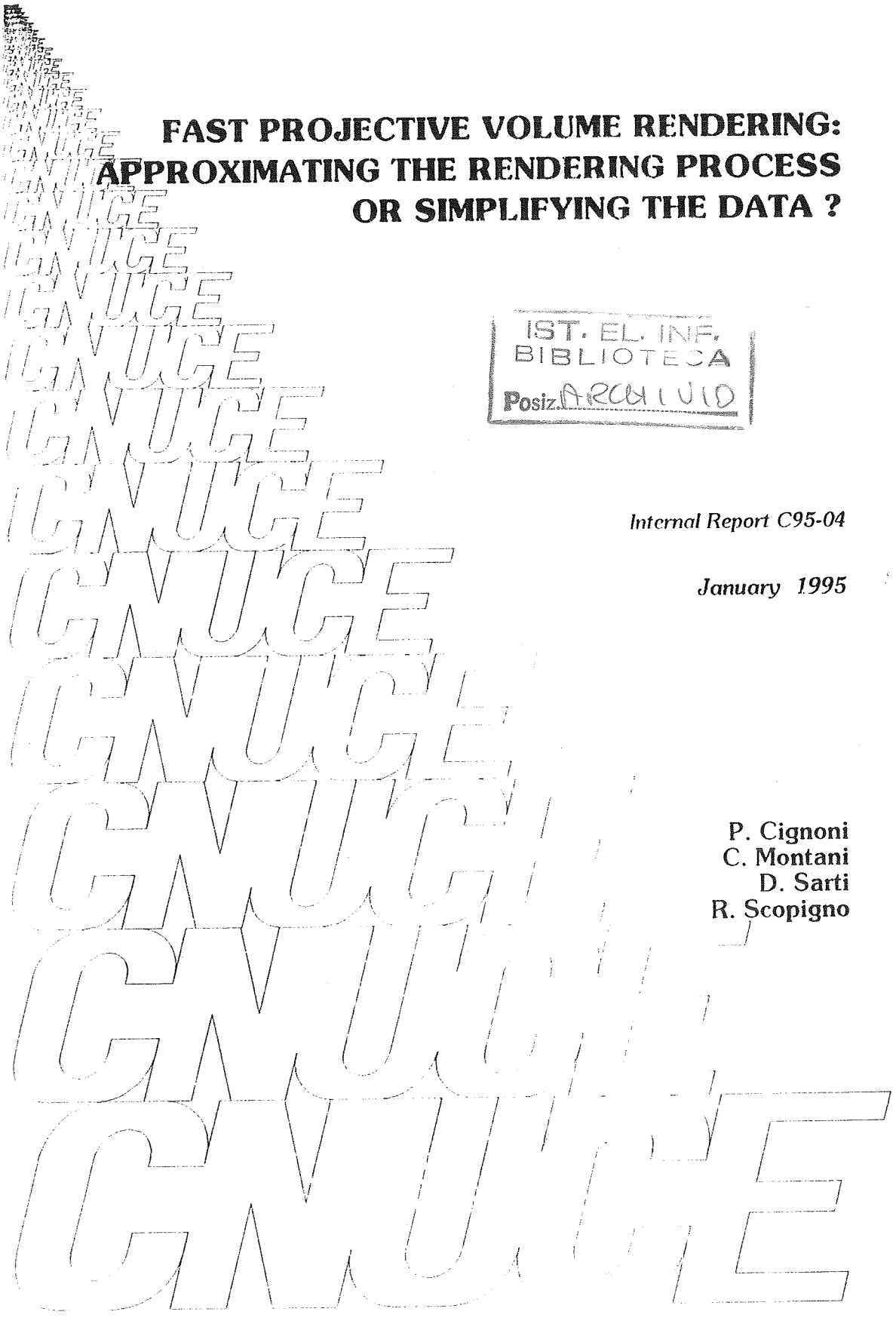
FAST PROJECTIVE VOLUME RENDERING: APPROXIMATING THE RENDERING PROCESS OR SIMPLIFYING THE DATA ?

Internal Report C95-04

January 1995

P. Cignoni
C. Montani
D. Sarti
R. Scopigno



Fast Projective Volume Rendering : approximating the rendering process *or* simplifying the data?

