minimising the size of the mesh for a given accuracy is intractable (NP-hard); also, approximated algorithms that warrant a bound on the size of the solution with respect to the optimal one are hard to find, and hardly applicable in practice [2], [1]. Hence, heuristics can be adopted, which try to obtain a mesh of reduced size by following data simplification strategies. There are two basic classes of strategies for simplifying a mesh:

- *Refinement heuristics* 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. Selection strategies can be adopted to insert at each step a vertex that is likely to improve the approximation better.
- *Decimation heuristics* 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.

Although in 2D several heuristics have been proposed, experiences in this case show a substantial equivalence of most of them in the quality of results. Since the three-dimensional case is almost unexplored, extending 2D techniques that seem most suitable to 3D is a reasonable approach.

In the following subsections, we present two simplification methods: the first method is based on refinement and Delaunay tetrahedrization, and 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.

A. A method based on refinement

A refinement method that we proposed in [5] for convex datasets is extended here to deal also with non-convex curvilinear datasets. The basic idea comes from an early technique developed in the two-dimensional case, and widely used for approximating natural terrains [14]. An on-line algorithm for Delaunay tetrahedrization 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 tetrahedrization 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 split 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 first. If V is a convex dataset, then Σ is a tetrahedrization of the convex hull of V. If V is a non-convex curvilinear dataset, then a tetrahedrization 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 physical domain. Given a threshold μ for the accuracy, the following refinement procedure is applied:

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

This refinement procedure always converges since the number of points in V is finite, and 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. This can be done efficiently by using a bucketing structure similar to that proposed in [22] for dynamic triangulation in 2D, which maintains for each tetrahedron a list of data points of V contained inside it;
2. select a new vertex v_{max} from the points of V by SELECT_POINT: for the convex case, the point of V that maximises $E()$ is selected; for the curvilinear case, the point of V that maximises either $W()$ or $E()$ is selected, depending on whether $W(\Sigma)/E(\Sigma)$ is larger or smaller than δ/ε, respectively. This can be done efficiently by the joint use of the bucketing structure, and of a priority queue, maintaining tetrahedra according to their error/warping;
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 [20]: in the curvilinear case, update is always made on the tetrahedral mesh in computational domain, and Σ is obtained through lifting.

Further details on the implementation of the refinement procedure for convex datasets can be found in [5]. Such a procedure can be adapted to the case of curvilinear datasets on the basis of the previous discussion.

A further remark is necessary, though, for 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 tetrahedrization 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 4 for a two-dimensional example. Consistency can be tested by verifying

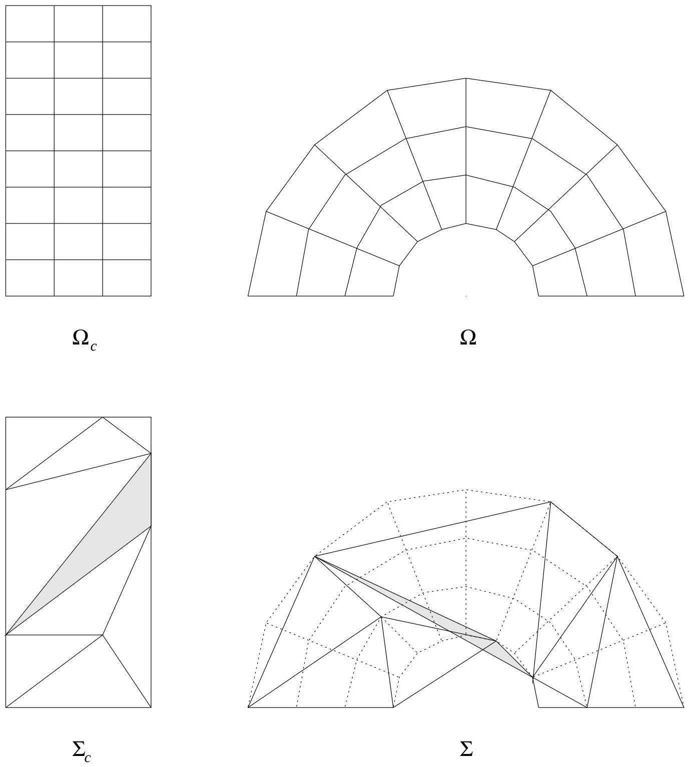

Fig. 4. Inconsistency in curvilinear mesh (2D example): mesh Σ_c is geometrically consistent, while its lifted image Σ is not.

whether each tetrahedron maintains its orientation both in computational and in physical domain.
We assign infinite warping to each tetrahedron that has an inconsistent lifting. In this way, inconsistent tetrahedra are refined first. We are warranted that the mesh in physical space will converge to a consistent one in a finite number of steps, although, in the worst case, it might be necessary to insert all data points. Indeed, let us consider the Delaunay mesh containing all data points in computational space: such a mesh is obtained by splitting each hexahedron of the original mesh into five tetrahedra. We know from the consistency of the original mesh that the lifting of each hexahedron in physical space is a convex polyhedron, and that no two such polyhedra overlap in physical space. Convexity warrants that when lifting the five tetrahedra covering a hexahedron we will obtain a consistent sub-mesh covering the lifted hexahedron exactly. Non-overlapping of hexahedra warrants that sub-meshes corresponding to different hexahedra will not overlap.
Experimental results show that in practice the mesh rapidly converges to a consistent one.

