J Vis (2012) 15:167–178
DOI 10.1007/s12650-011-0122-x

REGULAR PAPER

Erhan Okuyan · Uğur Güdükbay · Veysi İşler

Dynamic view-dependent visualization of unstructured tetrahedral volumetric meshes

Received: 27 June 2011 / Accepted: 8 December 2011 / Published online: 11 January 2012

Abstract Visualization of large volumetric datasets has always been an important problem. Due to the high computational requirements of volume-rendering techniques, achieving interactive rates is a real challenge. We present a selective refinement scheme that dynamically refines the mesh according to the camera parameters. This scheme automatically determines the impact of different parts of the mesh on the output image and refines the mesh accordingly, without needing any user input. The view-dependent refinement scheme uses a progressive mesh representation that is based on an edge collapse-based tetrahedral mesh simplification algorithm. We tested our view-dependent refinement framework on an existing state-of-the-art volume renderer. Thanks to low overhead dynamic view-dependent refinement, we achieve interactive frame rates for rendering common datasets at decent image resolutions.

Keywords Volume visualization · Direct volume rendering · View-dependent refinement · Progressive meshes · Unstructured tetrahedral meshes

1 Introduction

Visualization of large volumetric datasets is an important and challenging area. We focus on the view-dependent refinement of unstructured tetrahedral meshes, widely used in computational fluid dynamics. A representation to store the volume data that allows progressive refinement is crucial for this purpose. A good decimation algorithm is an important factor in constructing the levels of detail (LODs) of the original mesh to obtain a progressive representation. With a progressive mesh representation, the mesh should be refined during runtime in a view-dependent fashion.

Electronic supplementary material The online version of this article (doi:10.1007/s12650-011-0122-x) contains supplementary material, which is available to authorized users.

E. Okuyan · U. Güdükbay (✉)
Department of Computer Engineering, Bilkent University, Bilkent, 06800 Ankara, Turkey
E-mail: gudukbay@cs.bilkent.edu.tr
Tel.: +90-312-2901386
Fax: +90-312-2664047

E. Okuyan
E-mail: okuyan@cs.bilkent.edu.tr

V. İşler
Department of Computer Engineering, Middle East Technical University (METU), 06531 Ankara, Turkey
E-mail: isler@ceng.metu.edu.tr
Tel.: +90-312-2105591
Fax: +90-312-2105544

We propose a framework for dynamic view-dependent visualization of unstructured volumetric meshes. The framework uses a progressive representation of the volumetric data that supports view-dependent refinement. The progressive mesh representation (PMR) will be based on the representation presented in Cignoni et al. (2004), with a few key differences. We propose an algorithm that dynamically refines the progressive volumetric data in a view-dependent fashion, without requiring user input. Since the volume data can be highly transparent, a simple view test based on a screen space error threshold will not work. To accurately determine the importance of different regions of the volume data, the whole volume should be rendered. This should be done quickly; we propose a heuristic algorithm that performs a fast simplified rendering of the volumetric mesh. In this way, we can roughly calculate the importance of the different parts of the mesh for the final image with a small computational overhead.

Two notable studies on selective refinement of tetrahedral meshes are (Cignoni et al., 2004) and (Callahan et al., 2006). Our work differs from these in one key aspect; regions of the mesh are refined automatically according to their expected impact on the rendered image. We estimate the importance of different regions of the mesh according to the camera parameters, transfer functions etc., and refine the regions with higher importance while coarsening other regions. Cignoni et al. (2004) use user input to determine the regions to refine. Users specify certain spatial regions or field values of the mesh and refinements are performed according to this input. Callahan et al. (2006) propose several heuristics to determine the resampling of the faces. Most of these heuristics do not consider dynamic properties such as camera parameters; thus they are static resampling methods. Only view-aligned resampling method uses camera parameters, but in a limited way. Our method automatically refines the mesh regions occluded by transparent regions while coarsening the transparent regions. Similarly, it refines opaque mesh regions while coarsening the occluded regions by this opaque region. Our method can be used with any volume renderer that use irregular tetrahedral meshes.

2 Related work

Mesh simplification is a well-studied area. There are various types of mesh representations and simplification algorithms proposed for them. Many triangular mesh simplification algorithms could be used as base for tetrahedral mesh simplification algorithms. Hoppe proposes the PMR in Hoppe (1997). This representation is efficient and well-accepted, allowing view-dependent refinement of the mesh in a progressive fashion. The error metric used in a simplification algorithm is very crucial. Garland and Heckbert propose a quadric error metric in Garland and Heckbert (1997), which is used in many simplification algorithms. Trotts et al. simplify tetrahedral meshes via repetitive collapses of the tetrahedra's edges Trotts et al. (1998). Tetfusion collapses a tetrahedron into a vertex in one step Chopra and Meyer (2002), iteratively selecting the tetrahedron that will cause minimal error to the mesh. Staadt and Gross propose dynamic tests to avoid tetrahedron flips altogether Staadt and Gross (1998). These algorithms do not support level-of-detail adjustments or view-dependent refinement.

Cignoni et al. (1997) develop a multiresolution representation for volume data, using refinement-based and decimation-based approaches. Their model supports view-dependent refinement. They select mesh regions from different detail levels and merge them, and correct inconsistencies on the connecting surfaces. Cignoni et al. (2004) propose a PMR that supports view-dependent refinement. This approach refines the mesh based on user-specified selective refinement queries whereas our approach automatically refines the mesh based on camera parameters. Du and Chiang (2010) propose an out-of-core simplification algorithm and crack-free LOD volume rendering. This approach also supports selective refinement queries. Sondershaus et al. (2005) propose a segmentation-based mesh representation of volume data, which allows view-dependent refinement using a hierarchy of pre-constructed segments.