P. Cignoni[†], C. Montani[‡], D. Sarti[‡], R. Scopigno[⋆]

[†] Dip. Scienze dell'Informazione – Università di Pisa, C.so Italia, 40 - 56100 Pisa ITALY

Email: cignoni@cnuce.cnr.it

[‡] I.E.I. – Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56126 Pisa, ITALY

E-mail: montani@iei.pi.cnr.it

[⋆] CNUCE – Consiglio Nazionale delle Ricerche, Via S. Maria 36, 56126 Pisa, ITALY

E-mail: R.Scopigno@cnuce.cnr.it

Abstract

How to render very complex datasets, and yet maintain interactive response times, is a hot topic in volume rendering. In this paper we focus on projective visualization of datasets represented via tetrahedral tessellations. Direct projective visualization is performed by sorting tetrahedra with respect to view direction and then by projecting them onto the screen. Different sorting algorithms and "per tetrahedra" projection techniques are reviewed and evaluated. A new method for tetrahedra projection approximation is presented. In addition, we compare the results obtained by the optimization of the rendering process with those obtained by adopting a data simplification approach.

beyond current speeds, especially in the case of low or medium capacity workstations. In this paper we thus evaluate the possible optimizations of the Projected Tetrahedra algorithm [6], i.e. approximations of the rendering process and alternative sorting algorithms, together with the refinement of one of these approaches. In addition, we show that a data simplification approach can also be applied to produce significant speedups while maintaining similar approximations in the images produced. Therefore, we prove William's intuition [8] that true real-time interactive projection can be only obtained through data reduction.

The paper is organized as follows. Projective techniques for the direct visualization of volume datasets are briefly introduced in the next section. Section 3 describes the possible optimization of the Projective Tetrahedra algorithm. In Section 4 we show how a data simplification approach can give better results than the optimization of the "rendering process" presented in the previous section. Our conclusions are drawn in the last section.

2 Projective Direct Volume Rendering

Direct visualization of volume datasets represented by a tetrahedral decomposition was originally proposed by Max et al. [4], and Shirley and Tuchman [6]. Both these solutions apply a density cloud model [11] and solve the visualization task with a two–step approach: tetrahedra are first depth–sorted, and then each cell is projected onto the screen, and its color and opacity contributions are composed with the previous ones to form the resulting image. The two proposals differ on the method used to compute the contribution of each cell.

The algorithm by Max et al. implements a software scan conversion process in which the contribution of each cell to the generic pixel is computed by analytically integrating color and opacity along the line of sight.

Shirley and Tuchman's technique, called **Projected Tetrahedra (PT)**, approximates the contribution of each cell with a set of partially transparent triangles. This polygon–oriented approach is faster than the previous pixel–oriented one because conventional graphic hardware can be exploited. Shirley and Tuchman's proposal is addressed to the projective rendering of regular hexahedral grids. The first step in the visualization process is thus to decompose hexahedral cells into tetrahedra. An explicit depth sort of the cells is not required because it can be simply derived by the implicit ordering of the cells in the regular grid.

Solutions for the depth sort of tetrahedral grids have been proposed by Williams (*topological* sort on

curvilinear grids [8]), and by Max et al. (*numeric distance* sort on Delaunay triangulations [4]).

In this paper we limit ourselves to the projective visualization of datasets represented by Delaunay triangulations[1]. We assume that in the case of a hexahedral cells dataset, it would be decomposed into tetrahedral cells in a preprocessing step. The Projected Tetrahedra algorithm can be briefly described as follows.

PT algorithm:

1. apply a transfer function to each vertex in the dataset to map its scalar value into a color and a density value;
2. sort in depth the cells, back to front, according to the current view (parallel or perspective);
3. for each cell:
   (a) classify the cell according to its projected silhouette (Figure 2);
   (b) decompose the projected silhouette of the cell into triangles (Figure 2);
   (c) find color and opacity values for the *non zero thickness* (NZT) vertex using integration in depth on the cell in world coordinates;
   (d) for each split triangle: scan convert, interpolate color and opacity, compose with the current image (using graphic hardware features).