The time complexity of the refinement procedure is not crucial to our application, as long as it remains into reasonable bounds, because the algorithm is applied off-line to the volume dataset in order to build a multiresolution model (see Section IV). However, time analysis in case all n points of V must be inserted into Σ shows a bound of $O(n^3)$ in the worst case [5], while experiments show a subquadratic behaviour in practice. On the other hand, the space occupancy of this algorithm is quite high, because of the need

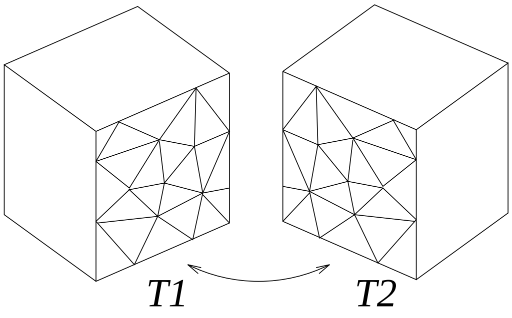

Fig. 5. Two adjacent blocks Σ_1 and Σ_2, and the coincident triangulations T_1 and T_2 of their common face.

of maintaining both a bucketing structure and a priority queue (see empirical evaluations in Section VI, Tables I and II).

A.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 space complexity into more manageable bounds, by splitting the dataset into blocks, and running the algorithm separately on each block. Assume, for instance, that a regular 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 indeed a tetrahedrization of the whole domain. 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). 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 vertex inserted does not lie on r, update will change neither T_1 nor the error and warping of data points lying on r; on the contrary, if the vertex inserted lies on r, it must be in particular the point maximising 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. Note that, however, the result will not be the same that we would obtain by running the refinement algorithm on the whole dataset, since the resulting tetrahedrization might not be globally Delaunay: the Delaunay property is verified only locally to each block.

B. A method based on decimation

The refinement method described above is hardly adaptable 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. Moreover, the Delaunay triangulation is not applicable to non-convex polyhedra, since it is undefined in the constrained case.

Experiences in the approximation of non-convex objects through 2D triangular meshes suggest that a decimation technique might be more adequate to the case of non-convex irregular datasets (see, e.g., [37], [19], [4]). In the following, we describe an algorithm that extends such heuristics to volume data: starting from the reference mesh Γ, vertices are iteratively discarded until possible. Given a threshold μ for the accuracy, the following refinement procedure is applied:

procedure DECIMATION(V, Γ, μ);
  $\Sigma \leftarrow \Gamma$;
  while Σ satisfies μ **do**
    $v_{min} \leftarrow$ SELECT_MIN_VERTEX(V, Σ, μ);
    $\Sigma \leftarrow$ REMOVE_VERTEX(Σ, v_{min})
  end while ;
  return (Σ)
end ;

The test of accuracy is simpler in this case than in the refinement procedure. Indeed, at each iteration, accuracy may worsen only because of local changes. Therefore, it is sufficient to maintain a variable storing the current accuracy, which is updated after each iteration by testing whether the accuracy in the changed portion of the mesh has become worse than the current one.

On the contrary, procedures SELECT_MIN_VERTEX and REMOVE_VERTEX are somehow more delicate than their respective counterparts SELECT_MAX_POINT and ADD_VERTEX.

Selecting a vertex to be removed involves an estimation of how much error and warping of the mesh may increase because of 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, and it may involve relocating many points lying inside such tetrahedra. We rather use heuristics to estimate apriori how much a vertex removal affects error and warping. Such an estimation is computed at all vertices before decimation starts, and it is updated at a vertex each time some 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, where 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

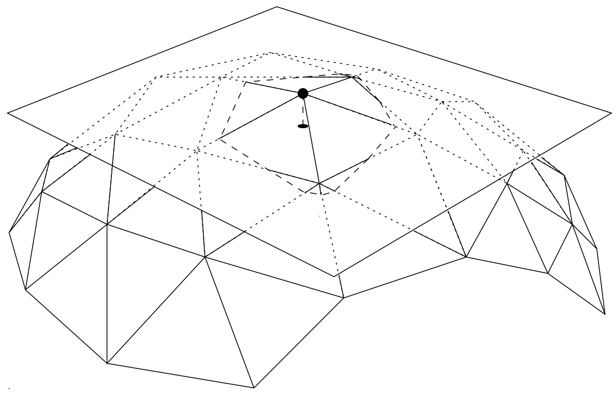

Fig. 6. An apriori estimate of warping increase caused by removing a boundary vertex v is obtained by measuring the distance of v from an average plane fitting its adjacent vertices on the boundary of Σ.

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.

Warping changes only if a vertex lying on the boundary of Σ is removed. Therefore, for each such 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 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 6): 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 that supports efficient selection. In this framework, the selection criterion adopted in procedure SELECT_MIN_VERTEX is symmetrical to the one used in the refinement algorithm: the vertex of Σ is selected which is expected to produce the smallest increase in either warping or error, depending on whether $W(\Sigma)/E(\Sigma)$ is larger or smaller than δ/ε.

