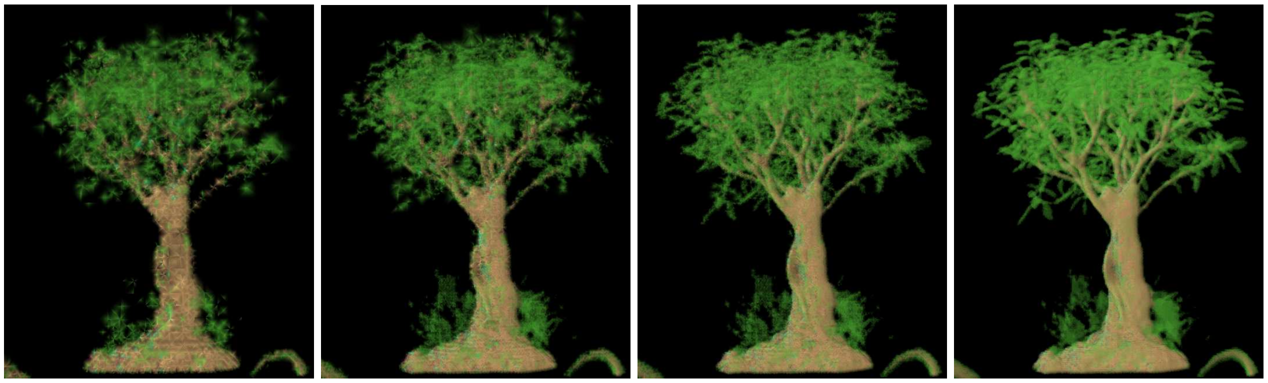
3D ROAM for Scalable Volume Visualization

Stéphane Marchesin* ICPS/IGG
Jean-Michel Dischler† INRIA Lorraine, CALVI Project
Catherine Mongenet‡ ICPS

LSIIT – Université Louis Pasteur – Strasbourg, France



(a) 2.5 fps – 104000 tetrahedra (b) 1 fps – 380000 tetrahedra (c) 0.45 fps – 932000 tetrahedra (d) 0.15 fps – 3065000 tetrahedra

Figure 1: The bonsaï data set at different detail levels. Frame rates are given for a 1024×1024 resolution

Abstract

The 2D real time optimally adapting meshes (ROAM) algorithm has had wide success in the field of terrain visualization, because of its efficient error-controlling properties. In this paper, we propose a generalization of ROAM in 3D suitable for scalable volume visualization. Therefore, we perform a straightforward 2D/3D analogy, replacing the triangle of 2D ROAM by its 3D equivalent, the tetrahedron. Although work in the field of hierarchical tetrahedral meshes was widely undertaken, the produced meshes were not used for volumetric rendering purposes. We explain how to compute a bounded error inside the tetrahedron to build a hierarchical tetrahedral mesh and how to refine this mesh in real time to adapt it to the viewing conditions. We further show how to achieve cell sorting in linear time, thus yielding real time view-dependent display of the volumetric object. We present examples of large volume data sets and compare our approach with a similar one. Our results outline the high quality and computational efficiency of our approach.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation – Viewing algorithms— [I.3.5]: Computational Geometry and Object Modeling—Object hierarchies

Keywords: hierarchical tetrahedral meshes, ROAM, volume rendering, level of detail

*e-mail: marchesin@icps.u-strasbg.fr
†e-mail: dischler@dpt-info.u-strasbg.fr
‡e-mail: mongenet@dpt-info.u-strasbg.fr

1 Introduction

Because of continuous improvements in simulation and data acquisition technologies, the amount of 3D data that visualization systems have to handle is increasing rapidly. Large datasets offer an interesting challenge to visualization systems, since they cannot be visualized by classical visualization methods. Such datasets are usually many times larger than what a video card's memory can hold. For example, a $1024 \times 1024 \times 446$ voxel dataset uses 446MB of memory. Rendering such datasets "as-is" is thus not possible. Therefore, there is a need for volume visualization methods that can both handle larger data sets, and have good scalability properties.

In this paper, we straightforwardly generalize the ROAM algorithm [5] to volumetric rendering. Though work has already been undertaken in this direction (see section 2), this paper proposes a new original approach, by introducing a 2D-3D analogy : we consider that the tetrahedron is the 3D equivalent of the triangle, and that the 3D scalar function is the 3D equivalent of the height function. We first apply a hierarchical decomposition of the 3D space similar to the 2D triangle binary tree. We use a hierarchical tetrahedral mesh as described in [22], which can be represented using a binary tree, where each node is a tetrahedron. Our contribution is a technique for building and refining a 3D mesh on-the-fly using two basic operations : node fusion and node split. To decide whether a split or a fusion should take place at the current node, we must compute the approximation error associated with an arbitrary tetrahedron of the hierarchy. For this purpose, we detail an algorithm suitable for integer point enumeration inside a tetrahedron of the hierarchy. Once computed, the mesh is rendered using the classical projective tetrahedra algorithm [18]. In this paper, we further show how to avoid an explicit cell-sorting pass, as cell-sorting can be performed during the rendering pass, and in linear time. We also show how to maintain mesh conformity. As a result, our method allows a high quality smooth level of detail rendering (see figure 1) depending on the desired balance between speed and quality. The number of tetrahedra thereby depends on the viewing conditions, as for traditional ROAM.