3 Optimization of Projected Tetrahedra

The **PT** algorithm is composed of two independent and sequential phases. We review and evaluate the two possible solutions to the *depth sort* of Delaunay tessellations, and then the optimization of the "per tetrahedra" *projection and composition* phase.

3.1 Depth sort optimization

The so called depth (or visibility) sort must guarantee that the cells are ordered in a such way that, given a viewpoint, if cell a obstructs cell b then b precedes a in the ordering.

[1]The Delaunay triangulations of a set of points P in E^3 consists of exactly those tetrahedra with vertices in P which satisfy the property that their circumsphere contains no other points of P into its interior [5].

The two depth sort algorithms proposed in literature for triangulated datasets are the **Meshed Polyhedra Visibility Ordering (MPVO)** algorithm by Williams [9], and the sorting algorithm proposed by Max et al. [4], called in the sequel **numeric distance sorting.**

MPVO is much more general than the second solution because it sorts the cells of any acyclic convex set (another solution for the visibility sort of unstructured meshes has been recently proposed by Stein et al. [7]; it is more general than MPVO, because allowes multigrid data sorting, but is also more complex in time). The MPVO algorithm requires linear time and uses linear storage. It is a topological solution, based on a preprocessing step (view independent) and two view–dependent steps:

1. an adjacency graph is built, with a node for each cell and a link for each couple of adjacent cells (preprocessing step);
2. a direction is assigned to each link, dependent on the current view; the *behind* relation is evaluated to set the correct direction of the link;
3. a topological sort of the graph, starting from the cells which do not obstruct any other cell, returns a correct inverse visible order.

The **behind** relation is evaluated, for each couple of adjacent cells, taking into account the shared face and the position of the viewpoint. The shared face defines a plane and two halfspaces. The link is therefore directed toward the cell whose half–space contains the viewpoint. To implement this, the plane equation for the shared face has to be evaluated at the viewpoint. The plane coefficients are computed and stored in the preprocessing phase (together with the adjacency graph).

The MPVO algorithm has a linear cost, but requires substantial geometric tests: when the viewpoint changes, all of the adjacency link orientations have to be fixed (i.e., an evaluation of a linear equation for each link).

The cost of MPVO has been empirically evaluated on the BluntFin [2] and the BuckyBall [3] datasets, represented using a multiresolution scheme [1]. The results are reported in Table 1, where: T_{adj} is the time for the construction of the DAG (view independent, thus preprocessing), T_{orient} is the time for the orientation of the adjacency links, and T_{top_sort} is the time for the topological sort. Times are in

[2]Produced and distributed by NASA–Ames Research Center; it is defined on a curvilinear grid composed of $40 \times 32 \times 32 = 40,960$ samples.

[3]Courtesy of AVS International Center; it represents the electron density around a molecule of C_{60} and is defined on a $32 \times 32 \times 32$ regular grid.

Dataset	Resolut. vertices	tetra	T_{adj}	T_{orient}	Topol. Sort T_{top_sort}	Num. Sort T_{dist}	T_{Qsort}	$T_{n\_d\_sort}$
BuckyBall								
error 0%	32,768	176,687	41.14	2.09	1.13	0.18	0.78	0.96
error 2%	10,808	63,649	13.67	0.68	0.35	0.07	0.24	0.31
error 5%	6,010	35,909	7.62	0.38	0.19	0.04	0.12	0.16
error 10%	3,088	18,567	3.78	0.18	0.09	0.02	0.06	0.08
BluntFin								
error 2%	2,279	13,094	2.59	0.12	0.08	0.02	0.04	0.06
error 4%	1,012	5,615	1.08	0.05	0.03	0.01	0.02	0.03

Table 1: Resolution of the test datasets and times, in seconds.

cpu seconds on an SGI Indigo R4000 XS24 workstation.

