Multi-threaded parallel projection tetrahedral algorithm for unstructured volume rendering

Liang Fan, Cheng Chen, Sirui Zhao, Xiaorong Zhang, Yadong Wu & Fang Wang

Journal of Visualization

ISSN 1343-8875

J Vis
DOI 10.1007/s12650-020-00701-7

Springer

J Vis
https://doi.org/10.1007/s12650-020-00701-7

REGULAR PAPER

Liang Fan · Cheng Chen · Sirui Zhao · Xiaorong Zhang · Yadong Wu · Fang Wang

Multi-threaded parallel projection tetrahedral algorithm for unstructured volume rendering

Received: 1 July 2020 / Revised: 11 August 2020 / Accepted: 27 August 2020

Abstract Volume rendering methods have been extensively studied in recent years due to their effectiveness and expressiveness for unstructured grid data visualization. Although existing volume rendering methods have demonstrated great success, we observe that these methods have extensive interpolation and integral operations, which may adversely affect their efficiency and further prevent them being applied in interactive visualization. To boost efficiency and achieve interactive rendering rates, we propose a novel multi-threaded parallel projection tetrahedral algorithm based on multi-core architecture. By analyzing the parallelism of volume rendering methods, we find that the visibility sorting and classification/decomposition of projection polygons are the most time-consuming parts. To reduce the execution time of these two parts, we design corresponding parallel methods. In this manner, our method can dramatically improve efficiency and further enable user interactions for progressive unstructured grid analysis. The visibility sorting part includes partial sorting and global sorting: In partial sorting, we partition disordered tetrahedral depth array and obtain several loosely coupled subarrays, and in global sorting, we sort each subarray with multi-threads technique. In the classification/decomposition of projection polygons part, we normalize tetrahedral projection to ensure arbitrary tetrahedral produces the same number of triangles, and then store the produced vertex data into vertex array with offset computation that ensures correct order for the multi-threads runtime. The experimental results show that the proposed multi-threaded projection tetrahedral algorithm can achieve a speedup of 3.4X on a 20 cores CPU and outperforms the fastest VTK implementation at a speedup of 2.5X, which verifies the efficiency of our algorithm.

Keywords Unstructured grid · Volume rendering · Multi-threaded parallel · Projection tetrahedral

1 Introduction

Unstructured grid, usually expressed as unstructured volumetric data, is widely used in medical imaging, computational fluid dynamics (CFD), meteorology and other scientific computing fields (Kaufman and Mueller 2005). With the development of scientific computing and simulation, the scale of various

L. Fan · S. Zhao · X. Zhang
Southwest University of Science and Technology, Mianyang, China
E-mail: andyfanplus@gmail.com

C. Chen (✉) · F. Wang
China Aerodynamics Research and Development Center, Mianyang, China
E-mail: chengchen@cardc.cn

Y. Wu
Sichuan University of Science and Engineering, Zigong, China
E-mail: wyd028@163.com

Published online: 19 September 2020

unstructured volume data is growing rapidly, and how to gain an insight into these data is the major goal of research. Volume rendering methods regard volumetric data as a semi-transparent material that can absorb and emit ray (Max 1995) and directly produces 2D screen images from 3D data. With strong spatial expressiveness, users can utilize volume rendering to see through the data, explore internal structure of the data and gain new insights (Min et al. 2000; Kaufman 1996).

Extensively used volume rendering algorithms for unstructured grids can be classified as image space and object space (Kaufman and Mueller 2005). Ray casting algorithm is the most representative volume rendering technique based on image space implementation. It accumulates the contribution of three-dimensional scalar field along the ray through optical integration as final pixel value on the screen (Max 1995). Ray casting for unstructured grid performs adaptive sampling based on the size of tetrahedral cells and then calculates the intersection point of the ray and tetrahedral surface to determine the next sampling point (Garrity and Michael 1990). The connectivity of tetrahedral cell is used to calculate next cell that the ray enters after leaving current tetrahedral cell. Therefore, the shortcomings of ray casting algorithm for unstructured grid are extensive calculations and storage overheads. To enhance the algorithm efficiency, Weiler et al. (2003, 2004) and Mueller et al. (2007) optimize the ray casting algorithm through hardware acceleration. Moreover, Miranda and Filho (2011) leverages orthogonal integration to implement ray casting algorithm for hexahedral grid rendering. The algorithms based on object space decompose volume data into a set of basis primitives, which are individually projected to screen and assembled into an image. Cell projection method (Max et al. 1990; Shirley and Tuchman 1990; Wilhelms and Van Gelder 1991; Williams 1992) is the most representative volume rendering technique based on object space, which maps grid cells to screen one by one and then calculates the contribution of grid cells to screen pixels through integral operations to produce final image. Hence, the algorithm based on cell projection needs correct visibility order. The NNS algorithm proposed by Newell et al. (1998) et al. calculates visibility order based on primitive depth, which is an approximate sorting method with simple sorting pass. Stein et al. (1995) and Williams (1992) proposed accurate visibility sorting algorithms which sort cells with the connectivity of cells strictly. The projection tetrahedral (PT) algorithm proposed by Shirley and Tuchman (1990) is a popular cell projection algorithm. Build on Shirley's work, Wylie et al. (2002), Marroquim et al. (2008) and Maximo et al. (2010) proposed hardware-accelerated PT algorithms to enhance performance. The PT algorithm first performs visibility sorting on all tetrahedral cells, and then sorted tetrahedral cells are mapped to image plane and decomposed into a series of triangles; finally, the triangle vertex data are loaded into GPU memory for rasterization and assembled into final result. Thus, the PT algorithm could produce approximate volume rendering results with a series of semi-transparent triangles. For its efficiency, the PT algorithm is one of the most significant object space-based rendering technique for unstructured grid.

However, the implemented PT algorithms only use sequential computation on CPU and thus cannot provide smooth interactive experience for users without applying additional high-performance visualization techniques. As the trends of mainstream computer hardware are dominated by multi-core processors, visualization algorithm should effectively leverage parallel hardware to improve performance.