The paper is organized as follows : in section 2 we present some related works. Section 3 introduces the mesh construction principles. Section 4 shows how to compute the approximation error inside a tetrahedron and how to decide whether or not a fusion or split operation is needed. In section 5, we show how to build and refine the mesh using the computed error. In section 6, we explain how cell-sorting and rendering is achieved in linear time with respect to the number of cells. Section 7 is dedicated to experimental results as well as a comparative study. Finally we give some concluding remarks and discuss further improvements.

2 Related works

Many papers address the issue of large volumetric dataset visualization. Different approaches are commonly used, which can be divided into two main groups : texturing-based methods and cell projection-based methods.

The first class of methods uses only graphics hardware to perform real time rendering, for example using 3D textures. Engel et al. [6] sum up approaches using the graphics hardware. However, such methods are limited by the available video memory. Later techniques have subsequently been developed to circumvent this limitation [12, 16, 11].

The second class of methods performs rendering of objects made of volumetric cells by projecting each of these cells, and are thus called "cell-projection" methods. Such methods allow rendering volumes with a high degree of quality. Among the numerous cell projection methods, the projective tetrahedra algorithm by Shirley et al. [18] uses the graphics hardware to render tetrahedral cells. Each tetrahedron is decomposed into a different number of triangles according to the observer's position, and these triangles are then sent to the video card for rendering. This method has been improved by later papers on the subject. To compensate for linear alpha interpolation, Stein et al. [19] use a texture as an opacity lookup table. Wylie et al. [21] use vertex shaders to achieve view independent projective tetrahedra rendering. Guthe et al. [10] use pixel shaders to achieve higher rendering quality.

Since cell projection methods use alpha blending to accumulate each cell's contribution, a cell-sorting pass is required to obtain visually correct back-to-front rendering. Recent methods allow lowering the complexity of this sorting pass [3], without reaching the optimal linear complexity. Weiler et al. [20] use the graphics hardware to achieve ray casting of a tetrahedral mesh. However, this approach is not suitable for large tetrahedral meshes.

In another context, i.e. in the field of terrain rendering, Duchaineau et al. [5] propose a method suitable for 2D height map visualization called ROAM. This method is based on a hierarchical decomposition of a surface into triangles that approximates the height map, and allows bounding the error associated with this approximation.

The approach uses progressive dynamic meshing, i.e. the closer to the observer a part of the mesh is, the denser it gets. By dynamically adapting the mesh to the viewing conditions, the ROAM algorithm allows visualization of large height maps.

Methods for generating hierarchical 3D meshes have also been studied. These methods use different subdivision schemes to obtain such meshes. A tetrahedral mesh is used in [22, 9] for surface visualization. Cignoni et al. [2] study the simplification of generic tetrahedral meshes, while preserving mesh topology. Pascucci [15] studies subdivision meshes in arbitrary dimensions. Mello et al. [14] apply a spatial multiresolution decomposition to volume rendering using the projective tetrahedra algorithm. Roettger and Ertl [17] use an octree-based hierarchical mesh for level of detail volume visualization : the leaf cube nodes are split into five tetrahedra each, which are in turn rendered using the projective tetrahedra algorithm [18]. This approach is the closest to ours, since it uses a hierarchical 3D mesh rendered using the projective tetrahedra algorithm. We use a similar subdivision scheme where the subdivision depends on an error computation. However, our mesh construction technique differs in many points : in our case, a tetrahedral mesh is directly built from the voxel object, thus generating only one tetrahedron per node. As a consequence, we introduce a completely different error controlling technique, a new mesh conformity technique and a new view dependent visualization technique. These are much closer to the conventional 2D ROAM. The gain of these efforts is to yield a number of tetrahedra adapted to the viewing conditions, as for 2D-ROAM.

3 Principles of mesh construction

Zhou et al. [22] describe a hierarchical tetrahedral mesh construction technique in the framework of isosurface visualization. The initial subdivision of the voxel data is obtained by dividing the bounding cube into 12 tetrahedra (see figure 2).

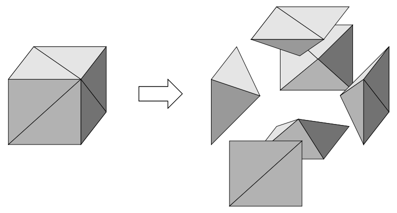


Figure 2: Initial volume subdivision into 12 tetrahedra.

The hierarchy is obtained by recursively splitting each of these tetrahedra into two tetrahedra along its longest edge. This scheme leads to a regular subdivision creating three classes of tetrahedra (see figure 3) :

- exactly one face is parallel to a coordinate plane, and two edges of the face are parallel to a coordinate axis (case a)
- exactly one face is parallel to a coordinate plane, and exactly one edge of the face is parallel to a coordinate axis (case b)
- two faces are parallel to a coordinate plane (case c)

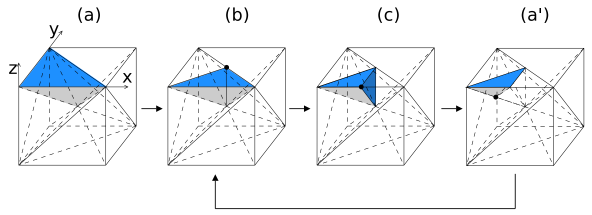


Figure 3: Hierarchical subdivision steps.

