J Vis
DOI 10.1007/s12650-016-0359-5

REGULAR PAPER

Tao Hou · Li Chen

On-the-fly simplification of large iso-surfaces with per-cube vertex modifiability detection

Received: 2 July 2015 / Revised: 3 January 2016 / Accepted: 29 February 2016

Abstract Marching Cubes based iso-surface extraction is widely used for data visualization. However, the increasing size of volume sets has made extracted iso-surfaces difficult to manipulate, and applying out-of-core simplification on them is considerably slow. We present an on-the-fly simplification algorithm for out-of-core iso-surfaces generated by Marching Cubes based extraction. Our algorithm shifts between extraction and decimation during the processing of volume sets, and never stores the entire extracted iso-surface in the main memory. The key of our algorithm is that we exploit the extraction pattern of Marching Cubes to determine when the mesh operator can be applied on certain generated vertices. This enables the decimation to be applied after any specified number of triangles are extracted. It also provides a framework for on-the-fly processing of large iso-surfaces. Our algorithm is more efficient than cascading out-of-core extraction and simplification, while providing high simplification quality comparable to in-core algorithms.

Keywords Iso-surface simplification · On-the-fly processing · Out-of-core processing · Volumeset visualization

1 Introduction

The visualization of volume data created from many domains has become an important method for observing data and discovering knowledge behind it. One straightforward way to visualize volume data is to explore each slice, which, however, is difficult for the observer to grasp a general view of the data. In contrast, one can usually convert the 3D data into an intermediate format (i.e., the iso-surface), which enables one or more phenomena or structures of interest in a dataset to be rendered independently. Because iso-surfaces are typically polygonal meshes, the rendering is considerably fast on modern graphics hardware.

The improvements in resolution and accuracy of data acquisition devices have led to fast increasing in volume data size, making the extraction and rendering of iso-surfaces a big challenge for commodity PCs, especially when the iso-surfaces fail to fit into main memory. In order to cope with this, iso-surfaces need to be simplified to reduce the number of triangles for memory storage, and more importantly for interactive visualization.

This work was supported in part by the National Science Foundation of China (61272225, 61572274, 91515103, 51261120376).

T. Hou · L. Chen
School of Software, Tsinghua University, Beijing, China

T. Hou (✉)
VMware Information Technology (China) Co. Ltd, Beijing, China
E-mail: houtao.sh@qq.com

Published online: 08 April 2016

In this paper, we present an on-the-fly simplification algorithm for iso-surfaces extracted from large volume sets using a method similar to Marching Cubes (Lorensen and Cline 1987). Our algorithm interleaves the extraction and simplification on partially extracted iso-surfaces and never needs the finest level iso-surface to be completely kept in the main memory. Using our algorithm, no separate processes of extraction and simplification are needed, so that the headaches of dereferencing or reconstructing the topology of large out-of-core meshes are eliminated. A simplified iso-surface can be produced directly from the volume set. The extraction pattern of Marching Cubes (Lorensen and Cline 1987) is exploited to help the algorithm explicitly determine when a generated vertex can be manipulated by the mesh operator. The main contributions of our approach are:

– An on-the-fly iso-surface simplification algorithm for large volume sets without needing to store all extracted faces. It runs fast, consumes little memory, and yields high simplification quality.
– A general framework for on-the-fly iso-surface processing of large volume sets based on the per-cube vertex modifiability detection. It allows any specified number of faces to be extracted before the processing of iso-surface, and constrains only a small portion of unmodifiable area to be preserved during processing.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the iso-surface extraction and massive mesh simplification algorithms. Section 3 gives a general description and some critical points of our algorithm. Section 4 lists some implementation details. Section 5 presents the results. Section 6 gives some discussion and Sect. 7 draws a conclusion.

2 Related work

2.1 Iso-surface extraction

A number of methods have been proposed concerning the extraction of iso-surfaces from volume data. Some deal with structured cubes and produce evenly tiled iso-surfaces, while others generate simplified iso-surfaces from unstructured grids.

The Marching Cubes algorithm (Lorensen and Cline 1987) is the best-known iso-surface extraction algorithm. It produces triangular faces based on a lookup table and uses linear interpolation to derive the coordinates. Though Marching Cubes algorithm produces iso-surfaces with rather high quality, it has some ambiguity problems (Newman and Yi 2006). Several approaches (Shirley and Tuchman 1990; Van Gelder and Wilhelms 1994; Nielson and Hamann 1991) have been proposed to produce disambiguous iso-surfaces. Our method is applicable to iso-surfaces produced by MC-like methods that treat the volume data cube by cube.