Among various direct volume rendering techniques, Koyamada's algorithm (Koyamada, 1992), is one of the earlier influential algorithms that is more suitable for software implementations. Recently hardware-based volume renderers are more popular. Callahan et al. (2006) render the tetrahedral mesh as a set of faces. In their method, the sorting of the facets are performed with both CPU and GPU, achieving high rendering rates.

3 Proposed framework

We propose a framework for the view-dependent refinement of unstructured volumetric models. The framework supports direct volume visualization and selective refinement of different parts of the model for different viewing parameters. The framework is based on a new progressive volume-data representation that

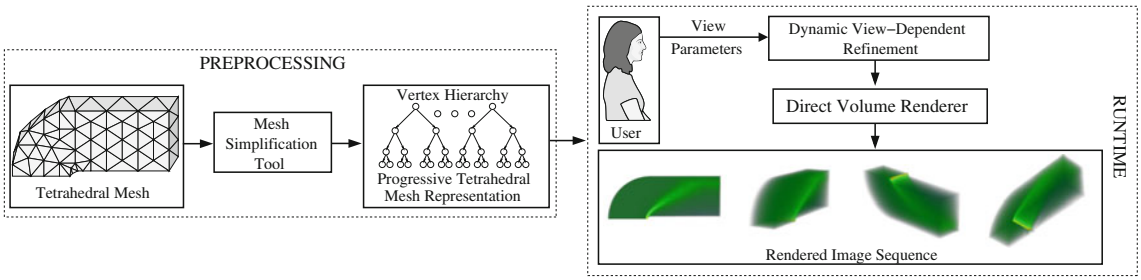

Fig. 1 The overview of the proposed framework

supports selective refinement. The detail level of the mesh can be set independently at different parts of the model depending on the viewing parameters. Figure 1 gives an overview of the proposed framework. The framework consists of the following stages:

1. *Preprocessing: construction of the progressive mesh representation* The tetrahedral mesh is the input to the framework, containing the vertices, which includes the position and scalar values, and the tetrahedra. The mesh simplification tool works on the input and constructs the PMR. It uses a decimation algorithm to obtain lower detail levels of the mesh and constructs the vertex hierarchy.
2. *Runtime: dynamic view-dependent refinement and direct volume rendering*
   - 2.1 *Dynamic view-dependent refinement using view parameters* The dynamic view-dependent refinement algorithm refines the PMR according to the viewing parameters. This algorithm estimates the impact of refining different regions of the mesh on the output image so that the important parts are represented in high detail. In this way, the simplification ratios can be much higher than using non-view-dependent detail adjustment approaches for the same target quality.
   - 2.2 *Render the refined mesh using a direct volume renderer* We modify a direct volume renderer to support direct volume visualization with the PMR.

We use a volumetric data representation that allows view-dependent volume rendering. Our representation is based on Cignoni et al. (2004) with some key differences. Similar to their work, we use an edge-collapse-based decimation algorithm to obtain lower detail levels. Repeated edge collapses construct binary vertex trees, which are the backbone of the PMR.

Figure 2 shows the two basic operations of the edge-collapse decimation algorithm. Edge collapses and vertex splits, which are the inverse of each other, are used to coarsen or refine the mesh. Figure 2 presents a simple tetrahedral mesh with eleven vertices and eight tetrahedra. The tetrahedra are *abcd, abde, abef, acdh, adei, aefj, befk*, and *bcdg*. The edge *ab* is collapsed into vertex *v*. As a result of this collapse, the tetrahedra that use both *a* and *b* vertices are collapsed as well. The tetrahedra that use one of the *a* and *b* vertices are modified to use vertex *v* instead. The resulting mesh contains five tetrahedra: *vcdh, vdei, vefj, vefk*, and *vcdg*. Splitting the vertex *v* restores the vertices *a* and *b*, obtaining the initial version of the mesh.

There are two key differences between Cignoni et al.'s representation and ours. The first difference is the active vertex mechanism, which eliminates the necessity of maintaining tetrahedral information during the refinement. The second one is related to handling possible tetrahedral flips during selective refinement.

3.1 Active vertex mechanism

In Cignoni et al. (2004), whenever a vertex splits or an edge collapses, affected tetrahedra are updated accordingly. This brings significant overhead. We avoid this using active vertex mechanism. Before the mesh simplification begins, the tetrahedra contain pointers to the vertices in the original mesh. During the simplification, pairs of vertices are collapsed into a newly created parent vertex, and tetrahedra that use one of these collapsed vertices must be modified to use the newly created parent vertex. Figure 3a shows an example vertex hierarchy. In this example, the vertices of the original mesh are all active; i.e., the mesh is in its finest state. The tetrahedron T consists of four vertices, v_0, v_1, v_2, and v_4, which are all active in this example. Assume that we change the detail level of this mesh by performing some collapses and thus obtain the mesh shown in Fig. 3 b. In this case, only the vertices v_0, v_1, v_8, and v_{11} are active. The vertices v_2 and

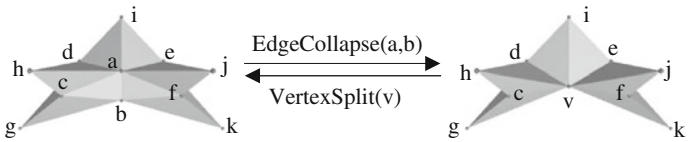

Fig. 2 Vertex split and edge collapse

v_4 of T are no longer active. We must now use a mechanism that will map v_2 to v_8 and v_4 to v_{11} because v_8 and v_{11} are the vertices that represent the inactive ones.