Once a vertex v has been selected, we need to tetrahedrize the polyhedron resulting from the elimination of all the tetrahedra incident on v. Therefore, removing it from the mesh is not necessarily possible: this difficulty is related to the fact that it may not be possible to tetrahedrize 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. In particular, 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 7a for an example in 2D). Geometrical consistency of the mesh may be violated if some tetrahedron "flips over", i.e., it changes its orientation, because of edge collapsing (see Figure 7b for an example in 2D). Consistency can be tested simply by checking the orientation of each

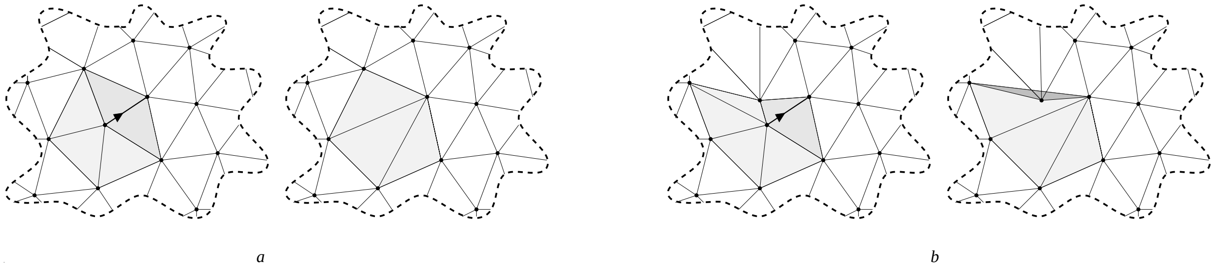

Fig. 7. Edge collapse in 2D: (a) a valid collapse; (b) an inconsistent collapse.

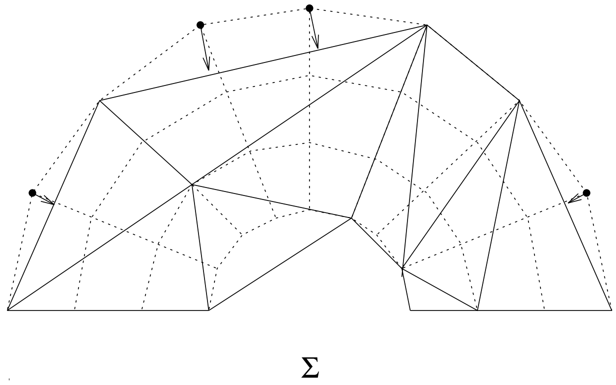

Fig. 8. Points that fall outside the mesh are assigned to tetrahedra by projecting them on the boundary faces.

tetrahedron incident at v before and after collapse. If collapse results impossible, then no mesh update occurs, while 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 relation 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, in case v was 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 8). 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 at each vertex that was adjacent to v, and the priority queue is updated accordingly.

IV. A multiresolution model

Each one of the algorithms described in the previous section can be regarded as producing a "historical" sequence of tetrahedra, namely all tetrahedra that appear in the current mesh Σ during its construction. Based on such an observation, we extend here to the three-dimensional case a simple idea to manage multiresolution, which we proposed in [9] in the two-dimensional case, for the multiresolution representation of terrains.

Each tetrahedron of the sequence is marked with two accuracies $\mu_b = (\delta_b, \varepsilon_b)$ and $\mu_d = (\delta_d, \varepsilon_d)$, called its *birth* and *death*, and corresponding to the worst and best accuracy of a mesh containing it, respectively. Therefore, we have $\delta_d \leq \delta_b$ and $\varepsilon_d \leq \varepsilon_b$ (in short, $\mu_d \leq \mu_b$).

Referring to a historical sequence generated by the refinement algorithm, we have that birth and death are the accuracy of the current mesh when the tetrahedron was inserted into it, and when it was discarded form it, respectively. The two values are swapped in case the historical sequence is built by decimation.

A. Querying the model

Given a query accuracy $\mu = (\delta, \varepsilon)$, we have that a mesh at accuracy μ will be formed by all tetrahedra that are μ-*alive*, i.e., such that $\mu_d \leq \mu \leq \mu_b$. Based on this fact, we use birth and death 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 that belong to a given query region.

Since a multiresolution model contains a huge number of tetrahedra, we have adopted a minimalist data structure, which is suitable to maintain the multiresolution model on a sequential file.

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. Therefore, space occupancy only depends on the number of sites, and on the number of tetrahedra in the historical sequence. Sites and tetrahedra are stored in two different files. Both sites and tetrahedra are sorted in the order they appear in the mesh during construction through refinement (in the inverse order, if the model is built through decimation). Therefore, tetrahedra result in a non-increasing order of birth.

In this case, the sequence of tetrahedra belonging to a model at a given resolution μ is obtained by sequentially scanning the file, while selecting tetrahedra according to their birth and death: 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. Tetrahedra covering a given range of accuracies are obtained similarly. Vertices of such tetrahedra are obtained by scanning the sequence of sites up to the highest element indexed by a tetrahedron in the set extracted.

Note that performing a combined windowing operation would require a subsequent filter to scan all tetrahedra after their extraction.

Search efficiency might be improved by adopting data structures for range queries, such as the *interval tree* [31], or the *sequence of lists of simplices* [3]. However, such data structures might introduce a relevant memory overhead. In particular, adopting the sequence of lists of simplices would make sense only if the list of all accuracies spanned by the multiresolution model (which might be as large as the number of tetrahedra forming it) can be maintained in the main memory. The interval tree gives optimal time performance, but its application would be effective only if the whole model can be maintained in the main memory.

On a different perspective, spatial indexes [36] might be adopted to improve the performance of windowing operations, but also such structures involve some memory overhead.

B. 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, while providing an implicit encoding that allows the client to make the structure explicit efficiently.

If the model is built through procedure REFINEMENT, by exploiting the properties of Delaunay tetrahedrizations, 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 tetrahedrization [20] while vertices are received. Note that this is much a cheaper task 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 through procedure DECIMATION, a similar technique may be adopted, following Hoppe [19]. 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 on 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 to perform a *vertex-split* operation that inverts the *edge-collapse* performed by the decimation algorithm [19].