Instead of tiling faces evenly on each region, adaptive extraction algorithms produce multiresolution grids for regions of the iso-surface with different degrees of details, and can reduce the amount of faces inherently. Dual Contouring (DC) (Ju et al. 2002) places vertices of iso-surface dual to octree cubes, and can reproduce sharp features. Dual Marching Cubes (DMC) (Schaefer and Warren 2005) generates polyhedral grids dual to the original octree cubes, and extracts iso-surfaces from the dual grids. A simplicial partition method (Manson and Schaefer 2010) partitions the octree cubes into topologically consistent tetrahedra using dual vertex for each line, face, and cube. It then extracts iso-surfaces using Marching Tetrahedra (Akio and Koide 1991). Peng et al. (2014) proposed an interactive mesh-importance specifying method using transfer functions, and introduced a parallel adaptive iso-surface extraction framework using Pyramid Peeling to progressively generate simplified iso-surface based on mesh importance.

However, most of these adaptive extraction algorithms suffer from spoiled topology of the iso-surface. Though the simplicial partition method (Manson and Schaefer 2010) produces manifold and intersection-free iso-surfaces, the simplification quality of all these adaptive extraction algorithms cannot be compared to those iterative simplification algorithms (Schroeder et al. 1992; Hoppe 1996; Garland and Heckbert 1997). Another drawback of adaptive extraction is that it fails to deal with out-of-core datasets.

2.2 Out-of-core simplification

The research of mesh simplification began with the in-core algorithm. However, as meshes, which became larger and larger, finally exceeded memory capacity, few in-core algorithms worked. This was when out-of-core simplification became a hot topic of research.

A category of out-of-core approaches cut the mesh into pieces and simplify them individually, then reassemble the simplified pieces. Some methods (Bernardini et al. 2002; Prince 2000) cut the mesh by partitioning the triangles and constrain those boundary vertices to be preserved during the decimation. Another phase of treatment is needed to decimate the over-tessellation near the cutting boundary. Brodsky and Pedersen (2003) proposed a cutting method by partitioning the vertices, which produces no over-tessellation after stitching.

The out-of-core vertex clustering (Lindstrom 2000) extends the in-core version (Rossignac and Borrel 1993) by batch processing the mesh, and improves the representative vertex positioning using QEM (Garland and Heckbert 1997). A memory-insensitive version (Lindstrom and Silva 2001) further extends the out-of-core vertex clustering using external sort. Though the vertex clustering methods run fast, they produce drastic simplified meshes.

An external mesh data structure (Cignoni et al. 2003) is proposed to enable the connectivity query of large meshes. Though simplification using this data structure produces high-quality simplified meshes, the maintenance of the data structure considerably slows down the simplification.

The stream simplification algorithm (Wu and Kobbelt 2003) keeps a constant-size buffer for the large mesh, and decimates only the buffered portion. It continuously reads faces from the mesh file and writes processed faces into the output file. It decimates only vertices surrounded by a closed ring of triangles and treats boundary edges as those that are adjacent to only one triangle. One drawback of this is that it is unable to differentiate the processing boundary (the boundary between the processed and unprocessed portion) from the real mesh boundary. It will have to keep the real mesh boundary in memory until all triangles are processed. Another drawback is that the detecting method fails on the non-manifold regions of the mesh.

Isenburg et al. (2003) proposed a large mesh processing paradigm called processing sequences. Processing sequences are particular interleaved ordering of triangles and vertices, with vertices always preceding the first triangle that references them. They demonstrated the use of processing sequences by adapting the stream simplification (Wu and Kobbelt 2003).

Because some elaborately designed out-of-core simplification algorithms can produce high simplification quality comparable to in-core algorithms, applying them after fully generating the whole iso-surface is straightforward and acceptable. However, cascading these two procedures causes too much redundant operations compared to our on-the-fly algorithm, which decimates the iso-surface immediately after it is partially extracted.

2.3 Tandem algorithm

The tandem algorithm (Attali et al. 2005) extracts and immediately simplifies the iso-surface as each layer of cubes is processed. The algorithm delays the collapse of newly generated triangles using a mesh isotropy criterion to avoid long and skinny triangles, producing better triangulation. However, the layer-by-layer approach places a constraint on the number of faces generated before decimation, while our algorithm permits an arbitrary face increment.