Our idea is to keep the references in the tetrahedron structure and find the active vertex that represents the vertex stored in the tetrahedron structure during runtime. For example, when the volume renderer processes T in Fig. 3b, it requests vertices v_0, v_1, v_2, and v_4. v_0 and v_1 are active vertices and can be used. v_2 and v_4 have to be mapped to the active vertices. The mapping is done by following the parent links, until an active vertex is found. The dashed lines in Fig. 3b show such link traversals. With a simple caching mechanism, the overhead of these traversals is reduced significantly. Whenever an active vertex of a vertex is accessed, first the validity of the cached information is checked. If the cached information is valid, then it is used. Otherwise, with the described link traversals, correct information is found. The cached information along the traversal path are also updated. Thus, a link traversal would be required only if there has been a related vertex hierarchy change which invalidate the cached information. Accordingly, active vertex mechanism does not bring any redundant link traversal overhead.

The volume renderer traverses each tetrahedron and finds its active vertices. If all active vertices of a tetrahedron are different, then the tetrahedron is active. Otherwise, the resulting geometry is not a tetrahedron. While refining or coarsening the mesh with this mechanism, simply maintaining the vertex hierarchy is sufficient; maintaining the active tetrahedra, which could take up a significant time, can be avoided. The active vertex mechanism also groups the job of finding the active tetrahedra together. If the tetrahedral information were to be updated, the list of active tetrahedra could be maintained during the refinement. However, this job would be distributed among many refinement operations. With active vertex mechanism, such list can be maintained with a single traversal of tetrahedra. Such traversal can be very efficiently parallelized with GPU whereas tetrahedral updates have to be done serially on CPU. Thus, even though active vertex mechanism increases the total work volume of finding active tetrahedra, since all the tetrahedra have to be traversed not just affected ones, due to parallelism, the process would be faster.

3.2 Progressive mesh representation of volumetric data

The two data structures defining the unstructured tetrahedral meshes are the vertex and tetrahedron structures. The vertex structure minimally contains the coordinates and the scalar value of the vertex. In order to support selective refinement, our representation adds additional fields; active vertex id, pointer to edge-collapse/vertex-split record, parent and child vertex pointers. Parent and child vertex pointers define the vertex hierarchy and active vertex id stores the cached active vertex information. Edge-collapse/vertex-split record stores the information that will be used to split a vertex or collapse an edge, such as error values and the affected vertices. The tetrahedron structure contains ids of the tetrahedron's vertices in the finest mesh. Since the number of vertices is significantly smaller than the number of tetrahedra, the memory overhead of the added fields is relatively small.

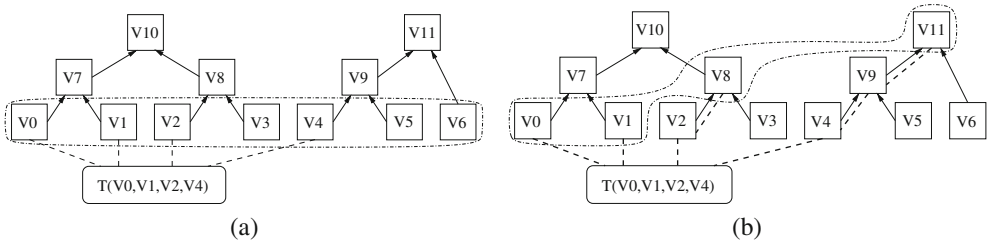

Fig. 3 Active vertex mechanism: **a** initial mesh, **b** simplified mesh

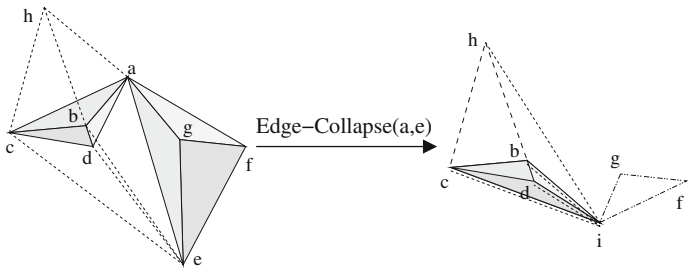

Fig. 4 Covering a volume multiple times in a tetrahedral mesh due to tetrahedron flips

We use an edge-collapse-based decimation algorithm. The decimation algorithm iteratively selects the edges and collapses them until the desired simplification level is reached. At each iteration, a prey edge is selected, collapsed and the mesh consistency is maintained. The success of the algorithm depends on the collapse order of the edges, which is based on geometric and attribute errors. The quadric error metric proposed by Garland and Heckbert (1997) is used to determine collapse errors. To ensure mesh consistency, tetrahedron flips should be handled. Collapse operations affect some tetrahedra by moving one of their vertices to the opposite side of their unaffected face, thus flipping their volume (cf. Fig. 4). The edges that would cause tetrahedron flips are not collapsed.