In this paper, we propose a novel multi-threaded parallel PT algorithm to accelerate both the visibility sorting processing and classification/decomposition of projection polygon processing, which dramatically improves the efficiency of PT algorithm and enables user interactions for unstructured grid analysis. Our main contributions are as follows:

1. We propose multi-threaded parallel sorting algorithm to improve visibility sorting efficiency. First, we calculate the depth of each tetrahedral cell and stored it in a depth array; then, the disordered depth array is divided into several subarrays with no intersection of numeric intervals. Therefore, the loosely coupled subarrays can perform parallel sorting in the multi-thread runtime.
2. We develop a normalized classification/decomposition method of projection polygon to ensure that decomposing arbitrary tetrahedral projection produces the same number of triangles. Given the viewpoint, there are four cases of tetrahedral projections. After the normalization, all tetrahedral projections are decomposed into four triangles. Therefore, for tetrahedral projections with less than four triangles after decomposition, we need to construct extra vertex data, which are used to fill the vertex array and do not affect final rendering results.
3. We propose a multi-threaded parallel method for the classification/decomposition of projection polygons. After normalization in contribution (2), all tetrahedral projections produce the same number of triangles. Therefore, we can store produced vertex data into vertex array with the loop iteration factor by calculating offset. Our approach ensures that vertex array accessing is thread-safe.

4. We provide a detailed comparison of our algorithm with implemented algorithms in VTK and calculate the speedup of our algorithm relative to sequential PT algorithm. Moreover, to further investigate performance, we also provide the breakdown of execution time experiment and scalability analysis. Compared with existing algorithms, our method shifts the visibility sorting and projection polygon classification/decomposition burden to multiple cores of CPU, which provides seamless interoperability to ensure high performance.

The remainder of this paper is organized as follows: We summarize related work in Sect. 2. In Sect. 3, we provide a detailed description of our algorithm. Section 4 summarizes our experiments and results. In Sect. 5, we provide final remarks and optimize directions for future work.

2 Related work

Volume visualization for unstructured grid is a focused research field in scientific visualization, and vast research has been done on the design (Garrity and Michael 1990) and optimization (Weiler et al. 2003) of unstructured grid rendering algorithm. In this section, we limit our coverage to the most related work on cell projection algorithms.

The PT algorithm proposed by Shirley and Tuchman (1990) is a popular volume rendering technique based on cell projection. They first performed visibility sorting for each tetrahedral cell, then projected the tetrahedral onto image plane based on visibility order and finally decomposed the projection polygons into a series of triangles. The PT algorithm leverages hardware rasterization instead of ray casting, which not only makes full use of hardware, but also performs more efficiently. Moreover, Shirley also proposed a method for converting non-tetrahedral grid to tetrahedral grid, so that the algorithm can also apply in a different unstructured grid volume rendering. The extensive floating calculations involved in visibility sorting and classification/decomposition of projection polygons are main performance bottlenecks of PT algorithm. Therefore, researchers build on Shirley's work and proposed more efficient PT algorithm for unstructured grid volume rendering. Wylie et al. (2002) leveraged vertex shaders and proposed an optimized PT algorithm. Marroquim et al. (2008) proposed GPU-based approach which not only dramatically reduced the CPU/GPU data transfer overheads, but also ensured the interactive frame rate. Maximo et al. (2010) proposed a hardware-assisted algorithm which calculated tetrahedral projection with vertex shaders and fragment shaders that makes full use of graphics hardware. Moreover, they also implemented a CUDA-based visibility sorting algorithm to perform more efficiently.

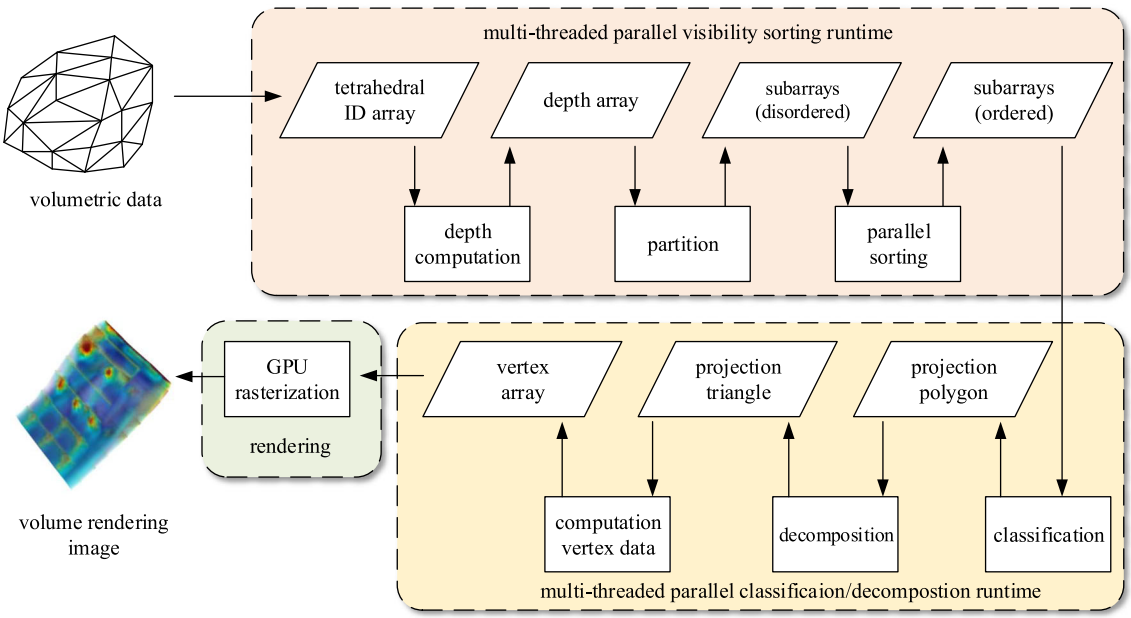

Fig. 1 Overview of the multi-threaded parallel projection tetrahedral algorithm

The hardware-assisted visibility sorting (HAVS) algorithm proposed by Callahan et al. (2005) is a significant volume rendering algorithm for unstructured grid, which combines the advantages of technologies based on image space and object space. They performed a partial sort of the cells in preparation for rasterization in object space and incrementally sorted the fragment stream with *kbuffer* approach. Their method not only simplified the CPU-based processing and shifted much of the sorting burden to the GPU, but also performed more efficiently.

The most similar to our work is Shirley's work, but the sequential algorithm cannot directly parallelize due to data dependencies. We build on their work and further propose multi-threaded parallel visibility sorting and classification/decomposition of projection polygons methods. The proposed parallel algorithm achieves significant speedup values and provides interactive performance of volume rendering.

3 Algorithm

As shown in Fig. 1, the parallel projection tetrahedral algorithm proposed in this paper first implements multi-threaded parallel visibility sorting processing based on tetrahedral depth, and then, the multi-thread approach is used to project sorted tetrahedral cells onto image plane and decompose tetrahedral projection into triangles; finally, we load vertex data produced by the classification/decomposition of tetrahedral projection into GPU memory and leverage graphics hardware to rasterize the vertex and produce final image.