3 Algorithm overview

An illustration of the structure of the on-the-fly algorithm is given by Fig. 1. Taking the volume data as input, our algorithm needs the user to specify two parameters: R and B. R is the decimation rate, which is the expected ratio between the face count of the simplified iso-surface and the original iso-surface. B is used to control the balance between time cost and simplification quality. Since our algorithm interleaves extraction and simplification, in each alternation, certain amount of faces (specified by B) are extracted at first, then simplification is performed. A bigger B value means more surface elements are dedicated for decimation, which may lead to a better simplification quality. However, it also means higher memory consumption and longer execution time (see Sects. 4.3 and 5 for more test details). Faces are extracted in a cube-by-cube way. After extracting faces from one cube, our algorithm converts the faces into indexed format and makes them ready for simplification. The extraction–simplification alternation repeats until the whole iso-surface is extracted and simplified.

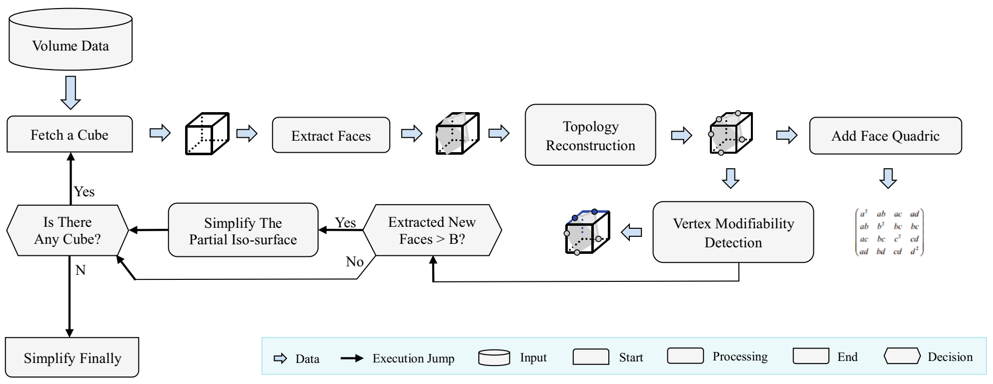

Fig. 1 General procedure of our on-the-fly iso-surface simplification algorithm

3.1 Per-cube vertex modifiability detection

One key part for designing on-the-fly algorithm of iso-surfaces is to decide when a vertex can be manipulated by the mesh operators (i.e., *modifiable*). According to the “processing sequences” (Isenburg et al. 2003), a vertex is considered modifiable when its adjacent faces in the mesh data file have all been processed. Mesh operators cannot be applied on unmodifiable vertices because it will be difficult to stitch the subsequently input faces to the boundary between the processed and unprocessed portions once modified. Another important reason is that triangles around unmodifiable vertices have not been completely generated; it will be impossible to evaluate the cost of simplification operators concerning these vertices.

We found that, based on the extraction pattern of Marching Cubes, vertices can be claimed modifiable in a cube-by-cube way. The key lies in the relationship between the primitives of MC’s cubes (corner, line, face, cube) and the primitives of the extracted faces (vertex, edge, triangle). MC marks each corner of the cube with a boolean sign based on whether the corner’s scalar value is below or above the iso-value. In a cube containing the iso-surface (having both kinds of vertices), for each line that exhibits a sign change, an iso-surface’s vertex is generated along the line; for each face that exhibits a sign change, iso-surface’s edges are generated on the face. It should be noted that if the scalar value on a corner of the cube happens to be the iso-value or is infinitely close, the vertex generated for the iso-surface would coincide with the corner of the cube. For a vertex placed on the line but not on the corner, all its adjacent triangles lie in the four adjacent cubes of the line. For a vertex placed right on the corner, all its adjacent triangles lie in the eight adjacent cubes of the corner. Figure 2 gives examples of the two scenarios.

As illustrated in Fig. 3a, if we assume that the algorithm traverses the volume data from left to right on the x axis, from front to back on the y axis and from top to bottom on the z axis, the seven adjacent cubes to the left, front, and top of any currently processing cube have been traversed. Thus, after a cube is processed, lines numbered 4, 7, 8, and corner numbered 4 in Fig. 3b have had all their adjacent cubes traversed. This means that for the vertices generated on these lines and corner, all the triangles adjacent to them have been generated. Therefore, these vertices become modifiable. Most of the internal cubes follow this rule. However, there are cubes that lie on the boundary of the volume data and more vertices may become modifiable after the processing of these cubes. These are cubes that lie on the right boundary, back boundary, bottom boundary, and combinations of them. Based on the numbering of Fig. 3b, Table 1 lists the different kinds of cubes and certain primitives of the cubes. The vertices generated on these primitives are claimed modifiable after the corresponding cube is extracted.