Max et al. reported [4] that, given a Delaunay triangulation and a viewpoint, a visibility ordering is always defined. In addition, reporting results from Edelsbrunner and Joe, they proved that a depth sort can be simply computed without having to store and manage topological data.
This alternative solution, here called **numeric distance sort**, computes for each tetrahedron t_i :

1. the center c_i and the radius r_i of the sphere circumscribed to t_i (such data are view independent, and are thus computed in a preprocessing phase);
2. the square of the distance d_i from the viewpoint to each center c_i;
3. the square length of the segment starting from the viewpoint and tangent to the sphere circumscribed to t_i, i.e. ${l_i}^2 = {d_i}^2 - {r_i}^2$;

It has been proved [4] that the visibility sort corresponds to the sort of segments l_i with respect to their length: a cell t_1 is behind t_2 *IFF* $l_1 < l_2$. To compute visibility using this approach we need:

preprocessing – to compute and store centers and radii of nt circumspheres;

for each view – to compute $2 * nt$ distances and to sort nt real numbers.

where nt is the number of tetrahedra in the triangulation.

In the rightmost columns in Table 1 we report the times required to sort the datasets with the **numeric distance** sort. The total time, $T_{n\_d\_sort}$, is split into the two subphase times: T_{dist} is the cost of computing distances, and T_{Qsort} is the cost of sorting these distances using an optimized version

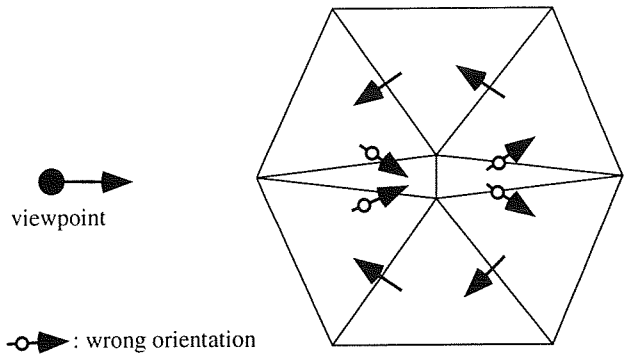


Figure 1: Four erroneous link orientations generate a region which cannot be sorted.

of the Quicksort algorithm.

A comparison of the two sorting approaches can be driven by the evaluation of the implementation complexity, numerical instability, and efficiency.

The implementation of the second solution is straightforward; it does not involve visiting a complex DAG, and only requires the evaluation of simple mathematical functions and the use of a numeric sorter.

Numerical instability, extremely probable when high resolution datasets are managed, is greatly reduced using the second solution. Obviously, imprecise distances can be computed but this does not halt the sorting process. This problem might at most lead to the incorrect depth ordering of some tetrahedra.

On the other hand, if the first approach is adopted precision is highly critical because errors in the orientation of the links may result in detecting false cycles. More dramatically, in some cases an incorrect ordering of one or more links can prevent an entire region in the grid from being sorted, as is shown in the 2D example in Figure 1.

To evaluate the efficiency of the two sorting solutions, we must take into account that the graph orientation is not required to sort the data by the *numeric distance* sorter, but this orientation is needed in the subsequent classification phase. Therefore, this processing is mandatory whichever sorting strategy is chosen, and its cost is thus not included in the T_{top_sort} times reported in Table 1.

The analysis of the results shows that both for efficiency reasons and higher numerical stability the *numeric distance* sort is the correct choice.