[**** ENRICO: mi sembra che cosi' non basti - ci vorrebbero anche le facce che vengono duplicate per formare i nuovi tetraedri. E' un po' noioso da spiegare: che si fa? Si potrebbero eliminare i dettagli facendo riferimento semplicemente al metodo di Hoppe e alla possibilita' di estenderlo in 3D ******]

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, while, on the other hand, the amount of information transmitted is larger.

The size of data transmitted can be reduced further by using geometric compression [12].

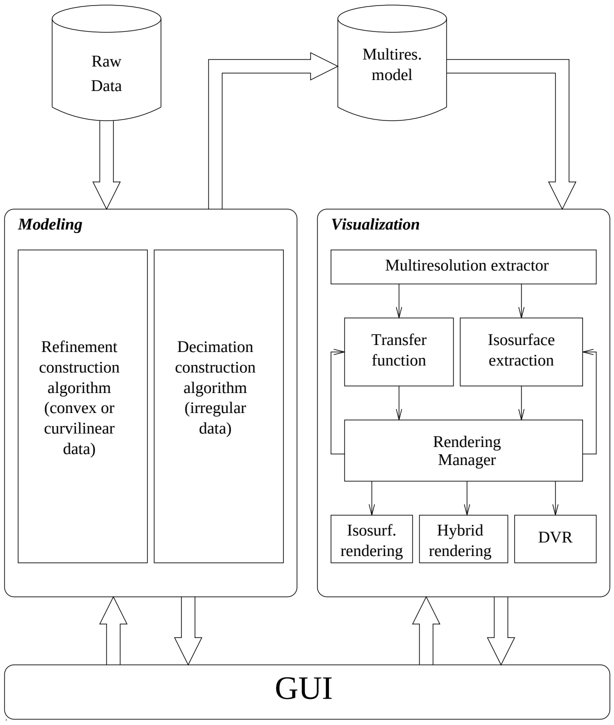

Fig. 9. The architecture of the TAn system.

V. The TAn system

On the basis of the multiresolution model and algorithms described in the previous sections, we have designed a volume visualization system, called **TAn** (**T**etrahedra **An**alyzer), which is able to manage multiresolution based on approximated tetrahedral representations of volume data.

A. System architecture

The architecture of TAn is depicted in Figure 9. The system is essentially composed of two modules, the *modeling module* and the *visualization module*, which communicate with each other through the multiresolution data structure, while each of them can communicate with the user through a Graphical User Interface.

- The modeling module contains the algorithms for building a multiresolution model, starting from a volume dataset: either the *refinement* or the *decimation* algorithm is used to build the model, depending on the type of the dataset in input. The user selects an input dataset, and construction parameters through the GUI; then, the system reads the corresponding data file, and it runs a construction algorithm. The resulting multiresolution models is stored by using the data structure described in Section IV.

  The modeling module is essentially intended to run off-line, during a phase in which the multiresolution model is prepared, and stored on the file system for subsequent visualization.

- Once a multiresolution model has been built, the visualization module can access it through a submodule called the *multiresolution extractor*, which contains query processing routines that access the multiresolution data structure, as explained in Section IV-A.
- Tetrahedra extracted from the multiresolution model are piped to two independent submodules: one that manages a *transfer function*, and one that performs *isosurface extraction*.

  A transfer function is applied to the range covered by the extracted data, in order to provide color and opacity for each vertex used in direct volume rendering. The user can load, edit, and store transfer functions through the GUI.

  Isosurfaces are obtained through a method called the *Marching Tetrahedra (MT)*, which is a straightforward adaptation of the Marching Cubes (MC) [25] to tetrahedral meshes: each tetrahedron is classified in terms of the value of its four vertices, and triangular patches are obtained by using linear interpolation along each edge intersected by the isosurface. Isosurface patches are extracted from all tetrahedra loaded in memory, and for either one or more isovalues provided by the user through the GUI. The user can also define color and opacity for each isovalue independently.

  This stage essentially provides geometries, namely a set of tetrahedra prepared for direct volume rendering, and a set of isosurface patches prepared for surface rendering, respectively.
- Geometries are piped to the Rendering Manager submodule that controls visualization on the basis of the data currently loaded in memory and of parameters provided by the user. This submodule essentially is aimed to filter the geometries (triangles and/or tetrahedra) that should be visualized, at each time, and in each location of space. In this way, we are able to implement mechanisms such as *progressive rendering* – where a low-level mesh can be used during interactive phases, while a high level mesh is used when the user can wait longer for visualization; and *multiresolution rendering* – where different LoDs are used in different portions of space, e.g. to either enhance quality or magnify a selected portion of the dataset [8]. Filtering is again performed on the basis of the birth, death, and location of each tetrahedron or triangle.

  In order to improve performance, the user is allowed to ask for further interactive extraction of isosurfaces only from tetrahedra of interest. In this case, the Rendering Manager module pipes back to the isosurface extractor only a pointer to the current tetrahedra list, and collects more isosurface patches.
- The geometries selected for visualization can be piped to one among three different modules, depending on the rendering modality selected by the user.

  If only *isosurface rendering* is enabled, then a proper module that visualizes them through standard surface graphics is invoked, which is passed the set of isosurface patches of interest. Note that, if translucent surfaces are used, it is necessary to sort isosurface patches in depth order prior to visualization.