In the next two sections, we respectively describe the error computation and the mesh construction technique.

4 Error computation

In order to achieve mesh construction, we use two basic operations : node split when refining a node downwards, and node fusion when refining a node upwards. The next two subsections detail how the error is computed in the case of a node split, and how an error bound is obtained in the case of a node fusion.

4.1 Error computation during node split

The error must be computed for each tetrahedron of the hierarchy. To measure the error we have to compute the difference between voxel values and values interpolated at integer points inside the tetrahedron. To achieve this, we need to enumerate each integer point inside a given tetrahedron of the hierarchy.

Many algorithms for integer point enumeration inside a generic polyhedron exist [1] ; they are usually based on Fourier–Motzkin elimination [4]. However, due to their genericity, these algorithms are not suitable, with regard to speed, for real time point enumeration. Thus, we have developed our own point enumeration algorithm using the specific properties of the tetrahedra of the chosen hierarchy. This algorithm is based on the decomposition of a tetrahedron into slices, and subsequent point enumeration inside the resulting triangular slices.

Property 1 *Tetrahedra of the hierarchy have their edge vectors coordinates in* $\{-1,0,1\}$.

Property 2 *Tetrahedra of the hierarchy have one of their faces parallel to one of the coordinate planes (Oxy, Oxz, Oyz).*

The hierarchical subdivision of the hierarchy as described in [22] generates three classes of tetrahedra. To prove properties 1 and 2, we first examine one member of each of these classes, and then generalize the proof to the whole hierarchy.

Let us consider the three representatives of these classes as shown on figure 3 Their edge vectors are given on figure 4. These vectors have their coordinates in $\{-1,0,1\}$. Thus, tetrahedra (a), (b) and (c) verify property 1. Moreover, each of these three tetrahedra has at least one of its faces parallel to one of the base planes (the base planes are one of the coordinate planes *Oxy*, *Oxz* or *Oyz*, and the faces are shown in blue/dark on figure 3), and thus verify property 2.

Tetrahedron a	Tetrahedron b	Tetrahedron c
$(0,1,0)$	$(1,1,0)$	$(1,1,0)$
$(1,0,0)$	$(1,0,0)$	$(1,0,0)$
$(1,-1,0)$	$(1,-1,0)$	$(0,1,0)$
$(-1,1,1)$	$(1,-1,1)$	$(0,1,-1)$
$(1,1,-1)$	$(1,1,-1)$	$(1,1,-1)$
$(1,-1,1)$	$(0,0,1)$	$(0,0,1)$

Figure 4: The edge vectors along the six edges of the three kinds of tetrahedra of the hierarchy as shown in figure 3.

To generalize properties 1 and 2 to all the tetrahedra of the hierarchy, we need to examine the operations that transform a tetrahedron of a given class into another of the same class :

- a rotation of $\frac{k\pi}{2}$ angle, $k \in \mathbb{Z}$,
- an homothety of 2^n ratio.

Each of these transformations preserves property 1, so that any composition of these transformations preserves the property too. Thus, all the edge vectors of the tetrahedra of the hierarchy have their coordinates in $\{-1,0,1\}$. Similarly, these transformations keep property 2. Therefore, all the tetrahedra of the hierarchy have one of their faces parallel to one of the base planes.

Using property 2, a tetrahedron can be easily decomposed along one of these planes into parallel triangular slices containing the integer points (see figure 5). To switch from one triangular slice to another, edge vectors along the edges of the tetrahedron have to be computed. Property 1 ensures that all points enumerated using these edge vectors are integer points. Thus, we can derive a simple enumeration algorithm that scans all the integer points inside a tetrahedron of the hierarchy.

This algorithm uses the edge vectors associated with a tetrahedron (see algorithm 1). We first split the tetrahedron into slices, and then compute the edge vectors to go from one slice to another. We subsequently compute the edge vectors inside a triangular slice (see figure 6) and finally iterate each integer triangle line.

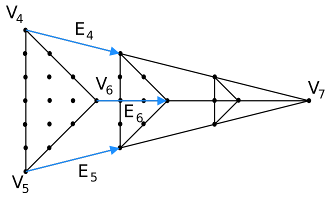


Figure 5: Splitting a tetrahedron into slices for point enumeration.

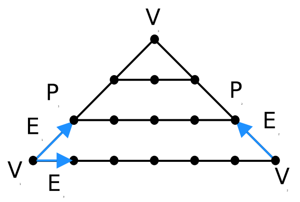


Figure 6: Enumerating points inside a triangular slice.

Algorithm 1 Error computation by point enumeration inside a tetrahedron of the hierarchy

```
 1: Find one face of the tetrahedron parallel to one of the base planes
 2: Cut the tetrahedron into slices parallel to this plane
 3: for each triangular slice $V_1V_2V_3$ do
 4:     $E_1$ =edge vector along $V_1V_3$
 5:     $E_2$ =edge vector along $V_2V_3$
 6:     $P_1 = V_1$
 7:     $P_2 = V_2$
 8:     repeat
 9:         for each point $P$ from $P_1$ to $P_2$ along $E_3$ do
10:             error=max(error,error($P$))
11:         end for
12:         $P_1 = P_1 + E_1$
13:         $P_2 = P_2 + E_2$
14:     until $P_1$ has reached $V_3$
15:     error=max(error,error($V_3$))
16: end for
```

To compute the error associated to a tetrahedron, we need to iterate each point inside this tetrahedron. However this operation has a high cost. In order to speed up hierarchy refinement during visualization (see section 6), the hierarchical mesh along with errors for the current object are cached until a certain depth. This is even more beneficial, as tetrahedra of lower depths have more points to iterate, thus their error is more expensive to compute than the error of tetrahedra of higher depths. However, it is not possible to cache the full hierarchy because a hierarchy at full detail might be many times larger than the original voxel data.