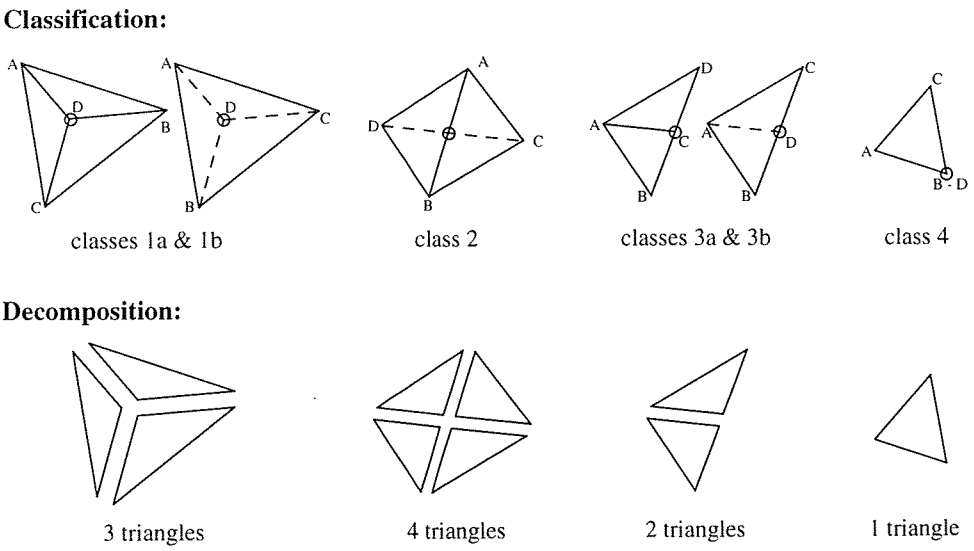


Figure 2: Classification and decomposition into triangles of a tetrahedral cell depending on the current profile in viewing space (○: NZT vertex).

3.2 Projection (classification + rendering) optimization

Once the cells have been depth–sorted, standard PT operates on each cell as follows. Each cell is projected in screen space and its projected silhouette is classified according to Figure 2. Only four projections are possible: two of them which are combinatorially different plus two degenerated cases. The frequency of the degenerated classes 3 and 4 was estimated in less than 5% of the total [10], and therefore a faster management of classes 1 and 2 is taken into consideration in most approximations. Classification of cells directly derives from the orientation of the adjacency links.

For each class, a *non zero thickness* (**NZT**) vertex is determined. The NZT vertex (a single point in screen space) corresponds to two points on the frontier of the cell. The standard PT algorithm computes the density and the color of the NZT vertex by evaluating the thickness of the cell in correspondence to the line of sight passing through it. A particle volume density model [11] is adopted. The projected silhouette is then split into triangles (Figure 2). Each triangle is rendered and blended in hardware as a Gouraud–shaded facet, with a non zero opacity assigned to the NZT vertex alone. Stein et al. [7] proved that the PT method may produce artifacts, and proposed a solution based on the use of hardware assisted texture mapping. For the sake of simplicity, here we will consider the

standard PT method output as *the* correct image, without considering possible artifacts.

There are two possible strategies which can reduce the cost of the projection phase: reducing the number of scan-converted facets and/or avoiding the color and opacity computations for the NZT vertices. With these strategies in mind, the optimization techniques evaluated here are:

- **Voxel** [8]: in this very rough approximation, color and opacity (irrespective of the cell thickness) are precomputed for each cell as an average of the values of its vertices; at projection time, all visible faces are rendered using flat shading;
- **Uniform Thickness Slab (UTS1)** [8]: color and opacity are interpolated on the projected silhouette, but the cells are considered as having uniform thickness. The computation of the NZT vertex is therefore avoided, and the corresponding cell thickness is not taken into account. A slight variation of this approximation, **UTS2**, in the case of class 1a visualizes the back face alone instead of the three front faces, thus reducing the number of split facets;
- **Centroid**: this new approximation avoids computing NZT coordinates and thickness by using the centroid of the cell as a splitting vertex, irrespective of the current view. Centroids data (coordinates, color and opacity) are computed in a preprocessing step, thus reducing the computations needed at run time.

To compute centroids data, for each cell the center c of the inscribed sphere and its radius r are computed by solving a linear system of four equations in four unknowns:

$$-\frac{a_i x_c + b_i y_c + c_i z_c + d_i}{\sqrt{{a_i}^2 + {b_i}^2 + {c_i}^2}} = r \tag{1}$$

where the coefficients a_i, b_i, c_i, d_i are the coefficients of the plane on which face i of the tetrahedral cell lies. The scalar value of the centroid is then computed by a simple linear interpolation.
Centroid approximation manages the splitting phase as follows:

- **class 4:** the same as the PT algorithm;
- **class 3:** the same as the PT algorithm, but the cell thickness is set to $c_1 * r$, where c_1 is an empirical constant;
- **class 2:** the splitting vertex and value are in this case the centroid's coordinates and value; the cell thickness is set to $c_1 * r$;

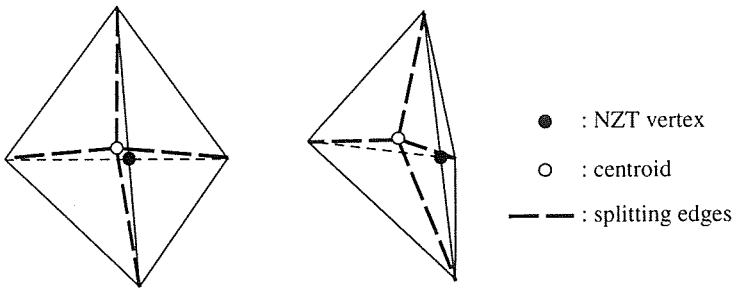


Figure 3: Different projections of class 2 cells.