3.1 Parallel visibility sorting

Given a viewpoint, the PT algorithm decomposes tetrahedral projection on image plane into triangle and approximates rendering result with vast semi-transparent triangles. Hence, an incremental visibility order for tetrahedral cells is necessary. A visibility order of a set of objects, from a given viewpoint, is a total order on the objects such that if *object a* obstructs *object b*, then *b* precedes *a* in the ordering, and can be computed with imprecise approaches and precise approaches. Imprecise sorting is simple and convenient, provides approximate sorting results and can be computed with conventional sorting methods; precise sorting is strictly based on the connectivity among cells, with high complexity and low sorting efficiency. In this paper, we build on the NNS (Newell et al. 1998) algorithm and propose a parallel visibility sorting method based on the tetrahedral depth. The tetrahedral depth is defined as dot product of viewpoint vector and tetrahedral centroid. For instance, if the viewpoint vector is *(a1, a2, a3)* and the centroid of a tetrahedral is *(b1, b2, b3)*, then the tetrahedral depth is $\sum_{i=1}^{3} a_i b_i$. And the incremental depth order for tetrahedral cells can be regarded as visibility order. To calculate the visibility order of whole tetrahedral grid in parallel, partial sorting and global sorting are necessary in our procedure.

As shown in Algorithm 1, we propose a partial sorting algorithm. The procedure obtains an array by calculating the depth of each tetrahedral; then, the depth array is partitioned into several subarrays with no intersection of numeric intervals. Nevertheless, random pivot may lead to subarrays load imbalance. To address thread waits, we have to ensure that each subarray has approximately the same number of elements. Therefore, we give a threshold *MAX* to limit the length of subarrays, and the *MAX* should not be too large.

The subarrays produced by partial sorting are disordered, so we need to perform ascending sort on each subarray, which is called global sorting. As partial sorting provided loosely coupled subarrays, the sorting on subarrays is thread-safe and different thread-performed subarrays sorting without influencing each other. The depth-based visibility order is evidently a numerical order, and quick sorting is an efficient algorithm for calculating numerical order. Here, we perform quick sorting on each subarray with multi-thread.

Algorithm 1 Partial sorting

Input: A tetrahedral grid data
1: Calculate tetrahedral depth array;
2: Partition one array into two, the subarrays have no intersection with each other, and partition with following steps:
(a) Pick a random element in the original array as *pivot*;
(b) Traverse the array from two sides to the middle, and put the element less than *pivot* to the left, meanwhile, put the element larger than *pivot* to the right;
(c) Partition the array into two subarrays at *pivot* position;
3: Repeat Step 2 until the length of each subarray less than *MAX*;
Output: A series of subarrays without intersection each other.

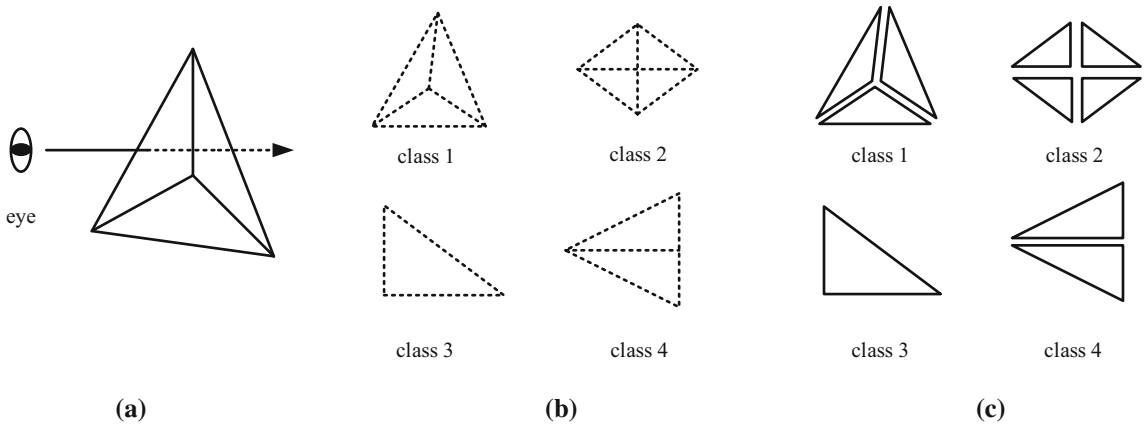

Fig. 2 Tetrahedral projection algorithm: **a** projection, **b** classification of projection polygon and **c** decomposition of projection polygon

3.2 Parallel classification and decomposition of projection polygon

The fundamental thought of PT algorithm is to decompose tetrahedral projection on the image plane into triangles which involves extensive vector, distance and interpolation calculations. Therefore, these floating computation overheads are main bottleneck of the PT algorithm. In this section, we implement a multi-threaded parallel method for classification/decomposition of projection polygon. First, to facilitate the offset computation, we normalize tetrahedral projection so that arbitrary projection produced the same number of triangles; then, we store the vertex data produced by decomposition into vertex array by calculating offset. Our approach ensures correct classification/decomposition of projection polygon in the multi-threads runtime.

3.2.1 Normalization

The classification/decomposition of projection polygons is based on visibility order. Therefore, visibility order is the sequence of classification/decomposition processing for tetrahedral projections. To perform algorithm in the multi-threads runtime correctly, we first normalize tetrahedral projection. Given a viewpoint (Fig. 2a), there are four cases of tetrahedral projections on the image plane (Shirley and Tuchman 1990) (Fig. 2b); then, tetrahedral projections are decomposed into triangles (Fig. 2c). We can see that the four projections produce one, two, three and four triangles, respectively. As shown in Fig. 2b, compared with *class* 1 and *class* 2, the relative position of vertices does not change in *class* 3 and *class* 4. Therefore, *class* 3 and *class* 4 can be regarded as special cases of *class* 1 and *class* 2, and the classification of tetrahedral cells is essentially the judgment of *class* 1 and *class* 2. We can see in Fig. 2b that the *class* 1 is triangle and the *class* 2 is quadrangle which exists internal intersection. To classify and decompose

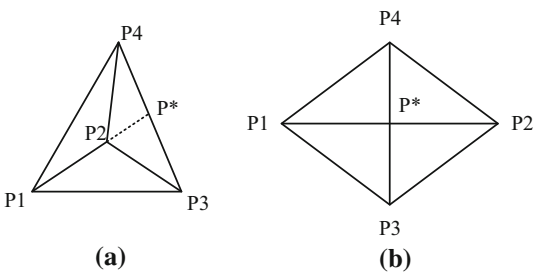

Fig. 3 Projection polygon: **a** class 1 and **b** class 2

projection polygons, the coordinate, color and opacity of diagonal intersection in *class* 2 are necessary. In this paper, we build on David's work (Antonio 1992) and propose a simple and efficient algorithm for determining whether the projection polygon is *class* 1 or *class* 2. And when projection polygon is *class* 2, our approach can solve the coordinate, color and opacity of diagonal intersection.