Knowing how to compute the error for a given tetrahedron, we can now adjust it to the view dependent case. Considering a given tetrahedron T viewed from an observation point Obs, the error function we introduce depends on three criteria : the error inside the tetrahedron $e(T)$ (obtained either by an error bound, or by extensive point enumeration), the distance between the tetrahedron and the observer $d(Obs,T)$, and whether or not this tetrahedron lies inside the view frustum :

$$e(T,Obs) = \begin{cases} 0 & \text{if T lies totally outside the view frustum} \\ \frac{e(T)}{d(Obs,T)} & \text{otherwise.} \end{cases} \tag{1}$$

Intuitively, this formula means that the error is inversely proportional to the distance to the observer and becomes minimal when the point is invisible.

4.2 Error bound computation during node fusion

To avoid the cost of error computation while merging two tetrahedra, we must propagate the error values upwards the tree. This means we need to find, given the error of two tetrahedra, an upper bound of the error of their parent tetrahedron. This is achieved by generalizing the 2D case from ROAM. In the original ROAM approach, Duchaineau et al. use a hierarchical error bound to avoid error computation during node fusion. To minimize the computations, we extend this concept to 3D. Thus, we do not need to compute the error while merging two tetrahedra since we can simply obtain a bound of this error by generalizing the following formula used for 2D ROAM : In 2D ROAM, when merging two triangles T_0 and T_1 (depicted in blue on figure 7(a)) into a single triangle T (depicted in red on the figure) and if $z(p)$ is the height of the 2D point p and v_c is the central vertex, the error value e_T for T can be bounded as follows :

$$e_T \leq \max\{e_{T_0}, e_{T_1}\} + |z(v_c) - z_T(v_c)| \tag{2}$$

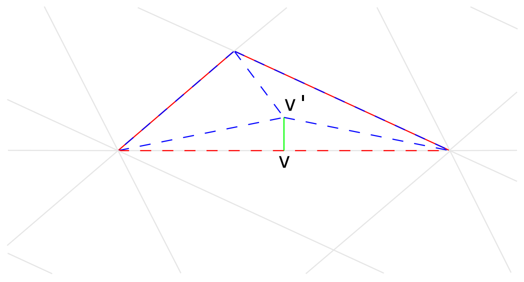
This formula gives an error bound provided that we already know the error for the two children of a triangle in the hierarchy. We obtain a similar hierarchical error bound of the volumetric error by changing the variables meaning according to our 2D-3D analogy : when merging two tetrahedra T_0 and T_1 (shown in blue on figure 7(b)) into a single tetrahedron T (shown in red on the figure), we can replace the height $z(p)$ of point p by its scalar value $d(p)$ and obtain a similar 3D error bound :

$$e_T \leq \max\{e_{T_0}, e_{T_1}\} + |d(v_c) - d_T(v_c)| \tag{3}$$

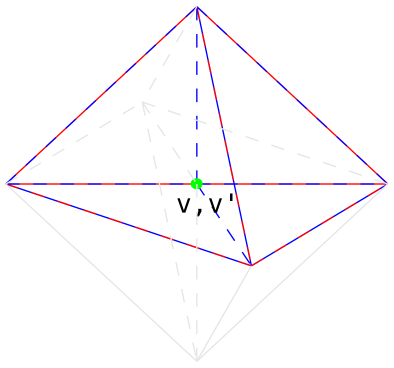
This bound is suitable for hierarchical evaluation of the error function, as used in 2D ROAM.

5 Mesh building

In order to build the tetrahedral mesh, two basic operations are used : tetrahedron split and tetrahedron fusion. To achieve view-



(a) Error bound in 2D ROAM



(b) Error bound in 3D ROAM

Figure 7: Error bound in 2D and 3D ROAM

dependent rendering, we keep the hierarchy from one frame to another, and modify only parts that need adjustment according to the error described in the previous section (section 4).

At each frame, the hierarchy is traversed recursively, and the error is considered according to the node's position in the hierarchy :

- if the node is a leaf and its error meets the error criterion as described in the next section, the node is kept as is. However, if the error criterion fails the leaf node is split into two new children nodes. These nodes can be in turn considered for further splitting.
- if the node is an ante-leaf (i.e. a node whose children are leaves) and satisfies the error criterion locally, a node fusion takes place and the two children nodes are discarded.