The boundary surface geometry is extremely important for mesh quality. Deformations of the surface produce significant visual impairments. Not allowing any surface edge to collapse eliminates surface deformations. However, it is quite restrictive and adversely affect the achievable simplification ratios. We classify surface vertices as *sharp* and *smooth*. A surface vertex v is considered sharp, if the angle between the normals of any two faces on the surface using the vertex is more than a certain threshold. We do not allow the collapse of any edge that contain a sharp vertex. This issue could also be solved by including boundary preservation into the error metric.

Another issue of mesh consistency is self-intersections, which cause similar problems like tetrahedron flips. Surface preservation eliminates most of the self-intersection cases. During our experiments, we did not observe any artifacts due to self-intersecting mesh. However, our framework does not guarantee to eliminate all the cases. In order to reduce computational complexity, we prefer not to perform extra checks to eliminate all these cases. The approach described by Staadt and Gross (1998) could be used to avoid self-intersections during decimation.

In the preprocessing stage, the decimation algorithm constructs the PMR. The PMR supports selective refinement of the mesh during runtime using vertex splits and edge collapses. The algorithm can only collapse edges whose vertices are both active and siblings in the hierarchy. Figure 3b provides examples of allowed edge-collapse and vertex-split operations. The vertices v_0, v_1, v_8 and v_{11} are active in the given example. Because no other pair of sibling vertices are active, only the edge between v_0 and v_1 can be collapsed. If that edge is collapsed, then v_7 becomes active, thus the edge between v_7 and v_8 becomes a candidate for collapse. Similarly, only the vertices v_8 and v_{11} are candidates for vertex splits, since they are the only active vertices who have children.

The vertex-split operation activates the child vertices and deactivates the input vertex. Then the vertex split and edge collapse candidate structures are maintained. The edge-collapse operation is executed in a similar manner. These two operations simply activate or deactivate vertices in the vertex hierarchy. The tetrahedra information is not updated with these operations as the direct volume renderers must check if a tetrahedron is active or inactive before processing it. Thus, updating the vertex hierarchy is sufficient to refine or coarsen the mesh selectively.

4 View-dependent refinement

The proposed PMR supports selective refinement during runtime. The regions of the mesh that have higher impact on the rendered image are automatically refined, while other regions coarsened. The detail levels of different parts of the mesh should be determined according to the viewing parameters. However, determining the important regions of the mesh is not an easy task. The volume is rendered through tetrahedra but the refinement is performed on vertex hierarchies. Thus the effect of the refinement of a vertex will be distributed through the volume of tetrahedra that use it. Furthermore, the refinement of seemingly unrelated

parts of the mesh will affect the geometries of several tetrahedra and change the effects of previous refinements. Mapping vertex hierarchies to tetrahedra is non-trivial, making selective refinement a non-trivial task. We develop a heuristic algorithm, taking into account several parameters that affect the importance of a vertex-split/edge-collapse operation; i.e., how much the output image quality changes after performing the operation. Five parameters contribute to the importance metric. The first two represent the normalized mesh error values introduced with an edge collapse. The other values represent the weight of the mesh error, affecting the output image. The formulations of the parameters will be given on the vertex split operation $v \rightarrow (p, q)$.

Color error The scalar values of vertices are used to calculate the color values using the transfer function, which is then used to calculate the color error. The color error can be defined as $|p.\text{color} - v.\text{color}|^2 + |q.\text{color} - v.\text{color}|^2$, where colors are normalized RGBA vectors.

Geometric error The geometric error can be defined as $|p.\text{position} - v.\text{position}|^2 + |q.\text{position} - v.\text{position}|^2$, where the positions are coordinates of the vertices. Since vertex coordinates does not change via the selective refinement, geometric errors can calculated in the preprocessing stage.

Light intensity The regions of a volume affect the final image in a way directly proportional to the intensity of the light reaching these regions.

Affected volume The affected volume for a vertex represents the total volume of tetrahedra that will be affected by the refinement of that vertex. The larger the volume, the bigger the affected image segment will be.

Camera distance The regions of the volume close to the camera usually have high impact on the rendered image, especially for opaque surfaces.

Ray-casting-based volume renderers send rays through each pixel. While passing through the volume, rays lose some of their intensity depending on the transparencies of the tetrahedra on the path. The effect of each tetrahedron that the ray visits are combined to calculate the pixel's intensity. The intensity of the light reaching a region directly affects how much that region can contribute to the color of the ray. Thus, calculating light intensities in each part of the volume is necessary to determine the importance of that region, which is computationally intensive. As a solution, we approximate the intensities to determine the importance of different regions.

In order to answer light intensity queries in a timely manner, we construct an octree representation of the volume data. The octree does not replace the proposed PMR, but is a low-resolution representation of the original data. The octree structure is constructed bottom-up using only the vertices. The approximate light intensity calculations are performed during runtime before rendering. In principle, the calculations are similar to any ray-casting-based volume renderer. Due to the regularity of the octree structure, the approximate light intensity calculations can be done very efficiently. The octree should be updated during refinement operations. We use flooding-based techniques to update the octree during the refinement. Whenever a refinement operation changes the vertex hierarchy, the corresponding octree cell is found. The owner vertices of the cells are updated starting from this cell and continuing to its neighbors.