  If only *direct volume rendering* is enabled, then the selected set of tetrahedra is passed to a Projected Tetrahedra (PT) algorithm [38], whose main phases are a depth sort of the tetrahedral mesh and then, for each cell in depth order, a split-and-compositing action that produces translucent triangles, visualized through standard surface graphics.

If both isosurfaces and direct volume rendering are used, then both tetrahedra and isosurface triangles are passed to a module that manages *hybrid rendering*. In this case, blending conflicts among tetrahedra and isosurface patches must be resolved. In order to do this, each tetrahedron which contains a surface patch splits into two parts, each of which is further tetrahedrized. The resulting set of tetrahedra and isosurface patches are then sorted in depth order, the PT algorithm is applied to tetrahedra, and the results are visualized in depth order through standard surface graphics.

It is easy to change this architecture into a network architecture based on a client/server model, by using the data transmission method described in Section IV-B. In this case, the server would contain the modeling module, plus a query processing module that provides, upon request from the client, a compressed data structure of the extracted mesh or set of meshes.

The client would incorporate the visualization module, where the multiresolution extractor would be simply a module that schedules requests to the server, collects answers, and decompress the data structure.

B. *Prototype implementation*

The architecture described in the previous section has been partially implemented. A first version of the TAn system has been released in the public domain in the first quarter of 1996, and it is available (SGI executables only) at our Internet site `http://miles.cnuce.cnr.it/cg/swOnTheWeb.html`. The system works on SGI workstations and uses OpenGL to manage graphics data output. Its GUI has been implemented by XForms [46], a portable and easy-to-use user interface toolkit available in the public domain (see at `http://bragg.phys.uwm.edu/xforms`).

We implemented the *refinement construction algorithm* both for convex and non-convex curvilinear data, but only the convex version is included in the first release of the system (experiments on curvilinear data shown in the next section were obtained with a stand-alone version of the algorithm). The *decimation construction algorithm* for irregular datasets is currently under implementation.

The *multiresolution extractor* provides a function for extracting a mesh at any LoD provided by the user. Two meshes can be loaded into main memory, one at a high LoD, and the other at a low LoD, and used for interactive rendering.

Figure 12 shows snapshot of the two GUI windows that allow the user to build a multiresolution model, and to extract LoD representations from it. The system provides statistics on the size of meshes at different LoDs: the user can therefore make his choice for the approximated models by taking into account the performances of the workstation used, the frame rate required, and the image quality degradation which may be accepted.

The following visualization features were implemented:

- loading and interactive editing of the transfer function;
- multiple isosurface extraction through the MT method;
- isosurface rendering with user defined color and opacity;
- direct volume rendering through the PT method;
- approximated hybrid rendering;
- interactive modification of view parameters;
- a progressive rendering modality.

A snapshot of the graphic output window, and of GUI windows related to rendering is presented in Figure 13. The window in the upper left corner is the main menu of the system; the window in the upper right corner allows the user to extract an isosurface and to assign it a given color and opacity; the other two windows on the right side are related to visualization and editing of the transfer function; the window in the lower left corner allows the user to interactively adjust view parameters; the window in the middle is used to select the rendering modality (isosurfaces, or DVR, or both).

The approximated hybrid rendering is implemented as follows. For each tetrahedron, the system explicitly stores its related isosurface facets. At rendering time all cells are depth-sorted and, for each cell, both the volume contribution (obtained with the PT algorithm), and the isosurface facets possibly contained into it are projected. Since tetrahedra are not split prior to depth-sort, the result is only approximated because of different parcels of a single tetrahedron cannot be sorted correctly with respect to its related isosurface patches. The degradation in image quality may be relevant when low resolution approximations are used, but it is highly reduced with the increase of resolution (i.e., the smaller the single cell, the smaller the visual error introduced by the approximated hybrid rendering). An example of approximated hybrid rendering is shown in Figure 15. The exact method for hybrid rendering, described in the previous section, is currently under implementation.

The progressive rendering modality can be selected by the user to improve interactivity. The mesh at low LoD is visualized during the highly interactive phases (e.g., while the user interactively modifies the current view), while the mesh at high LoD is automatically visualized when interaction does not occur for a given time period (i.e., during non–interactive phases). While in the current implementation the low LoD is set by the user, in a more sophisticate version it could be selected automatically by the system, depending on the graphics performance of the current platform, in order to ensure real time frame rate.

VI. Experimental Results

The performances of the system were evaluated on four datasets, representative of the two classes of regular and non-convex curvilinear datasets. Datasets were chosen as they are commonly used in the volume rendering field, 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;
- **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].

Multiresolution models of such datasets were built through the refinement construction algorithm, and the various visualization features of TAn were experimented on such models.

A. Multiresolution modeling features evaluation

Tables I and II report results on the construction of a multiresolution model from curvilinear and regular datasets, respectively. Each table reports: the complexity of the multiresolution model (total number of sites and cells, maximal RAM space occupancy during construction); computation times required to build the model; 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 bounding box containing the dataset, while error is a percentage of the range spanned by data values. Times are CPU seconds of an SGI Indigo workstation (MIPS R4000 100MHz).

The graph of Figure 10 shows the 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 with the increase of error. These results give a quantitative estimate of the advantage of founding approximate volume visualization on data simplification techniques.