Mesh conformity is enforced by propagating the node splits to the adjacent nodes. When splitting a tetrahedron into two (as shown on figure 8), the middle of its longest edge is chosen as a point of bisection (this point is called central vertex in [9] and is shown in red on the figure). This point uniquely determines the cutting plane of the tetrahedron (shown in dashed red on the figure). The 4 faces of a tetrahedron can be classified into two groups : 2 of these are base faces (the 3D equivalent of the base edge of ROAM) and the 2 other are non base faces. On figure 8, splitting tetrahedron (1) into two children nodes has an impact only on the tetrahedra sharing base faces, i.e. three other tetrahedra (tetrahedra (2), (3) and (4) on the figure). To enforce mesh conformity, node splits are recursively propagated to neighbors across the base faces. However, this is not sufficient to enforce mesh conformity for non base faces. To enforce conformity in this case, we extend the idea of ROAM

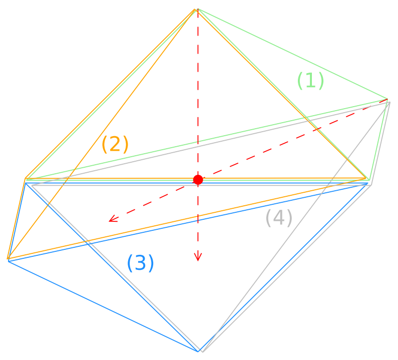


Figure 8: Propagating the conformity along the base faces

and enforce a maximum difference of one level of depth between arbitrary adjacent tetrahedra sharing a non base face.

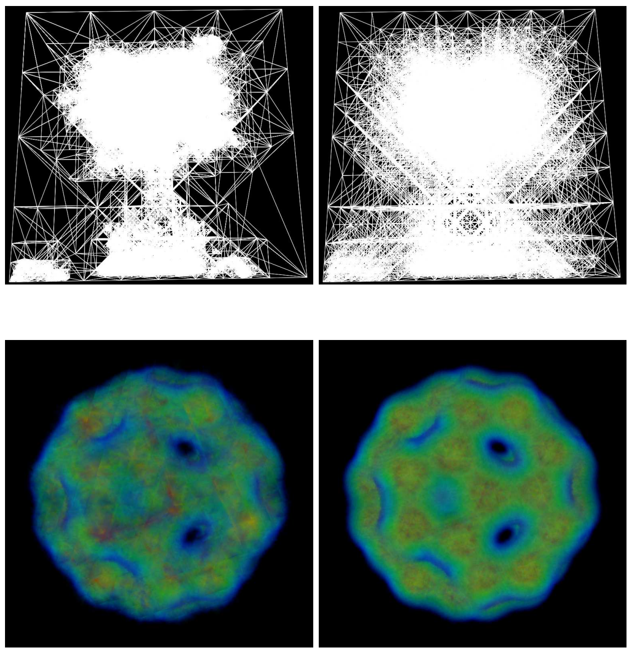


Figure 9: The influence of mesh conformity. Left column : non-conforming mesh ; right column : conforming mesh.

6 View-dependent rendering

To get a visually correct rendering, the cells of a volumetric mesh must be displayed back to front. Recent cell-sorting approaches allow sorting cells with low complexity [19, 3]. However, hierarchically splitting tetrahedra into two using a plane as described in [22] implicitly corresponds to building a binary space partition (BSP) as introduced in the early eighties by [7]. This approach is also used in the context of volumetric meshes in [14]. We can use it to sort the cells : the cell sorting pass is performed by recursively iterating the mesh hierarchy and choosing at each node which sub-node to process first. The complexity of such a sorting approach is optimal, since it is linear with the number of cells. Furthermore, it does not require building an additional data structure, as it can be achieved during hierarchy traversal. The hierarchy is recursively traversed as shown in algorithm 2. Each tetrahedron of the hierarchy is examined to test whether it is a leaf or not. If it is, the node is displayed. If it is not, i.e. the node is an internal node, its cut plane is computed. This plane subdivides the tetrahedron into two children tetrahedra. We then determine on which side of the cut plane the observer lies and accordingly call the drawing function for the two children nodes. Thus, tetrahedra further from the observer are displayed first and correct back-to-front rendering is achieved.

Algorithm 2 Rendering algorithm

```
1:  Hierarchical_Rendering(node n,viewpoint p)
2:  if ( n is a leaf node ) then
3:     n.Draw()
4:  else
5:     cp=n.Cutplane()
6:     if ( is_left(p,cp) ) then
7:        Hierarchical_Rendering(n.child(1),p)
8:        Hierarchical_Rendering(n.child(2),p)
9:     else
10:       Hierarchical_Rendering(n.child(2),p)
11:       Hierarchical_Rendering(n.child(1),p)
12:    end if
13: end if
```

During the hierarchy traversal, we render each leaf node using the projective tetrahedra algorithm [18]. Many improved versions of the projective tetrahedra algorithm exist ; it is possible to use any of these modified methods to render the tetrahedral mesh. In this paper we have implemented the improved method described by Stein et al. [19] to be able to run our technique on any type of graphics hardware, as it uses only basic texture mapping.

Our approach allows progressive rendering of the volumetric dataset by dynamically adjusting the detail level of the tree. Thus, we can maintain interactivity using a coarser mesh while the point of view is moving, and show a finer mesh when the observer's motion stops. To improve responsivity, we also bound the hierarchy refining time per frame by bounding the number of tetrahedra whose error can be computed within one frame.