To develop our algorithm, it is convenient to use vector representation. Consider that point P^* is the intersection of vector $\overrightarrow{\mathbf{P_1P_2}}$ and $\overrightarrow{\mathbf{P_3P_4}}$, as shown in Fig. 3. Then, we can derive the following linear combinations, where α and β both are real numbers:

$$P^* = P_1 + \alpha\overrightarrow{P_1P_2} \tag{1}$$

$$P^* = P_3 + \beta\overrightarrow{P_3P_4} \tag{2}$$

Then, subtracting Eqs. 1 and 2 yields:

$$\overrightarrow{0} = \overrightarrow{P_3P_1} + \alpha\overrightarrow{P_1P_2} + \beta\overrightarrow{P_4P_3} \tag{3}$$

To make the equations easier to read, we gave names to some intermediate values as follows:

$$A = \overrightarrow{P_1P_2}$$
$$B = \overrightarrow{P_4P_3}$$
$$C = \overrightarrow{P_3P_1}$$

And then, we constructed linear equations with Eq. 3 as follows, where Ax and Ay represent the x component and y component of vector A, respectively. And so are vectors B and C:

$$\begin{cases} Cx + \alpha Ax + \beta Bx = 0 \\ Cy + \alpha Ay + \beta By = 0 \end{cases} \tag{4}$$

Finally, the solution of Eq. 4 for α and β is now:

$$\alpha = \frac{ByCx - BxCy}{AyBx - AxBy}$$
$$\beta = \frac{AxCy - AyCx}{AyBx - AxBy}$$

The projection polygon is *class* 1 (Fig. 2b) when α and β are outside of [0, 1]. On the contrary, if α and β are in the interval [0, 1], the projection polygon is *class* 2 (Fig. 2b). Otherwise, we can solve the coordinate, color and opacity of intersection point P^* by linear interpolation.

As shown in Fig. 2b, *class* 1 and *class* 2 are decomposed into triangles by using internal intersection point as the common vertex of triangles, which produce three and four triangles, respectively (Fig. 2c). To facilitate the implementation of parallel classification/decomposition of projection polygon algorithm, we normalize the *class* 1 and *class* 2 so that both of two are decomposed into four triangles.

3.2.2 Offset computation

In OpenGL, vertex buffer object (*VBO*) array, index buffer object (*IBO*) array and vertex array object (*VAO*) array are collectively called the vertex array, and the relation among them is shown in Fig. 4. The vertex array stores the index and attribute of vertex to ensure the accessing and rendering efficiently. Each triangle vertex has attributes of *x*, *y*, *z*, *color*, *alpha* and *depth*. Therefore, the offset is six times the index value when querying vertex attributes in *VBO* with the index value in *IBO*. The advantage of using *IBO* is that only when there are multiple triangles with common vertices, it just stores the attribute of common vertices in *VBO* once, which saves space efficiently.

The sequential PT algorithm decomposes tetrahedral projection based on visibility order. Once finished the decomposition of one tetrahedral, produced vertex data will be stored into vertex array, until all tetrahedral are processed. And then, the vertices are once sent to GPU memory for rasterization and assembled final image. Therefore, visibility order is the sequence of vertex data stored into vertex array. In the single thread, the implementation of sorted tetrahedral cells corresponding to the data in vertex array one by one is simple. However, it is a tricky in the multi-threads runtime cause that the classification/

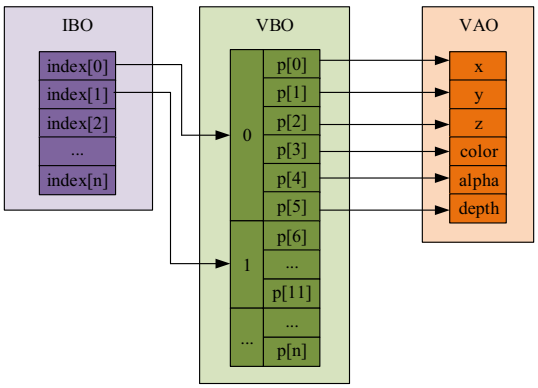

Fig. 4 Relation among vertex buffer object (VBO), vertex array object (VAO) and index buffer object (IBO)

decomposition of projection polygons performed simultaneously in different thread will lead to massive messy triangles in final image. In this paper, to address the disorder data in vertex array, we store the vertex data into vertex array with offset computation that ensures correct data order in vertex array.

According to Sect. 3.2.1, arbitrary tetrahedral projection produced four triangles with normalization processing, that is, 12 vertex index values. As shown in Fig. 3, *class* 1 and *class* 2 produce four and five projection points, respectively, with internal intersection points included. In this paper, we consider that *class* 1 produced five projection points, and extra projection vertex point will never be used and just convenient us to compute the offset of vertex data. Then, one tetrahedral projection produced 30 vertex attribute values after decomposition processing. Therefore, when processing one tetrahedral cell, the offset of *IBO* and *VBO* is 12 and 30, respectively. The offset relation is successfully established.

In Sect. 3.1, we performed multi-threaded parallel visibility sorting for all tetrahedral cells and obtained a series of ordered subarrays which stored tetrahedral IDs. Then, we performed the classification/decomposition processing for all tetrahedral cells in each subarray (see in Algorithm 2) and stored the produced vertex data into vertex array. A two-layer nested loop is necessary in our algorithm, the outer loop traverses all subarrays, while the inner loop processes tetrahedral cells in each subarray. We can calculate the correct position of vertex data in vertex array with offset computation using iteration factor, which ensures that the vertex array accessing is thread-safe. Therefore, we can accelerate the classification/decomposition processing by paralleling two-layer nested loop with multi-thread.

Algorithm 2 Classification and decomposition

Input: Sorted tetrahedral cell subarrays.
1: #pragma omp parallel **for**
2: **for** each *subarray* **do**
3: calculate the *IBO* and *VBO* shift of *subarray*;
4: #pragma omp parallel **for**
5: **for** each *cell* **in** *subarray* **do**
6: calculate the *IBO* and *VBO* shift of *cell*;
7: calculate α and β in Eq. 3;
8: decompose the projection of *cell* into triangles;
9: store vertex data into *IBO* and *VBO*;
10: **end for**
11: **end for**
Output: *IBO* and *VBO* of tetrahedral cells.