Figure 11 shows the spatial distribution of sites of the BluntFin dataset, compared with the spatial distribution of vertices of an approximated model at accuracy $(2.\%, 2.\%)$

As you may notice, the experiment reported in Table II for the BuckyBall dataset were run on a subsampling, because of limitations in the available RAM. A multiresolution model on the whole dataset, and on two subsampled datasets, were also obtained by using the *block-decomposition refinement* described in Section III-A.1. Results are presented in Table III. By adopting this method we can overcome the intrinsic limitations of RAM 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

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

[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. sites	% of sites
BluntFin (40x32x32)		40,960	
Multires Model: tot. tetra.= 590,831 RAM = 35,300 Kb construction time = 1,704 sec.			
Levels of Detail:			
$\delta \leq 4.0\%$, $\varepsilon \leq 4.0\%$	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	
Multires Model: tot. tetra.= 1,620,935 RAM = 95,241 Kb construction time = 7,794 sec.			
Levels of Detail:			
$\delta \leq 4.0\%$, $\varepsilon \leq 4.0\%$	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 I

Measures on multiresolution models built from curvilinear datasets (the Post triangulation times are higher than expected due to page swapping: the RAM size of the workstation used was only 64MB).

(lower part of Table II) and refined as 64 independent blocks (upper part of Table III). Note that, with the block decomposition refinement, total computation time reduces from 1,318 sec. to 532 sec., while we have only a small increase in the number of vertices 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 accuracies, 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 reduces to 10:1 at accuracy 1.0%.

Regular Datasets	no. tetra.	no. sites	% of sites
SOD (32x32x32)		32,768	
Multires Model: tot. tetra.= 177,588 RAM = 45,134 Kb construction time = 1,491 sec.			
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	
Multires Model: tot. tetra.= 472,130 RAM = 25,868 Kb construction time = 1,318 sec.			
Levels of Detail:			
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	43 %
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 II

MEASURES ON MULTIRESOLUTION MODELS BUILT ON TWO REGULAR DATASETS.

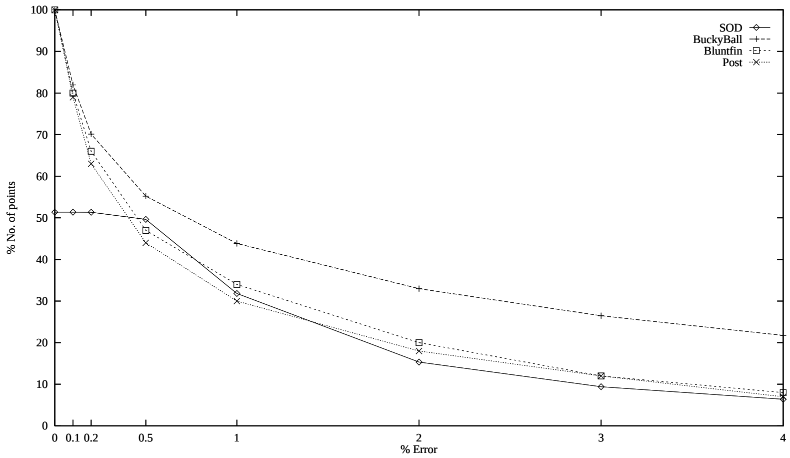

Fig. 10. Number of points in the simplicial model expressed as a function of the approximation error.

	no. tetra.	no. sites	% of sites
BuckyBall (32x32x32)		32,768	
Multires Model:			
tot. tetra.= 467,261 RAM = 530 Kb construction time = 532 sec.			
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	
Multires Model:			
tot. tetra.= 3,927,793 RAM = 4,933 Kb construction time = 5,412 sec.			
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	
Multires Model:			
tot. tetra.= 32,472,130 RAM = 4,935 Kb construction time = 44,086 sec.			
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 III

Tetrahedrization 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.

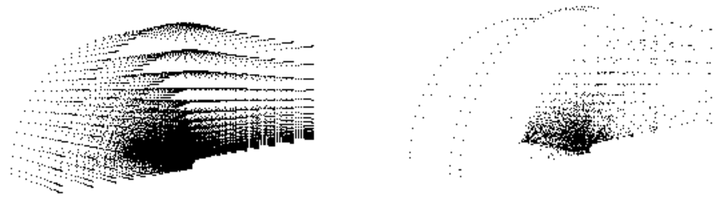

Fig. 11. Distribution of vertices of the BluntFin dataset: original dataset (40,960 sites) on the left, approximated mesh with $\delta \leq 2\%$ and $\varepsilon \leq 2\%$ (8,263 sites) on the right.

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 IV

Isosurface rendering (with threshold value 1.244), and direct volume rendering of the same dataset at different accuracies. Times are in seconds on an SGI Indigo XS24 R4000 ws.

B. Rendering features evaluation

Figure 14 presents visual results related to isosurface and direct volume rendering of three representations of the BluntFin dataset. The top images refer to the mesh at full resolution, the middle images refer to an approximated mesh at accuracy $(1.0\%, 1.0\%)$, while the bottom images refer to an approximated mesh at accuracy $(4.0\%, 4.0\%)$.

Numerical results on the size of the meshes, of the extracted isosurfaces, as well as times for DVR, are summarized in Table IV. The images provide evidence that the image degradation is almost un-perceivable when passing from full accuracy to $(1.0\%, 1.0\%)$ accuracy, while it is still small at $(4.0\%, 4.0\%)$, while the output sizes (and times) are highly reduced.