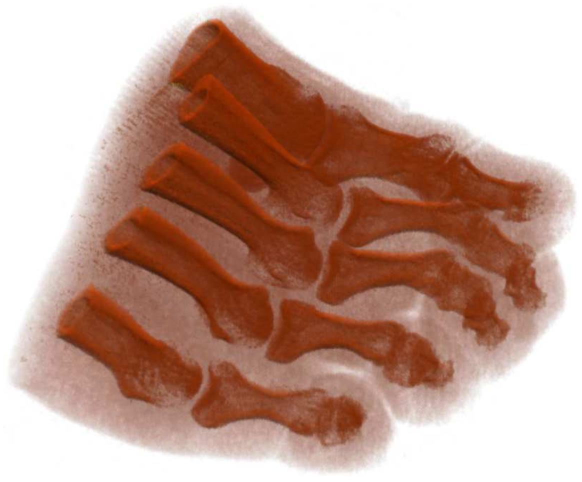


Figure 10: Rendering example for the foot dataset, non interactive rendering

7 Implementation and results

We have implemented our method using C++ and OpenGL, with and without mesh conformity enforcement. We have also implemented a volumetric rendering method based on the mesh described by Roettger and Ertl in [17] (this mesh uses an octree), however with a slightly different error computation technique to make the results comparable. We have conducted some benchmarks to measure the real time performance of our algorithm. These benchmarks and the associated frame rates were obtained at a 1024×1024, 32 bpp resolution, on a dual CPU Athlon MP 2000+ PC with a Geforce4 MX 440 video card (however, only one CPU was used during the benchmarks).

Results in number of tetrahedra as function of the error bound for the different algorithms are given on figure 11. These results show that our method generates less tetrahedra than [17], thus resulting in faster rendering. We explain this result by the fact that the mesh used in [17] generates five tetrahedra to render each node, while our mesh generates only one tetrahedron per node. However, enforcing mesh conformity incurs a cost in the number of tetrahedra, and this cost is clearly visible on the figure. As [17] does not give a technique to enforce mesh conformity, no comparison is possible on this topic. Mesh conformity also impacts picture quality, as shown by figure 9.

Our results also outline the scalability of the method, as they show that a reasonable number of tetrahedra is generated (as seen on figures 10, 11, 12), and more importantly that this number does not depend too much on the object's number of voxels but rather on its nature.

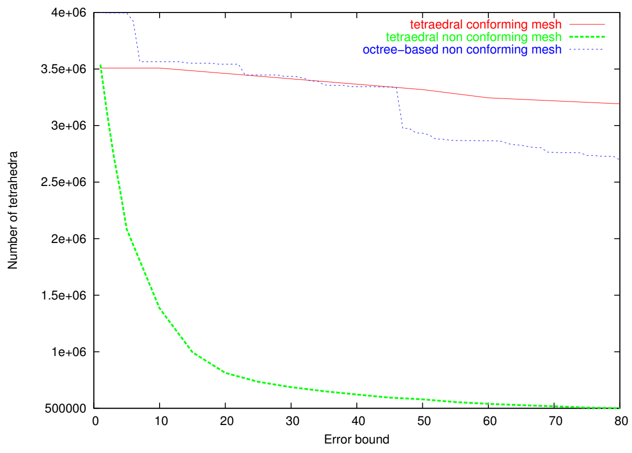
Figure 11 also demonstrates the scalability of our approach : the error bound allows direct control over the number of tetrahedra, which in turn is directly proportional to the frame rate. This figure also shows the consequences of enforcing mesh conformity in terms of number of tetrahedra.

Figure 12 exemplifies our results, from a performance as well as from a quality viewpoint, on miscellaneous objects : a plasma dataset ($64 \times 64 \times 64$ voxels), a fluid–solid interaction simulation dataset from [8] ($512 \times 256 \times 128$ voxels), a geological dataset ($1024 \times 1024 \times 446$ voxels) and a cloud dataset ($512 \times 512 \times 32$ voxels). The corresponding pictures, frame rates and number of tetrahedra are given for different mesh qualities, and conforming or non-conforming meshes.

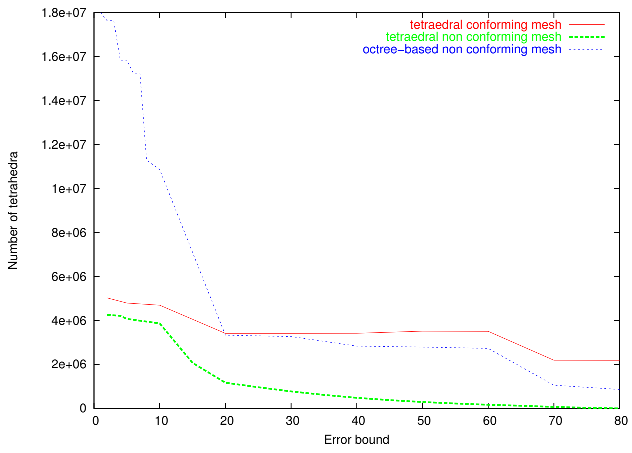
These results show that the overhead of enforcing the mesh conformity might not always be justified by the improvements in visual quality. Thus a method enforcing only visual continuity (like the one described in [17]) might be more suitable in some cases. Figure 1 shows different level of detail rendering for the bonsaï dataset with a conforming mesh.