4.1 Importance metric

The selective refinement algorithm must decide the vertices to split and edges to collapse in the vertex hierarchy. Since the algorithm must determine the edge-collapse/vertex-split operations that will be executed, the importance metric should be defined for edge-collapse/vertex-split records. Since the vertex hierarchies do not directly reflect the specific effects of the vertices, defining an ideal importance metric is very difficult; thus we employ a heuristic approach. We use the weighted combination of a few parameters as the importance metric. We multiply the parameters in order to combine them and use the exponents as weights. The importance metric is given in Eq. 1.

$$I(v) = \text{ColorError}(v)^{\alpha} \times \text{GeometricError}(v)^{\beta} \times \text{LightIntensity}(v)^{\gamma} \times \text{Volume}(v)^{\theta} \times \text{CameraDistance}(v)^{\sigma} \tag{1}$$

Selective refinement splits the vertices with the highest importance and collapse the edges with lowest importance. However, since refinements can only be done on active vertices in the vertex hierarchy, the refinement of a seemingly unimportant vertex can enable the refinements of more important vertices. The importance metric given in Eq. 1 is updated to take this into account by taking a weighted average of the importance of a vertex and the importances of its children (Eq. 2).

$$I_{\text{updated}}(v) = 0.50 \times I(v) + 0.25 \times I(v.\text{child0}) + 0.25 \times I(v.\text{child1}) \quad (2)$$

Since the importance metric is used during the view-dependent refinement, it must be calculated on the fly. Even though we use approximations for many parameters, the rendered images do not contain notable artifacts. The optimal values for exponent weights in the importance metric depend on the mesh characteristics and viewing parameters. We employed an experimental method to determine the weights. Color errors and geometric errors are computed only using the vertices of edge-collapse/vertex-split operations. For the collapse of an edge of a very thin tetrahedron, color errors and geometric errors can be very high. However, since the affected volume will be small, the effect of these errors would be small. The weights of geometric errors (β) should be higher when the tetrahedra are more regular, since the affected volume will be more related to geometric errors. The weight of the color errors (α) also depend on the affected volume. It should be higher for meshes with regular and uniform sized tetrahedra, since the affected volume sizes of the vertices would be closer to each other. The affected volume parameter is included in the importance metric as a support mechanism for geometric and color errors. For tetrahedral meshes with high regularity and uniformity, α and β should be higher compared to the weight of the affected volume parameter (θ). However, for more irregular meshes, θ should be higher compared to α and β.

The weight of the light intensity parameter (γ) depend on the accuracy of light intensity calculations. Light intensity calculations use a low-resolution regular grid rendering mechanism, which will introduce a blending effect on the light intensity estimations. If the opacity of the mesh is more uniform, the accuracy will be better. Otherwise, due to the blending effect, the accuracy will be worse. The γ value should be higher for more accurate estimations.

Although the affected volume parameter is very important, it does not directly reflect the affected image area. If the volume is closer to the camera, the affected image area would be larger. The camera distance parameter is used to correct this effect. The vertex camera distance is converted to a coefficient that will relate the affected volume parameter to the size of the affected image region. The weight of the camera distance parameter (σ) should be equal to θ. The blurring effects build up for distant parts of the mesh, making intensities of these parts less accurately estimated than closer parts. Thus, favoring the closer parts of the mesh results in better refinement.

The selective refinement algorithm is a heuristic algorithm that sets different regions of the volume data to the appropriate level of detail. It is called just before the rendering and refines the mesh according to the viewing parameters. The algorithm keeps track of every vertex that can be split and every edge that can be collapsed. The importance of these primitives are calculated and the mesh is updated accordingly for the desired detail level.

4.2 Required modifications for volume renderers

The proposed selective refinement framework can be used with a wide variety of direct volume renderers. Two modifications are required for this purpose. The first modification is to use the active vertex mechanism for selective refinement. The second modification is handling tetrahedron flips during selective refinement. When a tetrahedron flip occurs, the volume of the flipped tetrahedron is covered by more than one tetrahedron. That inconsistency can cause artifacts. Figure 4 presents a simple example to demonstrate this point. The collapse of the edge (a, e) causes tetrahedron T_{abcd} to flip and cover some volume below its base face. The tetrahedra T_{abch} and T_{bcde} are also affected with the edge collapse and they are stretched to cover the volume of T_{abcd}.

Flipped tetrahedra cause artifacts because of over-rendering of the flipped volume and mis-representations. Flips can be handled in different ways, depending on the volume renderer and rendered datasets. One approach is to allow tetrahedron flips. It is suitable for volume renderers that process the mesh as a set of faces, such as HAVS (Callahan et al., 2005). HAVS extracts the faces from the tetrahedra and sorts them in visibility order. It determines the contribution of a face on a pixel according to the distance between the face and the next face; thus, the flipped tetrahedra cannot cause over-rendering. Another approach is subtracting the contributions of the flipped tetrahedra. Since the flipped tetrahedra can be considered to have negative volumes, subtracting their contributions prevents over-rendering. The mis-representations caused by tetrahedron flips do not cause notable artifacts for most datasets. If the artifacts are noticeable, tetrahedron flips should be eliminated, e.g., using the approaches described by Cignoni et al. (2004) and El-sana and Varshney (1999). However, it introduces extra computational and memory overhead for selective refinement, since directed acyclic graphs have to be constructed and maintained.