The *per-cube vertex modifiability detection* solves the modifiability detection problem better than similar approaches (Wu and Kobbelt 2003; Attali et al. 2005), and provides a framework for on-the-fly mesh processing with greater flexibility.

3.2 Incremental mesh simplification

We adopted an incremental simplification similar to the stream simplification (Wu and Kobbelt 2003). Iterative edge contraction (Hoppe 1996; Garland and Heckbert 1997) is used for decimation of iso-surface with QEM (Garland and Heckbert 1997) measuring the contraction cost. The way to eliminate unmodifiable

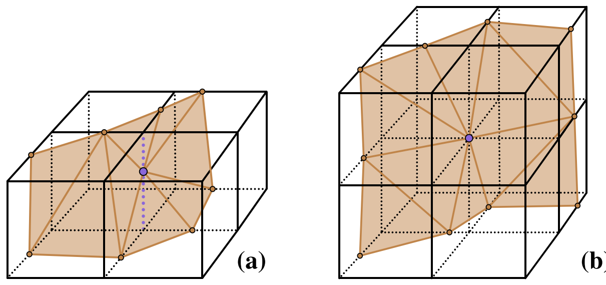

Fig. 2 **a** The *purple vertex* is placed on the line; all its adjacent triangles lie in the four adjacent *cubes* of the line. **b** The *purple vertex* is placed on the corner; all its adjacent triangles lie in the eight adjacent cubes of the corner

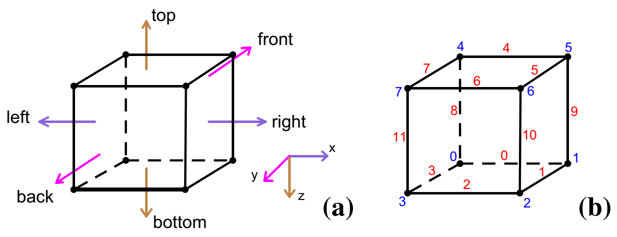

Fig. 3 **a** Traversing direction, **b** line and corner numbering

Table 1 Different kinds of cubes and certain primitives of the cubes. The vertices generated on these primitives are claimed modifiable after the corresponding cube is processed

Cube	Modifiable corner	Modifiable line
Internal	4	4, 7, 8
Right-bound	4, 5	4, 5, 7, 8, 9
Back-bound	4, 7	4, 6, 7, 8, 11
Bottom-bound	0, 4	0, 3, 4, 7, 8
Right-back	4, 5, 6, 7	4, 5, 6, 7, 8, 9, 10, 11
Right-bottom	0, 1, 4, 5	0, 1, 3, 4, 5, 7, 8, 9
Back-bottom	0, 3, 4, 7	0, 2, 3, 4, 6, 7, 8, 11
The last traversed	All	All

vertices from contraction is to keep some edges uncontractible. An edge is considered uncontractible when at least one of its vertices is unmodifiable. Figure 4 illustrates the extraction–simplification alternation.

Because our algorithm produces simplified iso-surfaces that are used for interactive rendering, we keep the (simplified) partially extracted iso-surface in core permanently and never write faces to the disk. We will call it the intermediate iso-surface in the following text. Because the memory consumption is somehow only proportional to the output iso-surface (see Sect. 4.3), and the output iso-surfaces mostly have to fit into the main memory for rendering, this may not limit the usage. One can also easily adapt our algorithm to a "completely stream" version, which produces memory-independent outputs like the stream simplification (Wu and Kobbelt 2003). A benefit of never writing faces to disk is that the whole intermediate iso-surface is capable of being decimated. It increases the amount of contractible edges.