Our work maintains interactivity and scalability during the data manipulation through the use of a level of detail approach. Using this approach, it is possible to have higher quality rendering during static observations and lower quality rendering during interaction with the model. To achieve interactivity on large models, we use a lower level of detail by stopping tree traversal at a certain depth. This depth is chosen in order to achieve good frame rates during interactive manipulations.

Our approach exploits both spatial and temporal coherence. Spatial coherence is used because a simplified structure (the mesh) is built over a complex one (the original voxel object) ; temporal coherence is exploited because the mesh hierarchy is kept from frame to frame and only the parts needing modifications trigger computations.



(a) Bonsaï dataset



(b) Foot dataset

Figure 11: The number of tetrahedra generated at different error levels for two datasets using a conforming tetrahedral mesh, a non conforming tetrahedral mesh, and a non conforming octree-based mesh

8 Conclusions

Our method allows interactive visualization and exploration of large data sets. For example, we are able to visualize geological datasets ($1024 \times 1024 \times 446$) in their entirety on a standard PC. While this method is designed from the beginning to be scalable, the rendering quality is similar to that of other cell-projection based methods when using a higher level of detail, and thus allows high quality images as well as real time rendering and as such is very versatile. Still, we are able to strictly control the approximation error of the tetrahedral mesh by imposing error bounds.

However, despite many advantages, our approach has one main limitation : due to the fact that the mesh size depends more on the object's nature than on its size, some objects are less suitable for visualization using our method than others. In particular, pure noise data can generate a mesh containing a number of tetrahedra proportional to the number of voxels of the original object when using a low error bound. Hence in such a case the hierarchy has to be very deep (down to the voxel level) to satisfy the error criteria. However, this limitation can be at least partially lifted by filtering the data before visualization.

Another limitation is the presence of small popping artifacts when the fusion/split operations take place. However, these artifacts could be removed using a geomorphing approach, similarly to what is done in [13].

Further improvements to our method are possible. For example, thanks to the use of a level of detail approach, it is possible to use the error metric as a way to interactively change an area of interest inside the object, pointing it using the mouse for example. Using the 4D equivalent of the tetrahedron and adapting our error computation model, the same idea could be applied to a 4D mesh for 3D+t visualization. Future works could also explore the adaptation of our approach to natively unstructured meshed objects.

References

[1] Alexander Barvinok and James E. Pommersheim. *An Algorithmic Theory of Lattice Points in Polyhedra*, pages 91–147. Cambridge University Press, August 1999.

[2] P. Cignoni, C. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of Tetrahedral Meshes with Accurate Error Evaluation. In *Proc. Visualization '00*, pages 85–92. IEEE, 2000.

[3] João Comba, James T. Klosowski, Nelson Max, Joseph S. B. Mitchell Claudio T. Silva, and Peter L. Williams. Fast polyhedral cell sorting for interactive rendering of unstructuredgrids. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 369–376. The Eurographics Association and Blackwell Publishers, 1999.

[4] George B. Dantzig and B. Curtis Eaves. Fourier–motzkin elimination and its dual. *Journal of Combinatorial Theory*, 14(3):288–297, 1973.

[5] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings of the 8th conference on Visualization '97*, page 81. IEEE Computer Society, 1997.

[6] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16. ACM Press, 2001.

[7] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133. ACM Press, 1980.

[8] Olivier Génevaux, Arash Habibi, and Jean-Michel Dischler. Simulating fluid-solid interaction. In *Graphics Interface*, pages 31–38. CIPS, Canadian Human-Computer Commnication Society, A K Peters, June 2003. ISBN 1-56881-207-8, ISSN 0713-5424.

[9] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces, 2001.

[10] Stefan Guthe, Stefan Roettger, Andreas Schieber, Wolfgang Strasser, and Thomas Ertl. High-quality unstructured volume rendering on the pc platform. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 119–125. Eurographics Association, 2002.

[11] Stefan Guthe and Wolfgang Strasser. Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics*, 28(1):51–58, February 2004.

[12] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straer. Interactive rendering of large volume data sets. In *Proceedings of the conference on Visualization '02*, pages 53–60. IEEE Computer Society, 2002.

[13] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[14] Vinicius Mello, Luiz Velho, Paulo Roma Cavalcanti, and Claudio Silva. *A Generic Programming Approach to Multiresolution Spatial Decompositions*, volume Visualization and Mathematics III. Springer Verlag, 2002.

[15] Valerio Pascucci. Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality. In *Computer Graphics Forum (Eurographics '99)*, volume 21(3), pages 451–460. The Eurographics Association, 2002.

[16] John Plate, Michael Tirtasana, Rhadams Carmona, and Bernd Frhlich. Octreemizer: a hierarchical approach for interactive roaming through very large volumes. In *Proceedings of the symposium on Data Visualisation 2002*, pages 53–ff. Eurographics Association, 2002.

[17] Stefan Roettger and Thomas Ertl. Fast volumetric display of natural gaseous phenomena. In *Computer Graphics International*, pages 74–83. IEEE Computer Society, 2003.

[18] P. Shirley and A. A. Tuchman. Polygonal approximation to direct scalar volume rendering. In *Proceedings San Diego Workshop on Volume Visualization, Computer Graphics*, volume 24, pages 63–70, 1990.

[19] Clifford Stein, Barry Becker, and Nelson Max. Sorting and hardware assisted rendering for volume visualization. In Arie Kaufman and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, pages 83–90, 1994.