3.3 Rasterization and rendering

The PT algorithm loads triangle vertex data produced by the classification/decomposition of tetrahedral projections into GPU memory. The GPU rasterizes vertices and renders semi-transparent triangle to approximate the volume rendering results. We obtain vertex array with the classification/decomposition

processing. And vertex array can be sent to the GPU memory though the OpenGL API. In this paper, we used the alpha compositing method to assemble into final image from back to front order, and each pixel value in the frame buffer will be calculated according to the following formula:

$$C_{new} = \alpha C + (1 - \alpha) C_{old}$$

where C_{new} is the new pixel color, C is the interpolated color of the polygon at that pixel, α is the interpolated opacity, and C_{old} is the current pixel value, originally just the background color.

Finally, the approach produced massive semi-transparent triangles (four times the tetrahedral cells), which are assembled into final image by the GPU.

4 Experiments and discussion

For the experimental evaluation, a platform consisting of DELL graphic workstation, with a 2.3GHz Intel(R) Xeon(R) CPU E5-2650v3, with 20 cores and 64-GB RAM, is utilized. The operating system is Windows 7 64 bit. All the programs were implemented in C/C++ using the OpenMP and are compiled using Visual Studio 2015 with the -O3 -fopenmp compiler optimization flags. In order to compare the performance, the execution time is used as a measure. The computation time is measured from the execution of rendering algorithms only, excluding grid reading from disk, etc. All time complexities are measured using the ftime() function of C/C++. To decrease random variation, all the execution time complexities are measured as the average of repeat tests.

Our experiments include five subparts. First, we provide a comparison between parallel and sequential versions of our algorithm. Second, we further explore how the different numbers of threads in two-layer nested loop, as shown in Algorithm 2, affect the performance. Third, we carefully conduct the comparison of our volume rendering algorithm to the implementation in VTK. Moreover, to further discuss the bottleneck, we compare the breakdown of execution time. Finally, a scalability analysis is given to investigate the performance in depth.

4.1 Performance of our algorithm

We have implemented parallel algorithms for the visibility sorting and classification/decomposition of projection polygons. Therefore, we test the different numbers of visibility sorting threads (*VST*) and classification/decomposition threads (*CDT*, denoting the number of outer loop threads) effect on our algorithm. We did three groups' experiment with the variable-controlling approach, where *VST* is set to 1 (only visibility sorting parallelization), *CDT* is set to 1 (only classification/decomposition parallelization) and *VST* equals *CDT* (both parallelization of two parts), respectively. Several sets of test unstructured grid are used to evaluate the performance, with various cell sizes of 1029 k, 1440 k, 2205 k and 3396 k.

As shown in Fig. 5, we gather the average speedup of our algorithm relative to sequential PT algorithm. The experimental results show that our algorithm achieves the maximum speedup of 3.4X with both *VST*

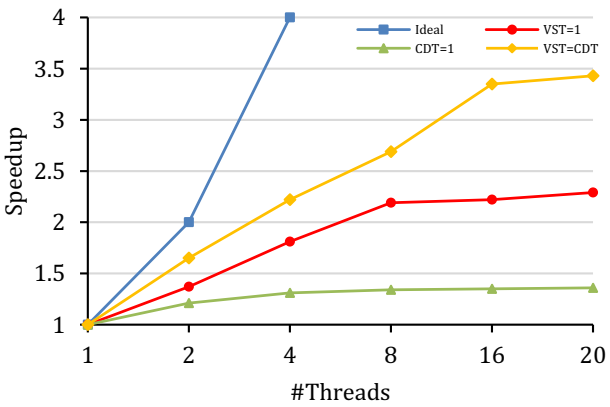

Fig. 5 Speedup of our algorithm with different #Threads relative to sequential algorithm

and *CDT* utilizing 20 threads. We can observe that the speedup of $CDT = 1 < VST = 1 < VST = CDT < Ideal$ in terms of performance, which is true for all parallel systems, and is often referred to as Amdahl's law. And the reason for the result is that the classification/decomposition processing consumes more time than visibility sorting processing in whole volume rendering algorithm. The speedup achieved by our algorithms cannot obviously increase since worker threads are larger than eight. There are many factors that prevent the performance improvement. In general, the memory bandwidth affects negatively high-performance computing, causing the processors idle as they wait for memory accessing, which is common problem in parallel processing, also called memory wall (Hutcheson and Natoli 2011). Moreover, the multi-threads runtime initialization is not negligible in parallel computing, for instance parallel region creation, thread synchronization, etc. Furthermore, the tetrahedral depth array partition and vertex data construction in our algorithm cause extra time consumption relative to the sequential algorithm.

To facilitate the parallel classification/decomposition of projection polygons processing, we construct extra triangle vertices to fill vertex array and that can produce correct rendering results in multi-threads runtime. As shown in Fig. 6, the results produced by our algorithm are same as sequential PT algorithm.

4.2 Two-layer nested loops analysis

In order to determine whether the proposed parallel algorithm is affected by inner loop parallelization, another control variable-controlling method is used. We maintain the $VST = 1$ state and just change the *#Threads* of outer and inner loop of Algorithm 2. Table 1 presents the execution time for different *#Threads* in outer and inner loop parallelization. We find that inner loop parallelization has lower effect on the speedup of classification/decomposition compared with outer loop. Inner loop parallelization is nested parallelization, which means less parallelism and more multi-threads environment initialization/synchronization time consumption. On the contrary, outer loop enjoys more parallelism and only needs to initialize once, meanwhile, the problem of nested loops, the basic constraints are also dependences and resources. The execution times marked with bold face in Table 1 show the similar performance with different thread combinations. In general, the operating system and compiler provide optimization algorithms to make full use of processor, for instance memory accessing optimization, load balancing optimization, multi-thread optimization, etc. Furthermore, the experimental platform only has 20 hardware cores, and enlarging total *#Threads* does not reduce the executing time.

4.3 Performance of different algorithms

We find that VTK-m has no implementation of existing volume rendering algorithm for unstructured grid, so we adopt VTK as benchmark to demonstrate our algorithm performance. As shown in Fig. 7, including the PT algorithm, VTK also implements HAVS (Callahan et al. 2005) algorithm, software-based ray casting algorithm (SW ray casting) (Bunyk et al. 1997) and ZSWEEP (Farias et al. 2000) algorithm for unstructured grid volume rendering. We test HAVS algorithm with *kbuffer* which equals 2 and 6, respectively. Volume rendering results of the VTK implementations are shown in Fig. 7. We can see that except the result rendered by HAVS algorithm exist visual artifacts, and the others are the same. Due to the visual artifacts and deprecation since VTK ver. 8.2, the HAVS algorithm will not be shown in flowing experiments.