4 Algorithm details

4.1 Face extraction

After extracting faces from one cube, following steps need to be performed in order to make the newly extracted region ready for decimation:

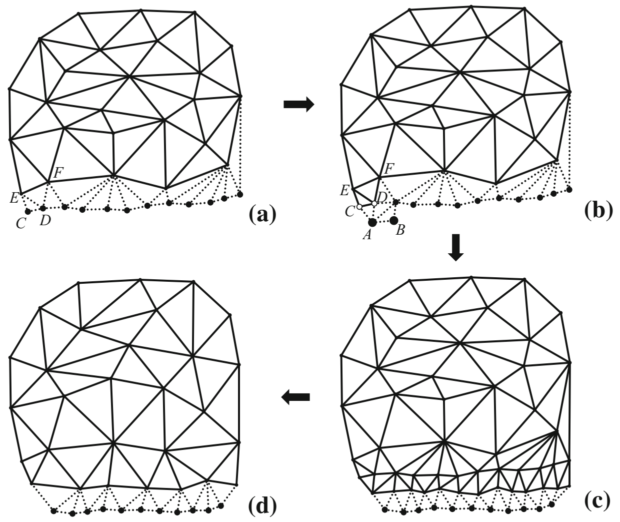

Fig. 4 One time of the extraction–simplification alternation. The vertices in *bold* are unmodifiable and the edges displayed as *dashed line* are uncontractible. **a** The intermediate iso-surface after the previous extraction and decimation. **b** Three faces extracted from a cube is added; two new vertices A and B are introduced; D and C are claimed modifiable; edges EC, DC, DE, and DF become contractible. **c** More faces are added to the intermediate iso-surface and the user-specified amount (B) is reached. **d** The intermediate iso-surface is decimated

- *Topology reconstruction* Converts the extracted triangle soups into indexed format based on the index hash table and adds newly introduced vertices into the index hash table;
- *Adding face quadric* Calculates the quadric matrix (Garland and Heckbert 1997) of each face and adds the quadric matrix to corresponding vertices' quadric matrices;
- *Vertex modifiability detection* Claims certain vertices modifiable based on Table 1. For each vertex claimed modifiable, the corresponding entry in the index hash table is first deleted, then edges that become contractible are added to a priority queue for decimation.

For topology reconstruction, an index hash table is maintained to map the coordinates of vertices to their indices.

4.2 Target size

The principle to derive the target size of decimation is similar to the stream simplification (Wu and Kobbelt 2003), which is to keep the decimation rate of the intermediate iso-surface equal to R after the decimation. Because it is explicitly known which area of the iso-surface is modifiable, only the decimation rate of the modifiable area is considered. Given the count of totally extracted faces F_g and the amount of uncontractible faces F_{u}, before the decimation of each time, the target face count F_{t} is set to:

$$F_{\mathrm{t}} = (F_{\mathrm{g}} - F_{\mathrm{u}}) \cdot R + F_{\mathrm{u}}.$$

We used the method similar to the stream simplification (Wu and Kobbelt 2003) to avoid the possible over decimation in the first several times of decimation. If the given decimation rate R is lower than an initial decimation rate threshold R_{i}, our algorithm first enters an initial phase. In the initial phase, before decimating the iso-surface, the face extraction process generates faces until their amount reaches B. The iso-surface is then decimated to $(B - F_{\mathrm{u}}) \cdot R_{\mathrm{i}} + F_{\mathrm{u}}$ faces. Because the amount of the extracted faces increases and the size of the simplified intermediate iso-surface stays approximately the same after each decimation, the real decimation rate of the intermediate iso-surface decreases gradually. The initial phase lasts until the real decimation rate is close to R.

4.3 Memory efficiency

The maximal main memory consumption is determined by the maximal size of the intermediate iso-surface kept in memory, plus the two slices of the volume data loaded each time. The values of R and B together

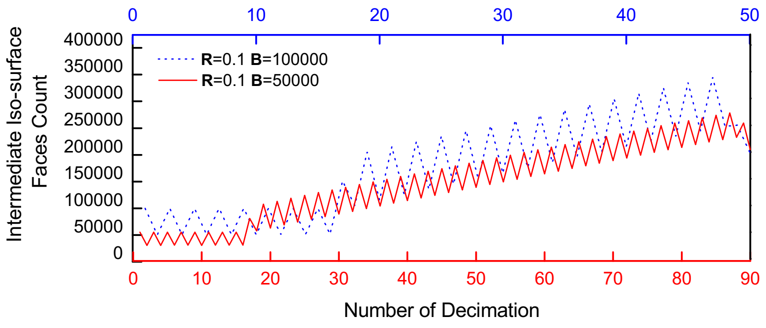

Fig. 5 Intermediate iso-surface size growth before and after each decimation for bunny dataset with R = 0.1, B = 50,000 and 100,000. Decimations are numbered by the sequence to which they take place. The early decimations without size growing indicate the initial phase