Table 1 The PSNR values and rendering times

Dataset	Bucky				Comb				Aorta				Sf2				Sf1			
LOD	F	S	N	C	F	S	N	C	F	S	N	C	F	S	N	C	F	S	N	C
Refinement ratio	100.0	12.3	47.1	0.5	100.0	10.2	70.8	3.0	100.0	19.4	38.0	2.0	100.0	9.6	83.6	0.9	100.0	11.2	80.0	6.47
No. tetrahedra	1,250.2	153.9	588.4	6.5	215.0	21.9	152.2	6.4	1,386.9	255.1	526.5	28.0	2,067.7	199.3	1,728.5	19.3	13,980.1	1571.4	11,184.2	903.9
PSNR value	N.A.	32.43	32.37	10.41	N.A.	37.46	37.36	16.57	N.A.	38.19	38.16	18.39	N.A.	40.94	40.39	14.71	N.A.	45.02	44.99	28.40
Rendering times	718	406	609	203	125	64	108	46	593	327	405	187	936	344	842	266	18,359	3,664	7,891	1,985

No. tetrahedra are given in thousands and rendering times are given in milliseconds
F finest, *S* selective refinement, *N* non-selective, *C* coarsest

5 Results

We analyze the performance of the proposed framework on different datasets. We use HAVS for volume rendering to measure rendering times for selective and non-selective refinement. The *k-buffer* size is set to 6. We also use a software-based volume renderer, which is slow but generates high quality images, with the proposed framework to compare the image quality of selective and non-selective refinement schemes. Transfer functions are selected to highlight the important features of the volumetric datasets. Some sophisticated techniques, such as visibility-driven transfer functions (Correa and Ma 2011) could also be used for this purpose. We use a wide range of camera parameters, in order to highlight the dynamic view-dependent refinement property. We compare the selective refinement scheme with non-selective refinement scheme. Non-selective version refines the mesh in reverse decimation order.

With the datasets used in experiments, HAVS generated some artifacts due to insufficient *k-buffer* size, particularly for simplified meshes where tetrahedra become more irregular. Since higher *k-buffer* sizes are not supported with the current implementation of HAVS, we were not able to compare our simplification method against HAVS's LOD methods (Callahan et al., 2006). We also were not able to compare our approach to selective refinement queries (Cignoni et al. 2004), since the implementation is not publicly available.

In the first group of tests, we set a certain level of image quality as the target quality. Then the mesh is selectively and non-selectively refined until the target quality is reached. The rendering times indicate the success of the refinement schemes. The quality of the mesh is measured in terms of *Peak-Signal-To-Noise Ratio* (PSNR) values, a widely accepted logarithmic scale for image comparisons. PSNR values are not perfectly accurate; however, they are a good tool for general evaluations. For selective refinement, the average per-frame overhead is added to the rendering times. For comparison, the finest and coarsest meshes are also included in the tests.

The tests are performed on a PC with an *nVidia 8800GT* graphics card and an *Intel Core 2 Duo 2.66GHz* CPU. The resolution is 512×512. The preprocessing step takes less than 30 min for the largest dataset, Sf1 with about 14 million tetrahedra. Table 1 show that selective refinement gives significant improvement over non-selective refinement. Depending on the dataset, up to 60% speed-ups are observed. Selective refinement also significantly reduces the number of rendered tetrahedra (up to 88%), which greatly reduces the memory requirements on the GPU.

The selective refinement is more successful for datasets where the parts of the mesh that define the features in the output image are spatially localized. Selective refinement can find and refine such localized regions where non-selective refinement cannot focus on a certain part of the mesh. Due to the irregular topology of the Aorta dataset, the octree representation is not as successful as in other datasets. Accordingly, selective refinement performance is affected. Using a higher octree size could produce better results for this dataset.

In the second group of tests, we compare the quality of images that selective and non-selective refinement generate for a fixed budget of rendering time. The finest and coarsest meshes are also rendered to give a comparison. Figure 10 shows frame rendering times and PSNR values for the visualization of Bucky and Sf2 datasets. The PSNR values for the frames of the animations show that selective refinement gives much better image quality than non-selective refinement for the approximately the same rendering times. The resolution of the animations are $1,024 \times 1,024$. Figures 5, 6, 7, 8, and 9 show rendered images from the visualizations. We observe that significant quality improvements are obtained with selective refinement. For some datasets, the quality of the selectively refined mesh reaches just below the finest mesh, for a fraction of the rendering time.

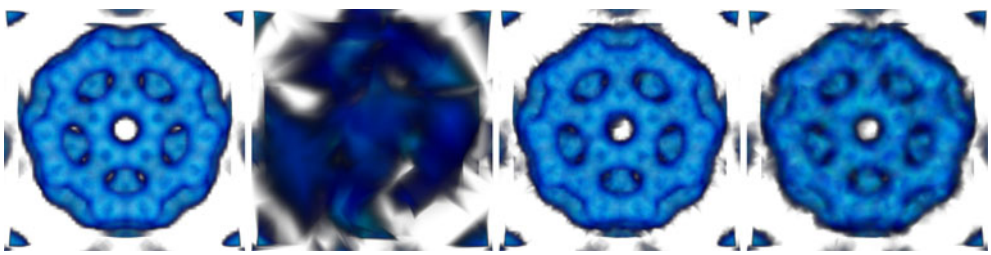

Fig. 5 Rendered images of the Bucky dataset. *Left-to-right* finest, coarsest, selectively-refined, non-selectively refined (Animation 1)