Table 2 presents that our algorithm shows the best performance, followed by the SW ray casting and VTK PT algorithms, and ZSWEEP algorithm is much less efficient than other algorithms. To obtain the best performance, we test our algorithm that visibility sorting and classification/decomposition processing both utilize 20 threads. The experimental results show that our algorithm obtains a speedup of 2.5X($\frac{0.35}{0.13}$) relative to the fastest implementation in VTK. In addition, the performance shows that our parallel PT algorithm drastically improves the speed of unstructured grid volume rendering and provides interactive performance for analyzing intermediate results. Partial rendering results of our algorithm are shown in Fig. 8.

4.4 Execution time breakdown

To better understand the bottleneck of our optimization, we breakdown the execution time into four parts: partial sorting, global sorting, classification/decomposition and rasterization. Four groups' control experiment results present the performance of various size cell of 1029 k, 1440 k, 2205 k and 3396 k, and to explore best performance, we maintain $VST = CDT$ state, as shown in Fig. 9. As expected, the sequential partial sorting and rasterization show similar performance with 1/20 work threads. Meanwhile, since global

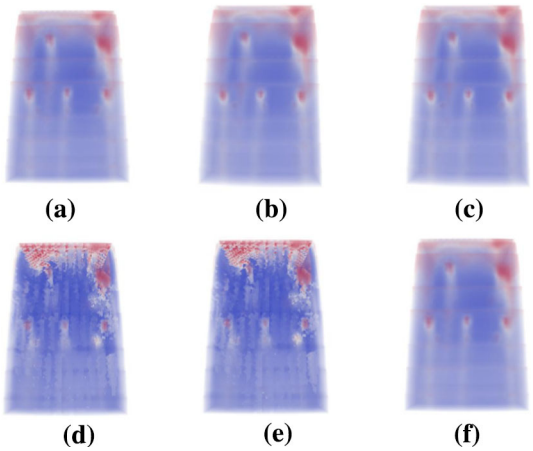

Fig. 6 Results of rendering comb dataset: **a** PT algorithm, **b** SW ray casting algorithm, **c** ZSWEEP algorithm, **d** and **e** HAVS algorithm with kbuffer = 2 and 6, respectively, **f** our algorithm

Table 1 Execution time in second among different numbers of threads in two-layer nested loops

		#Threads (outer loop)					
		1	2	4	8	16	20
#Threads (inner loop)	1	1.51	1.05	0.87	0.75	**0.68**	**0.68**
	2	1.45	0.94	0.81	0.73	**0.67**	**0.66**
	4	1.42	0.91	0.78	0.70	**0.67**	**0.68**
	8	1.40	0.94	0.78	**0.69**	**0.66**	**0.67**
	16	1.31	0.91	0.79	**0.68**	**0.67**	**0.68**
	20	1.28	0.92	0.77	**0.67**	**0.67**	**0.67**

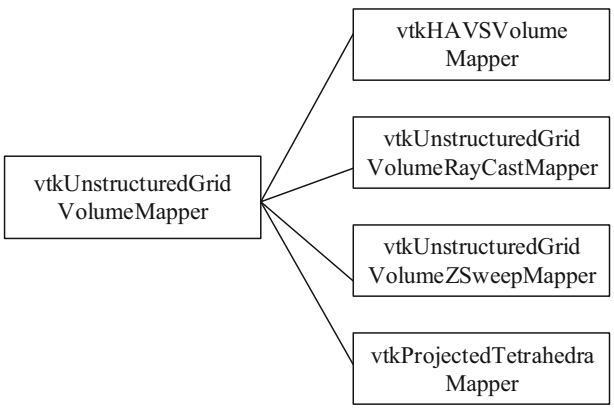

Fig. 7 Volume rendering algorithms for unstructured grids in VTK

sorting and classification/decomposition exploit more parallelism, execution time spent in both of two phases is significantly reduced, and the other two phases dominate the total execution time of our algorithm (see Fig. 9 two consecutive columns for comparison).

The execution time breakdown further analyzes time consumption of each phase. Classification/decomposition involves extensive vector, distance and interpolation calculations which cause inefficiency; as expected, experimental results show that classification/decomposition is the most time-consuming phase, and multi-threaded acceleration gains obvious speedup relative to sequential version. Moreover, our strategy also efficiently processes the visibility sorting by accelerating global sorting.

Table 2 Time comparison in second among different algorithms

#Cell	Algorithms			
	VTK PT	SW ray casting	ZSWEEP	Our PT
1029k	0.57	**0.35**	289.79	**0.13**
1440k	0.66	0.53	330.45	0.17
2205k	1.23	0.67	653.56	0.28
3396k	1.61	1.36	930.66	0.44

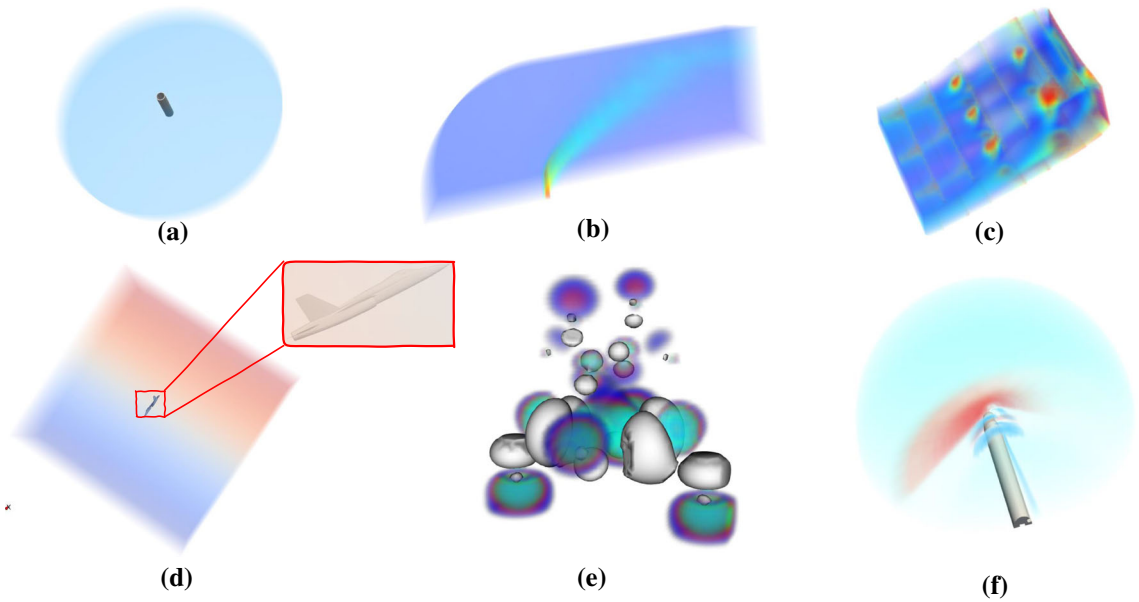

Fig. 8 Results of rendering our algorithm: **a** liquid oxygen post dataset, **b** blunt fin dataset, **c** comb dataset, **d** yf17 dataset, **e** iron protein dataset and **f** space shuttle launch vehicle dataset