Visualization results obtained with TAn, which are essentially based on the concept of *data simplification*, can be also compared with results obtained with approximation methods that are based on *graphics output simplification.*

In 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 contrary, a *geometry-based simplification* of an isosurface extracted from the mesh at full resolution would be driven by isosurface curvature ([37], [19]). An obvious computational advantage of the approach based on data simplification is that the major effort is taken in a preprocessing stage (i.e., when the either 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 curvature variations which are small in size, but reflect significant variations of the field (e.g., a sinusoidal function, having amplitude lower than the simplification threshold), and, 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 previous paper [7], we also compared the performance of DRV through the standard PT algorithm applied to a simplified mesh, to the performance of approximated versions of the PT algorithm [43] applied to a mesh at full resolution. Experiments showed evidence that images with visual degradations similar to those obtained using the approximated PT are produced using highly simplified datasets, thus obtaining much shorter processing times (about five times shorter).

The large difference in speedups is because standard approximated PT techniques only act on the pure rendering phase, thus achieving a reduction in overall time 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 and cell classification and splitting as well).

VII. Conclusions

TAn is currently the only volume visualization system distributed in the public domain that offers multiresolution features, at least to our knowledge. Our experience with it provides evidence that the visualization of volume data can be managed effectively and efficiently by using multiresolution features based on the concept of data simplification.

The experimental results show that managing multiresolution involves a limited increase in the space complexity: the ratio between the size of the multiresolution model is in the average case about 2.5 times the size of the mesh at maximal accuracy.

Moreover, the proposed representation supports the design of fast approximated, progressive or multiresolution visualization algorithms, which are aimed at providing significant speedups in rendering, and at increasing the acceptance of visualization as a useful working tool.

Critical points for the usability of our approach are in the high requirements in memory and processing time needed to build the multiresolution model. With the current implementation, the tetrahedrization of high resolution datasets (e.g., with more than $100K$ sites) may require a memory size beyond that available on current low–level workstations. This problem may be solved by building the multiresolution model on high–level workstations/supercomputers, or by redesigning this process in order to reduce its memory and processing requirements. For instance, our strategy based on block decomposition has given good results for regular and curvilinear datasets.

A possible extension of the proposed multiresolution model is to structure data to allow the extraction of approximated representations whose accuracy is variable through data domain. This is especially useful for *multiresolution visualization*, when different accuracy levels must be used inside a single image. In this context, it may be extremely useful to supply the user with tools to set a "focus region", and render data according to that selection [30].
Unfortunately, extracting meshes at variable resolution from our current model may originate consistency problems (i.e., possible discontinuities of the field, with consequent "cracks" in the isosurfaces, and aliasing in DVR). In a previous paper [8], we implemented multiresolution rendering by using two different meshes, at high and low resolution, respectively: the high resolution mesh is rendered inside a region of interest, while the other is used outside such a region. Topological inconsistencies that occur between the two meshes at the boundary of the region of interest were overcome by visualizing cells of both meshes that cross such a boundary, and using blending on such cells.
A more rigourous solution of such a problem should be given at the level of the *multiresolution extractor* module, by providing a mechanism for extracting a mesh whose accuracy varies "smoothly" and consistently through domain. In recent works [9], [32] we proposed alternative multiresolution data structures that provide efficient solutions to this problem, and that produce effective results in the two-dimensional case, e.g. for visualizing terrain models in the context of flight simulators. However, such structures may require a relevant overhead in terms of storage, which make them not easily extensible to the three-dimensional case.

We are currently working on the second release of the TAn system. TAn v.2 is based on OpenInventor, and its GUI is under development using the SGI RapidApp tool. We plan to distribute it in Q2 1998. The system has been redesigned quite from scratch, in order to improve performances, usability, and visual quality, while maintaining the same architecture described in Figure 9. The *Modeling* tool and the *Visualization* tool have been clearly separated, and related through the multiresolution data structure.
The *Modeling* tool is designed to manage all kinds of datasets. The simplification algorithm for irregular datasets is currently under implementation, and it will be completed and tested in short time. Experiences in 2D [19], [4], and with similar decimation techniques in 3D [34] suggest that the method should result at least as effective as that based on refinement. However, its performances (both in terms of time, and data simplification rate) will be compared with those of the refinement algorithm on convex and curvilinear datasets. Upon the results of this comparative evaluation, we will decide whether both algorithms, or only the decimation algorithm will be incorporated into TAn v.2 *Modeling* subcomponent.
The *Rendering* subcomponent has been substantially improved, in order to provide: faster DVR (TAN v.2 rendering speed is approximately 30 Ktetra/sec under OpenInventor on an SGI Indigo2 XZ R4400 200MHz ws, preliminary results); a new rendering approach which encompasses both exact hybrid rendering and exact management of transfer function discontinuities, based on cell slicing; a simplified GUI.
In conclusion, our goal is to found the rendering modules of our architecture on a new concept of *tetrahedral*

graphics, where tetrahedra are treated as atomic graphics primitives, just like triangles, and are efficiently processed by low-level functions provided by the graphics library, and possibly hardware-assisted. In this way, we would clearly separate the geometric aspects of volume visualization, which are treated by application programs/modules, from the purely graphical aspects, which should be standardized, and treated at library and hardware level.

Acknowledgements

We wish to thank Leila De Floriani, for her participation in the early stages of this project, and for many useful discussions; this work is part of a continuing collaboration between her group at the University of Genova, and the authors. Thanks are also due to Donatella Sarti, Pierluigi Sbaiz and Marco Servettini for their help in implementing the algorithms described in the paper.