- **class 1:** the coordinates of the splitting vertex are the coordinates of the NZT vertex (they are combinatorially known, no geometric computations are required). The value of the splitting vertex is set to the mean value between the centroid and the NZT vertex values. The cell thickness is set to $c_1 * r$.

The Centroid approximation therefore avoids the computation of:

- one edge intersection and two color&opacity interpolation for the cells in class 2;
- one plane–ray intersection and one color&opacity interpolation for the cells in class 1;
- one bilinear interpolation and one color&opacity interpolation for the cells in class 3.

Our experiments report that the best approximations were obtained using values of $c_1 = 2.5 \div 3.0$. Obviously, the approximation given by the centroid method is dependent, for each cell, on the distance between the position of the centroid and the current view–dependent position of the NZT vertex (Figure 3). In order to reduce this approximation error, the opacity of the vertices on the boundary of the projected silhouette was set to 0.3 times the value of the splitting vertex opacity.
When applied to faces which are parallel to the line of sight the method of the centroid may therefore result in a poor approximation (e.g., the rightmost cell in Figure 3). This may be common in the case of datasets obtained by decomposing a regular or curvilinear dataset into tetrahedra, but is much less frequent when the tetrahedral dataset is obtained from unstructured data or from the simplification of a curvilinear or regular dataset [1].

	PT	Centroid	UTS1	UTS2	Voxel
Bluntfin 2% error					
T_{render}	0.73	0.59	0.36	0.29	0.32
T_{vis}	0.91	0.77	0.54	0.47	0.50
$\frac{T_{render}(approx.)}{T_{render}(PT)}$	1.	0.80	0.49	0.39	0.43
$\frac{T_{vis}(approx.)}{T_{vis}(PT)}$	1.	0.84	0.59	0.51	0.54
BuckyBall 0% error					
T_{render}	12.19	8.67	5.02	4.02	4.41
T_{vis}	15.24	11.64	8.07	7.07	7.46
$\frac{T_{render}(approx.)}{T_{render}(PT)}$	1.	0.71	0.41	0.32	0.36
$\frac{T_{vis}(approx.)}{T_{vis}(PT)}$	1.	0.76	0.52	0.46	0.48

Table 2: Times on two datasets using PT and the evaluated approximations (times in seconds).

Our comparison of the standard PT algorithm and the three approximations considers both processing times and image quality.

The processing times are reported in Table 2, where T_{render} is the time required to project, scan convert and compose the tetrahedra, and T_{vis} is the total time ($T_{vis} = T_{orient} + T_{n\_d\_sort} + T_{render}$, thus all but the preprocessing T_{adj}; for the other results see Table 1). Centroid times were half the PT times, but about 30% higher than with the UTS2 method.

An evaluation of the rendering quality can be based on a subjective comparison (a visual analysis) or on a more objective one, based on the computation of differences between the images. The results of objective numeric comparisons are reported in Table 3. Images are compared by computing, for each corresponding pixel pair, the square length of the vector difference:

$diff(pix_1, pix_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$.

The "per pixel" mean value of the difference is reported in the table (with pixels colors represented in the range 0..255).

Visual results are reported in the color plates: in Figures 4, 5, 6 and 7 we show the results of classical PT and Voxel, UTS1 and Centroid approximations; in the Figures 8, 9 and 10 the difference images are presented.