4.5 Scalability

The scalability of a parallel system is the ability to increase performance as the number of the processors increases (Spiliotis et al. 2020). The speedup does not increase linearly with the number of processors; instead, it tends to saturate (see in Fig. 5, *#Threads* larger than eight). The parallel efficiency (Grama et al. 1993) is significant measurement for the scalability analysis of a parallel system, which is defined as the ratio of speedup to the number of processors. A scalable parallel system should increase speedup in such a rate that the efficiency is maintained as the number of processors increases. Figure 10 presents parallel efficiency of our algorithm in the different *#Threads* runtime with various *#Cells*; we find that our algorithm maintains a good scalability when *#Threads* is not exceeding eight; for instance, we increase *#Cells* and *#Threads*, and the efficiency can maintain at 0.65, see in Fig. 10. However, with *#Threads* increases, parallel overhead is not negligible part in the total execution time, and our multi-threaded PT algorithm has lower scalability when *#Threads* is larger than eight.

5 Conclusion

In this paper, we propose a multi-threaded parallel PT algorithm for unstructured grid volume rendering. Our approach provides multi-threaded implementation of visibility sorting processing and classification/decomposition of projection polygon processing. In the visibility sorting phase, we first partition disordered tetrahedral depth array into several subarrays with no intersection of numeric intervals and then sort each subarray with concurrent multi-threads. In the classification/decomposition phase, for single tetrahedral cell,

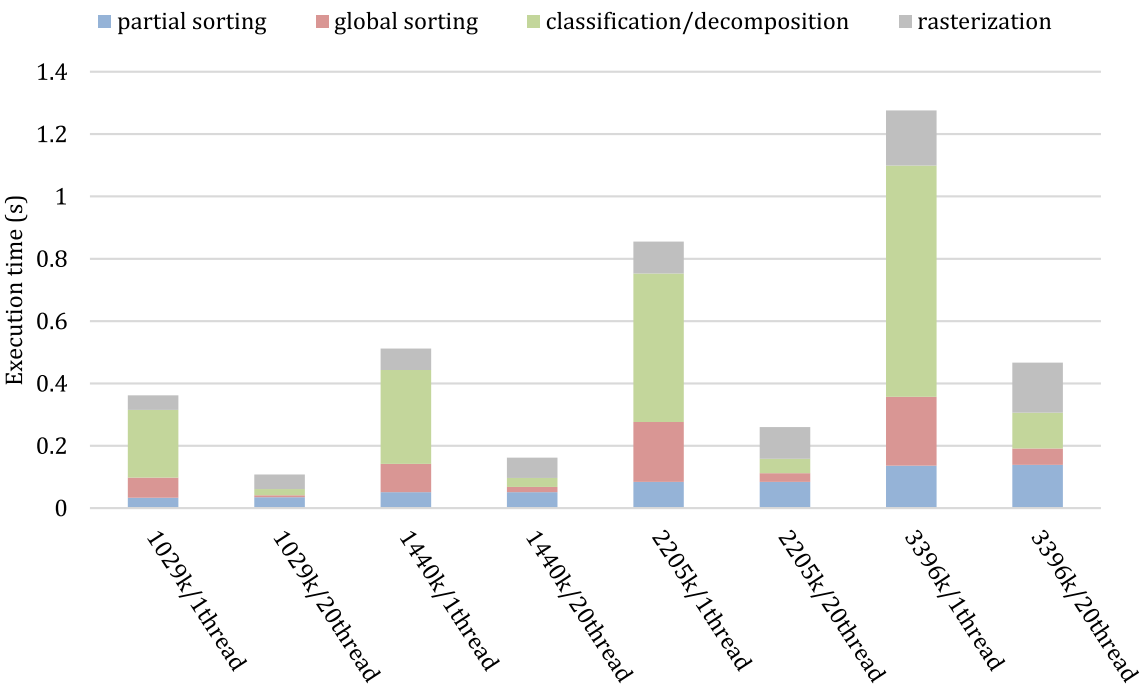

Fig. 9 Execution time of our proposed multi-threaded PT algorithm implementation. Note that labels on the abscissa axis denote #Cells and #Threads, respectively

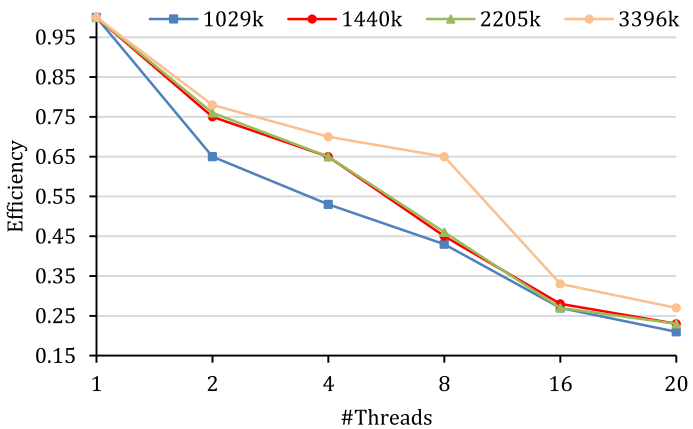

Fig. 10 Parallel efficiency of our proposed multi-threaded PT algorithm implementation. Note that legends denote #Cells of dataset

we first normalize the various tetrahedral projections to ensure that all tetrahedral projections produce the same number of vertex, and then store the vertex data into vertex array with offset computation. For the whole tetrahedral grid, we process each tetrahedral cell with multi-threaded method. The loosely coupled visibility sorting and classification/decomposition algorithm ensure that PT algorithm can perform correctly in the multi-threaded runtime.

Our method is a volume rendering algorithm based on software implementation, which can process the arbitrary shape of unstructured gird. The algorithms also applied to the unstructured grid are HAVS, ZSWEEP, ray casting algorithms, etc. In this paper, we compared the performance efficiency of our algorithm and other techniques, where our algorithm is the most efficient without visual artifacts. Otherwise, we calculated the speedup of our algorithm in different *#Threads* relative to sequential implementation with repeat tests. The experimental results show that our algorithm achieves a speed of 3.4X on a 20 cores CPU

and outperforms the fastest VTK implementation at a speedup of 2.5X. In addition, to further investigate performance and bottleneck, we also give the breakdown of execution time and scalability analysis. Compared with conventional volume rendering algorithm, our approach not only ensures the utilization efficiency of multi-core processor, but also dramatically boosts the algorithm performance, which provides interactive performance for analyzing intermediate results.