References

[1] P.K. Agarwal and P. Desikan. An efficient algorithm for terrain simplification. In *8th ACM-SIMA Symposium on Discrete Algorithms*, 1997.

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.).

[9] P. Cignoni, E. Puppo, and R. Scopigno. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer*, 13(5):199–217, 1997.

[10] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Computer Graphics Proc., Annual Conf. Series (Siggraph '96), ACM Press*, pages 119–128, Aug. 6-8 1996.

[11] J. Danskin and P. Hanrahan. Fast Algorithms for Volume Ray Tracing. In *Proceedings of the 1992 Workshop on Volume Visualization (Boston, MA)*, pages 91–98, October 1992.

[12] M. Deering. Geometry compression. In *SIGGRAPH '95 Conference Proceedings*, pages 13–20. ACM Computer Graphics Conf. Proc. Series, 1995.

[13] H. Edelsbrunner. An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10(3):251–260, 1990.

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

[15] R. Grosso, C. Luerig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *IEEE Visualization '97*, page (in press), Phoenix, AZ, Oct. 19-24 1997.

[16] B. Guo. A multiscale model for structured-based volume rendering. *IEEE Trans. on Visualization and Computer Graphics*, 1(4):291–301, December 1995.

[17] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

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

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

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

[21] A. D. Kalvin and R.H. Taylor. Superfaces: Poligonal mesh simplification with bounded error. *IEEE C.G.&A.*, 16(3):64–77, 1996.

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

[23] D. Laur and P. Hanrahan. Hierachical splatting: a progressive refinement algorithm for volume rendering. *ACM Computer Graphics (Proceedings of SIGGRAPH '91)*, 25(4):285–288, 1991.

[24] J. Lee. Comparison of existing methods for building triangular irregular network models of terrain from grid digital elevation models. *International Journal of Geographic Information Systems*, 5(3):267–285, 1991.

[25] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *ACM Computer Graphics (Proceedings of SIGGRAPH '87)*, 21(4):163–170, 1987.

[26] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. on Patt. Anal. and Mach. Intel.*, 11(7):674–693, 1989.

[27] 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.

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

[29] R. Neubauer, M. Ohlberger, M. Rumpf, and R. Schwirer. Efficient visualization of large–scale data on hierarchical meshes. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97*. Springer Wien, 1997.

[30] A. Pang. Spray rendering. *IEEE Computer Graphics and Applications*, pages 57–63, September 1994.

[31] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

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

[33] E. Puppo and R. Scopigno. Simplification, LOD, and Multiresolution - Principles and Applications. Technical Report C97-12, CNUCE, C.N.R., Pisa (Italy), June 1997. (also in: EUROGRAPHICS'97 Tutorial Notes, Eurographics Association, Aire-la-Ville (CH)).

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

[35] J. Ruppert and R. Seidel. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. In *Proceedings 5nd ACM Symposium on Computational Geometry*, pages 380–392, 1989.

[36] H. Samet. *The design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, MA, 1990.

[37] W.J. Schroeder, J.A. Zarge, and W. Lorensen. Decimation of triangle mesh. *ACM Computer Graphics*, 26(2):65–70, July 1992.

[38] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Computer Graphics (Proc. of 1990 WS on Volume Visualization)*, 24(5):63–70, Nov 1990.

[39] R. Westermann. A multiresolution Framework for Volume Rendering. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 51–57. ACM Press, October 17-18 1994.

[40] J. Wilhelms. Pursuing interactive visualization of irregular grids. *The Visual Computer*, 9:450–458, 1993.

[41] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transaction on Graphics*, 11(3):201–227, July 1992.

[42] 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.

[43] P.L. Williams. Interactive splatting of nonrectilinear volumes. In A.E. Kaufman and G.M. Nielson, editors, *Visualization '92 Proceedings*, pages 37–45. IEEE Computer Society Press, 1992.

[44] P.L. Williams. Visibility ordering meshed polyhedra. *ACM T.O.G.*, 11(2):103–126, 1992.

[45] 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.

[46] T.C Zhao and M. Overmars. Forms Library – a graphical user interface toolkit for X. Technical Report 95-, Department of Computer Science, Utrecht University, Utrecht, NL, 1995.

[47] 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.

Authors' biographies

Paolo CIGNONI **is research scientist at the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include computational geometry and its interaction with computer graphics, scientific visualization, volume rendering. He received in 1992 an advanced degree (Laurea) in Computer Science from the University of Pisa, where he is currently a PhD student.**

Claudio MONTANI **is research director with the Istituto di Elaborazione della Informazione of the National Research Council in Pisa, Italy. His research interests include data structures and algorithms for volume visualization and rendering of regular or scattered datasets. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1977. He is member of IEEE.**

Enrico PUPPO **is research scientist at the Institute for Applied Mathematics of the National Research Council of Italy, and since 1991 he has a joint appointment at the Department of Computer and Information Sciences of the University of Genova. His current research interests are in topological modeling, geometric algorithms and data structures, with applications to Geographical Information Systems, virtual reality, scientific visualization, and image processing. He received an advanced degree (Laurea) in Mathematics from the University of Genova, Italy, in March 1986. He is a member of IEEE.**

Roberto SCOPIGNO **is research scientist at the Istituto CNUCE of the National Research Council in Pisa, Italy; since 1990 he has a joint appointment at the Department of Computer Engineering of the University of Pisa. His research interests include interactive graphics, scientific visualization, volume rendering, parallel processing. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1984. He is member of IEEE and Eurographics.**