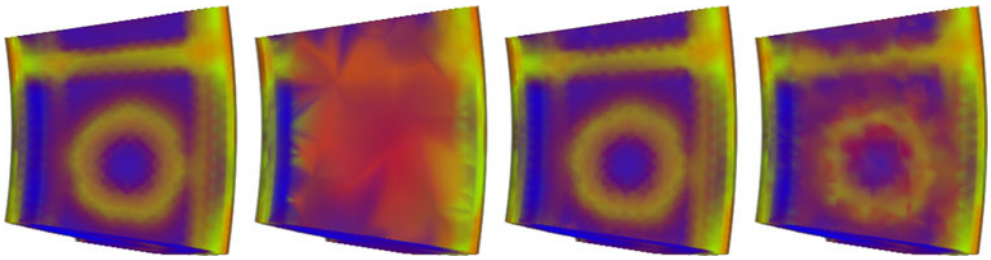

Fig. 6 Rendered images of the Comb dataset. *Left-to-right* finest, coarsest, selectively-refined, non-selectively refined (Animation 2)

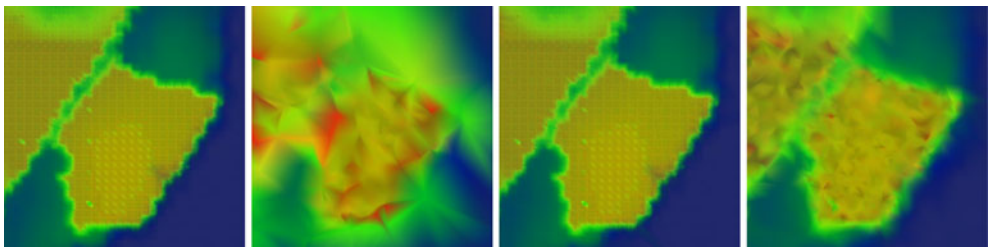

Fig. 7 Rendered images of the Sf2 dataset. *Left-to-right* finest, coarsest, selectively-refined, non-selectively refined (Animation 3)

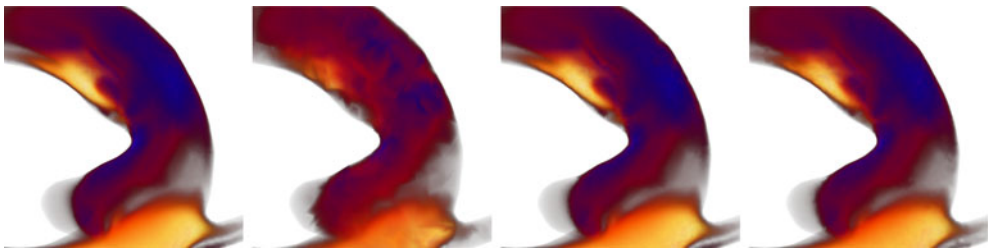

Fig. 8 Rendered images of the Aorta dataset. *Left-to-right* finest, coarsest, selectively-refined, non-selectively refined

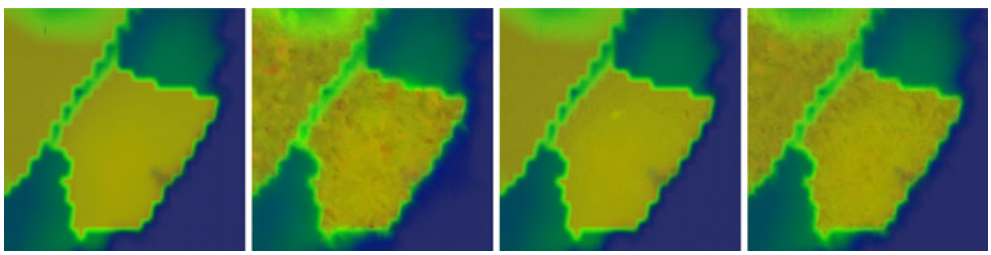

Fig. 9 Rendered images of the Sf1 dataset. *Left-to-right* finest, coarsest, selectively-refined, non-selectively refined

6 Conclusions

We propose a low overhead dynamic selective refinement scheme for unstructured tetrahedral meshes. We use a progressive mesh representation that support selective refinement. In addition to static selective refinement queries, such as spatial refinements or field value-based refinements, we incorporate