[20] Manfred Weiler, Martin Kraus, Markus Merz, and Thomas Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Procceedings of IEEE Visualization '03*, pages 333–340. IEEE, 2003.

[21] Brian Wylie, Kenneth Moreland, Lee Ann Fisk, and Patricia Crossno. Tetrahedral projection using vertex shaders. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 7–12. IEEE Press, 2002.

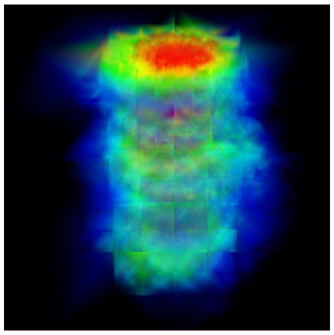
[22] Yong Zhou, Baoquan Chen, and Arie Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In *Proceedings of the 8th conference on Visualization '97*, pages 135–ff. IEEE Computer Society Press, 1997.

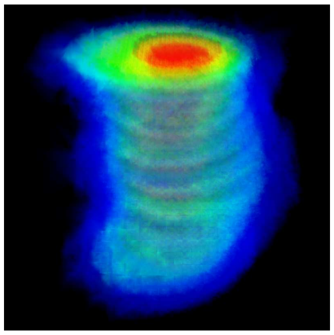
Non conforming mesh, low quality

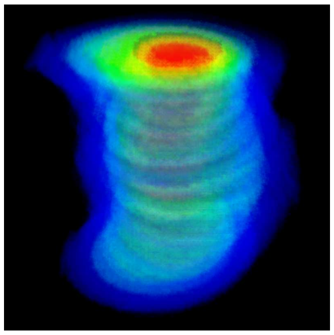
Non conforming mesh, high quality

Conforming mesh

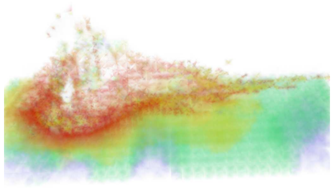
Plasma

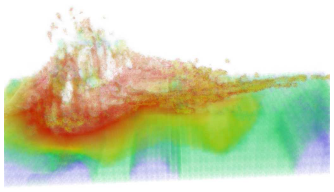

(a) 34000 tetrahedra, 5 fps

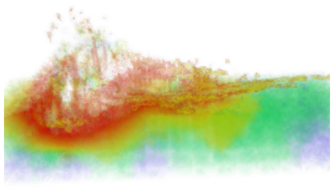

(b) 480000 tetrahedra, 0.8 fps


(c) 520000 tetrahedra, 0.8 fps

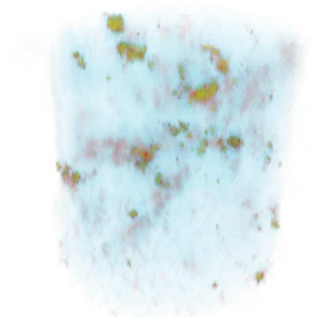
Fluid

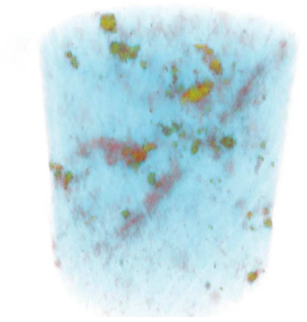

(d) 25000 tetrahedra, 7.5 fps

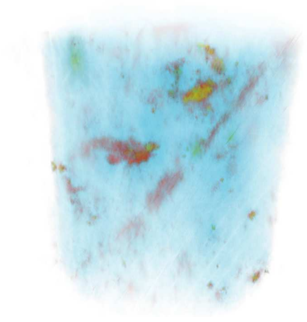

(e) 320000 tetrahedra, 0.9 fps


(f) 580000 tetrahedra, 0.6 fps

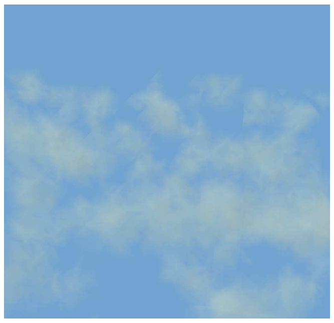
Geological

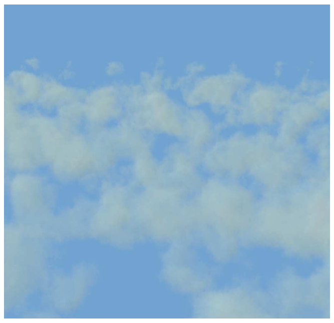

(g) 18000 tetrahedra, 5.5 fps


(h) 76000 tetrahedra, 1.7 fps


(i) 141000 tetrahedra, 1.2 fps

Cloud


(j) 4400 tetrahedra, 10 fps


(k) 260000 tetrahedra, 1 fps

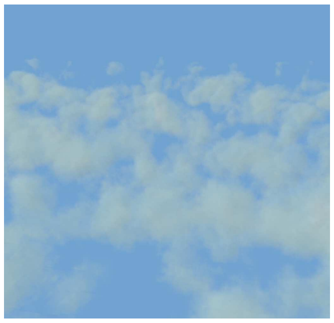

(l) 500000 tetrahedra, 0.5 fps

Figure 12: Results