There are several optimizing works for future research. First, in our algorithm, we processed all tetrahedral cells and sent the vertex data to GPU memory for rasterization once. When grid is large, the vertex array will severely consume CPU/GPU memory. In the future work, we will explore sending the vertex data to GPU memory in batches and reduce memory consumption by reusing the vertex arrays. Otherwise, to implement the multi-threaded parallel classification/decomposition of projection polygons, we normalized tetrahedral projection and constructed extra vertex data to fill vertex array. Although vertex data have no effect on final results, extra data increase the CPU/GPU memory consumption and data transfer overheads, and GPU rasterizing extra vertex will increase time-consuming. Therefore, it is also the focus of future work to remove these vertex data from vertex array while ensuring correct rendering result. Finally, our algorithm implementation is based on VTK and OpenMP, and the other parallel frameworks, such as NVidia CUDA and Intel TBB, are ill-considered in our work. The multi-threaded PT algorithm based on multiple parallel frameworks comparison and optimization is another interesting direction for our future work.

Acknowledgements The authors wish to thank Hung, Ekaterinaris, Rogers and Buning for providing available test datasets. This work was supported by the National Numerical Windtunnel Project and the Key Technology Research of Service Robot Basic Software (2020YFG0031).

References

Antonio F (1992) Faster line segment intersection. In: Blair Kirk D (ed) Graphics Gems III (IBM Version), The Graphics Gems Series. Academic Press, Cambridge, pp 199–202

Bunyk P, Kaufman AE, Silva CT (1997) Simple, fast, and robust ray casting of irregular grids. In: Hagen Hans, Nielson Gregory M, Post Frits H (eds) Dagstuhl '97, Scientific Visualization, Dagstuhl, Germany, 9-13 June 1997. IEEE Computer Society, pp 30–36

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

Farias RC, Mitchell JSB, Silva CT (2000) ZSWEEP: an efficient and exact projection algorithm for unstructured volume rendering. In: Lorensen William E, Crawfis Roger, Cohen-Or Daniel (eds) Proceeding of the 2000 Volume Visualization and Graphics Symposium, VVS 2000, Salt Lake City, Utah, USA, October 9-10, 2000. ACM / IEEE Computer Society, pp 91–99

Garrity Michael P (1990) Raytracing irregular volume data. Comput Graph 24(5:35–40

Grama A, Gupta A, Kumar V, (1993) Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib Technol Syst Appl 1(3):12–21

Hutcheson A, Natoli V (2011) Memory bound versus compute bound: a quantitative study of cache and memory bandwidth in high performance applications. In: Technical report, Stone Ridge Technology

Kaufman AE (1996) Volume visualization. Vis Comput 28(1):165–167

Kaufman AE, Mueller K (2005) Overview of volume rendering. In: Hansen Charles D, Johnson Christopher R (eds) The visualization handbook. Academic Press / Elsevier, Cambridge/Amsterdam, pp 127–174

Marroquim R, Maximo A, Farias RC, Esperança C (2008) Volume and isosurface rendering with gpu-accelerated cell projection*. Comput Graph Forum 27(1):24–35

Max NL (1995) Optical models for direct volume rendering. IEEE Trans Vis Comput Graph 1(2):99–108

Max N, Hanrahan P, Crawfis R (1990) Area and volume coherence for efficient visualization of 3d scalar functions. Acm Siggraph Comput Graph 24(5):27–33

Maximo A, Marroquim R, Farias RC (2010) Hardware-assisted projected tetrahedra. Comput Graph Forum 29(3):903–912

Min C, Kaufman AE, Yagel R (2000) Volume graphics. Springer, Berlin

Miranda Fábio M, Filho WC (2011) Accurate volume rendering of unstructured hexahedral meshes. In: Lewiner T, da Silva TR (eds) 24th SIBGRAPI Conference on Graphics, Patterns and Images, Sibgrapi 2011, Alagoas, Maceió, Brazil, August 28-31, 2011. IEEE Computer Society, pp 93–100

Mueller C, Strengert M, Ertl T (2007) Adaptive load balancing for raycasting of non-uniformly bricked volumes. Parallel Comput 33(6):406–419

Newell M, Newell R, Sancha T (1998) A solution to the hidden surface problem, pp 27–32. 07

Shirley P, Tuchman A (1990) A polygonal approximation to direct scalar volume rendering. Acm Siggraph Computer Graphics

Spiliotis IM, Bekakos MP, Boutalis YS (2020) Parallel implementation of the image block representation using openmp. J Parallel Distrib Comput 137:134–147

Stein CM, Becker BG, Max NL (1995) Sorting and hardware assisted rendering for volume visualization. In: Kaufman Arie E, Krueger Wolfgang (eds) Proceedings of the 1994 Symposium on Volume Visualization, VVS 1994, Washington, DC, USA, October 17–18, 1994. ACM, pp 83–89

Weiler M, Kraus M, Merz M, Ertl T (2003) Hardware-based ray casting for tetrahedral meshes. In: Turk G, van Wijk JJ, Moorhead II Robert J (eds) 14th IEEE Visualization 2003 Conference, VIS 2003, Seattle, WA, USA, October 19-24, 2003. IEEE Computer Society, pp 333–340

Weiler M, Mallón PN, Kraus M, Ertl T (2004) Texture-encoded tetrahedral strips. In: 2004 IEEE Symposium on Volume Visualization and Graphics, VolVis 2004, Austin, TX, USA, October 10-12, 2004, pp 71–78. IEEE Computer Society

Wilhelms J, Van Gelder A (1991) A coherent projection approach for direct volume rendering. In: Thomas JJ (ed) Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991. Providence, RI, USA, April 27-30, 1991, pages 275–284. ACM

Williams PL (1992) Interactive splatting of nonrectilinear volumes. In: Kaufman AE, Nielson GM (eds) Proceedings IEEE Visualization '92, Boston, Massachusetts, USA, October 19-23, 1992. IEEE Computer Society, pp 37–45

Williams PL (1992) Visibility-ordering meshed polyhedra. ACM Transactions on Graphics

Wylie B, Moreland K, Fisk LA, Crossno P (2002) Tetrahedral projection using vertex shaders. In: IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics, VolVis 2002, October 28–29, 2002, Boston, MA, USA