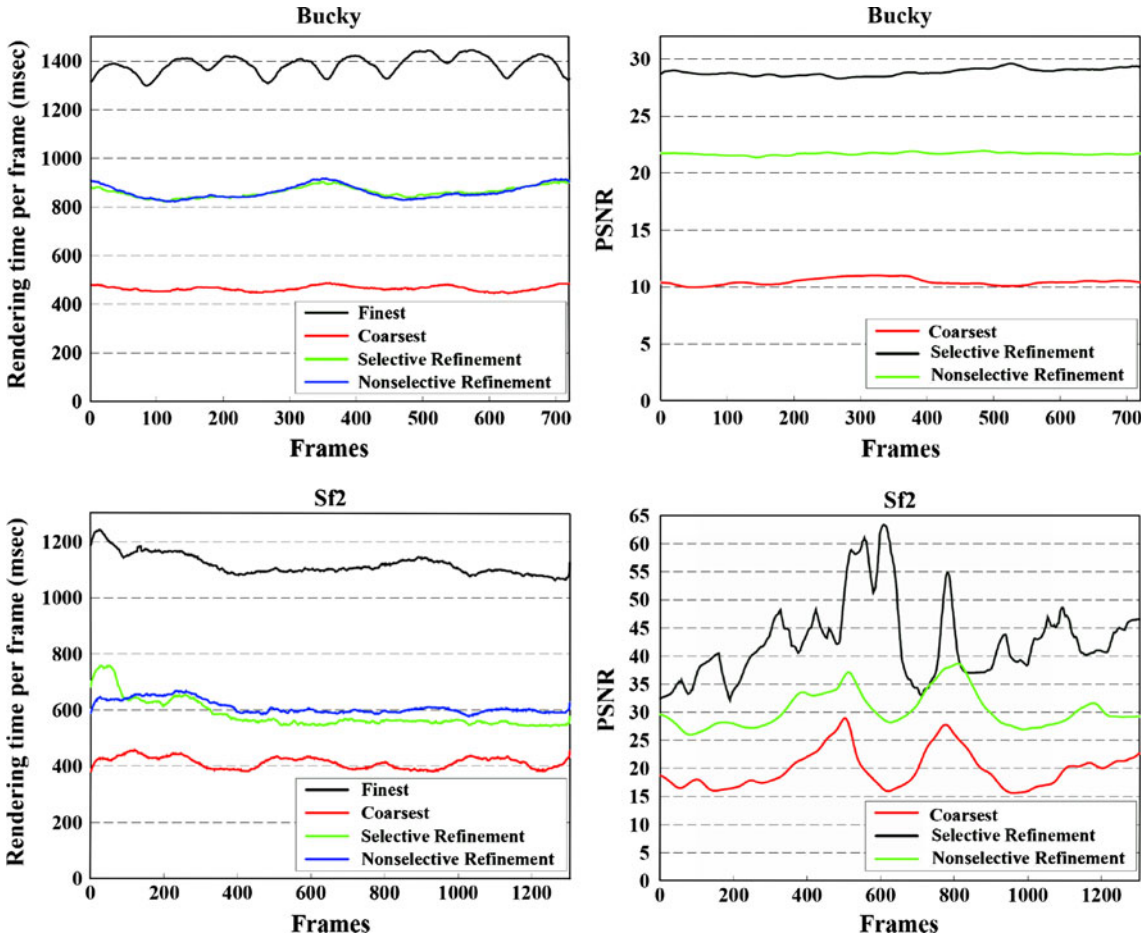

Fig. 10 Rendering times and PSNR values for Bucky Ball and Sf2 datasets

view-dependent dynamic refinement capabilities. We propose a heuristic selective refinement algorithm, which automatically refines the mesh according to the camera parameters. We propose an importance metric that estimates the effect of the vertices on the rendered image.

We test the proposed scheme on several datasets with different characteristics and viewing parameters. The results are quite satisfactory. We achieve up to 60% speed-ups and up to 88% reduction in GPU memory requirements for the same image quality with selective refinement as compared to the non-selective refinement. We also show that different direct volume renderers can be modified to work with the our framework with little extra effort. As a future work, we plan to extend our implementation on much larger datasets (containing hundred millions of tetrahedra).

Acknowledgments The authors are grateful to Koji Koyamada for the volumetric datasets. The Comb dataset is courtesy of NASA. The Sf1 and Sf2 datasets are courtesy of David R. O'Hallaron and Jonathan R. Shewchuk (CMU). We are grateful to Rana Nelson for proofreading.

References

Callahan SP, Bavoil L, Pascucci V, Silva CT (2006) Progressive volume rendering of large unstructured grids. IEEE Trans Vis Comp Graph 12(5):1307–1314

Callahan SP, Ikits M, Comba JLD, Silva CT (2005) Hardware-assisted visibility sorting for unstructured volume rendering. IEEE Trans Vis Comp Graph 11(3):285–295

Chopra P, Meyer J (2002) Tetfusion: an algorithm for rapid tetrahedral mesh simplification. In: Proceedings of IEEE Visualization, pp 133–140

Cignoni P, De Floriani L, Magillo P, Puppo E et al (2004) Selective refinement queries for volume visualization of unstructured tetrahedral meshes. IEEE Trans Vis Comp Graph 10(1):29–45
Cignoni P, Montani C, Puppo E, Scopigno R (1997) Multiresolution representation and visualization of volume data. IEEE Trans Vis Comp Graph 3(4):352–369
Correa C, Ma KL (2011) Visibility histograms and visibility-driven transfer function. IEEE Trans Vis Comp Graph 17(2):192–204
Du Z, Chiang YJ (2010) Out-of-core simplification and crack-free LOD volume rendering for irregular grids. Comp Graph Forum 29:873–882
El-sana J, Varshney A (1999) Generalized view-dependent simplification. Comp Graph Forum 18:83–94
Garland M, Heckbert P (1997) Surface simplification using quadric error metrics. In: Proc. SIGGRAPH, pp 209–216
Hoppe H (1997) View-dependent refinement of progressive meshes. In: Proc. SIGGRAPH, pp 189–198
Koyamada K (1992) Fast traversal of irregular volumes. In: Visual computing—integrating computer graphics with computer vision, Springer, Berlin, pp 295–312
Sondershaus R, Straßer W (2005) View-dependent tetrahedral meshing and rendering. In: Proceedings of the 3rd International Conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp 23–30
Staadt O, Gross M (1998) Progressive tetrahedralizations. In: Proc. IEEE Visualization, pp 397–402
Trotts I, Hamann B, Joy K, Wiley D (1998) Simplification of tetrahedral meshes. In: Proc. IEEE Visualization, pp 287–295