To evaluate rendering quality we must differentiate between data obtained by the decomposition of regular hexahedral cells or by the triangulation of irregularly distributed pointsets. On the first datasets, UTS2 approximation gives better images in shorter times. The BuckyBall results in Table 3 seem to be in contradiction with this assertion but in this case, although the global error of Centroid

	Mean difference
Bluntfin (2% error)	
PT vs. UTS2	14.65
PT vs. Centroid	10.85
BuckyBall (0% error)	
PT vs. UTS2	11.55
PT vs. Centroid	8.11

Table 3: Numerical comparison of the images (standard PT algorithm, UTS2 and Centroid approximations).

is lower than that of UTS2, the Centroid error is often located in restricted areas and therefore is more evident. On the second class of datasets, the insensitivity of UTS2 to the thickness of the cells makes Centroid the method that gives better visual and numerical results, though if at the cost of longer processing.

4 Optimization via data simplification

A multiresolution representation for volume datasets has been proposed in a previous paper [1]. Multiresolution is achieved through a sequence of tetrahedralizations approximating the scalar field at increasing precision. The construction of the model is based on an adaptive incremental triangulation approach driven by the local coherence of the scalar field. The result is a number of Delaunay triangulations (i.e. a pyramidal structure) in which each level i gives an approximation of the volume dataset with an error not exceeding a threshold ϵ_i.
Obviously, lower resolution levels may be rendered when faster visualization is required, e.g. for setting visualization parameters such as the current view or the transfer function used to map the field scalar values into colors and densities.

Here, we have experimented with a multiresolution representation to speedup the projection process and evaluate the quality of the images generated.

In order to estimate the performances of the rendering process based on the simplification of the data, we ran the PT code on a number of simplified representations of the two test datasets, Bluntfin and Bucky. The processing times reported in Table 4 are the preprocessing time (DAG creation) and

Dataset	no. tetra	Times Preproc.	T_{vis}	Speedup	Image Error (mean value)
BuckyBall					
error 0%	176,687	41.14	15.24	1.	0.
error 2%	63,649	13.67	5.02	3.03	1.91
error 5%	35,909	7.62	2.78	5.48	4.62
error 10%	18,567	3.78	1.42	10.73	10.02
BluntFin					
error 2%	13,094	2.59	0.91	1.	0.
error 4%	5,615	1.08	0.39	2.33	3.35
error 6%	2,796	0.52	0.21	4.33	5.39

Table 4: Visualization times obtained usinga multiresolution representation of the test datasets (times in seconds).

the total visualization time (DAG orientation, numeric distance sort, visualization via PT).
The images produced using the standard PT are reported in Figures 11 and 12, while the image differences computed on an output of the PT algorithm on BluntFin at precision 2% are in Figures 13 and 14.

By comparing the results in Tables 2, 3 and 4 it can be noted that images with approximations similar to those obtained using UTS2 are now produced using datasets with a substantial simplification (e.g. on BuckyBall, a similar approximation is obtained using the multiresolution level with a 10% error), and therefore with much shorter processing times: about 8 sec. for UTS2 images and 1.5 sec for an image using the multiresolution representation.

5 Conclusions

In this paper we have focused on the projective visualization of datasets represented via tetrahedral tessellations. Speedup techniques have been reviewed and a new approximation method has been proposed. Approximation methods have been compared in terms of both processing times and the quality of the images produced.

In addition, the results obtained by the optimization of the rendering process have been compared with those obtained by adopting a data simplification approach. Approximated representations of the test datasets have been built using an adaptive incremental triangulation approach driven by the local

coherence of the scalar field and are represented in a multiresolution scheme. We have showed through numerical and visual comparisons that data simplification produces images with a comparable level of approximation in shorter times. This proves that data simplification is the key approach to reducing visualization times, especially in those interactive phases where the user tolerates a high approximation degree and where timely response to user input is crucial.

Acknowledgments

This work is part of a joint project between CNR and the University of Genova, which regards the representation and visualization of volume data by the use of simplicial complexes. We therefore thank Leila De Floriani (University of Genova) and Enrico Puppo (I.A.C.–C.N.R., Genova) for their valuable suggestions and discussions.

References

[1] 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.

[2] 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.

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

[4] N. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3D scalar functions. *A.C.M.Computer Graphics (Proc. of 1990 WS on Volume Visualization)*, 24(5):27–33, 1990.

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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] P.L. Williams and N. Max. A volume density optical model. In A.E. Kaufman and W.E. Lorensen, editors, *1992 Workshop on Volume Visualization*, pages 61–68. ACM Press, 1992.