Table 2 Some small datasets that are tested

Name	Resolution	iso	v	f
Bunny	512 × 512 × 361	2000	1,024,433	2,048,290
Head	500 × 500 × 476	100.5	2,400,610	4,786,342
Foot	256 × 256 × 256	127.5	294,013	586,220
Teapot	256 × 256 × 178	60	254,629	509,300

dictate the intermediate iso-surface size for a given dataset while R determines the slope of the size growing and B determines the amplitude the curve vibrates. Figure 5 illustrates the intermediate iso-surface size growth for the bunny dataset in Table 2 with the same value for R and different values for B. If we do not count the uncontractible faces, the upper bound for the face count of the intermediate iso-surface is $RF + B$ (F denotes the face count of the fully extracted iso-surface). Because the unmodifiable vertices approximately come from a slice of the volume data, the amount of uncontractible faces has a limit. From Fig. 9 of Sect. 5 we can see that the uncontractible faces only cover a small portion of the whole generated iso-surface. Therefore, for most datasets, they can be ignored.

5 Results

To evaluate the performance and simplification quality of our on-the-fly algorithm, we implemented a cutting-based out-of-core mesh simplification algorithm and an out-of-core indexed-format iso-surface extraction algorithm. The cutting-based out-of-core simplification partitions the mesh by vertex similar to (Brodsky and Pedersen 2003). The out-of-core extraction utilizes our framework based on the per-cube vertex modifiability detection and demonstrates another usage of the framework.

We compared the results of our on-the-fly algorithm with the cutting-based out-of-core algorithm and the QSlim (Garland and Heckbert 1997) algorithm on various datasets. All tests were run on a commodity portable notebook with 1.70GHz dual-core Intel i5-3317U CPU, 4 GB RAM, and 5400 RPM disk.

We used metro (Cignoni et al. 1998) tool to evaluate the approximation error of the simplified iso-surface. The measured error in our test was the RMS error of metro tool normalized by the diagonal of the bounding box, while sampling the original mesh onto the simplified mesh.

Because metro tool and QSlim only accept in-core mesh, we ran tests using the three simplification algorithms on the small datasets shown in Table 2. The first four rows of Table 3 present the test results on these datasets with decimation rate 0.2. In these tests, we set B to 10,000 for our on-the-fly simplification. The measured error in Table 3 demonstrates that the simplification qualities of the three algorithms on small datasets were quite similar under such moderate decimation rate. Figure 6 shows the similarity of the simplified iso-surfaces for the foot dataset. Note that for most of the small datasets, the QSlim algorithm spent approximately the same time with our on-the-fly algorithm. This may be caused by the fact that QSlim algorithm uses regular array as the vertex and face container while our algorithm uses hash table. However, our algorithm is still faster than the cascade of out-of-core algorithms.

Table 3 Test results on different data resets. The out-of-core and QSlim algorithms share the extraction process and have identical mesh generation time

Dataset	Algorithm	v	f	Gentime (s)	Simptime (s)	Totaltime (s)	Error (%)	Mem (mb)
Bunny	Ours	205,046	409,658	–	–	23.82	0.00501	171
	Out-of-core	205,050	409,659	10.82	21.55	32.37	0.00493	140
	QSlim	205,048	409,659		12.75	23.57	0.00492	636
Head	Ours	482,994	957,267	–	–	51.95	0.0650	386
	Out-of-core	483,757	957,266	20.16	52.01	72.17	0.0708	328
	QSlim	483,752	957,268		32.24	52.40	0.0703	1487
Foot	Ours	59,225	117,244	–	–	8.68	0.0876	54
	Out-of-core	59,340	117,244	4.22	6.12	10.34	0.0864	58
	QSlim	59,319	117,245		3.49	7.71	0.0863	185
Teapot	Ours	50,892	101,860	–	–	9.34	0.00523	48
	Out-of-core	50,890	101,859	4.95	5.23	10.18	0.00661	37
	QSlim	50,892	101,861		2.93	7.88	0.00497	160
Bunny*	Ours	4166	8041	–	–	9941.96	–	835
	Out-of-core	4167	8041	8633.76	2559.41	11193.17	–	1762
Head*	Ours	58,152	100,034	–	–	2189.54	–	244
	Out-of-core	58,366	100,036	1509.30	1050.54	2559.84	–	1495

* Upsampled datasets compared with the original sets

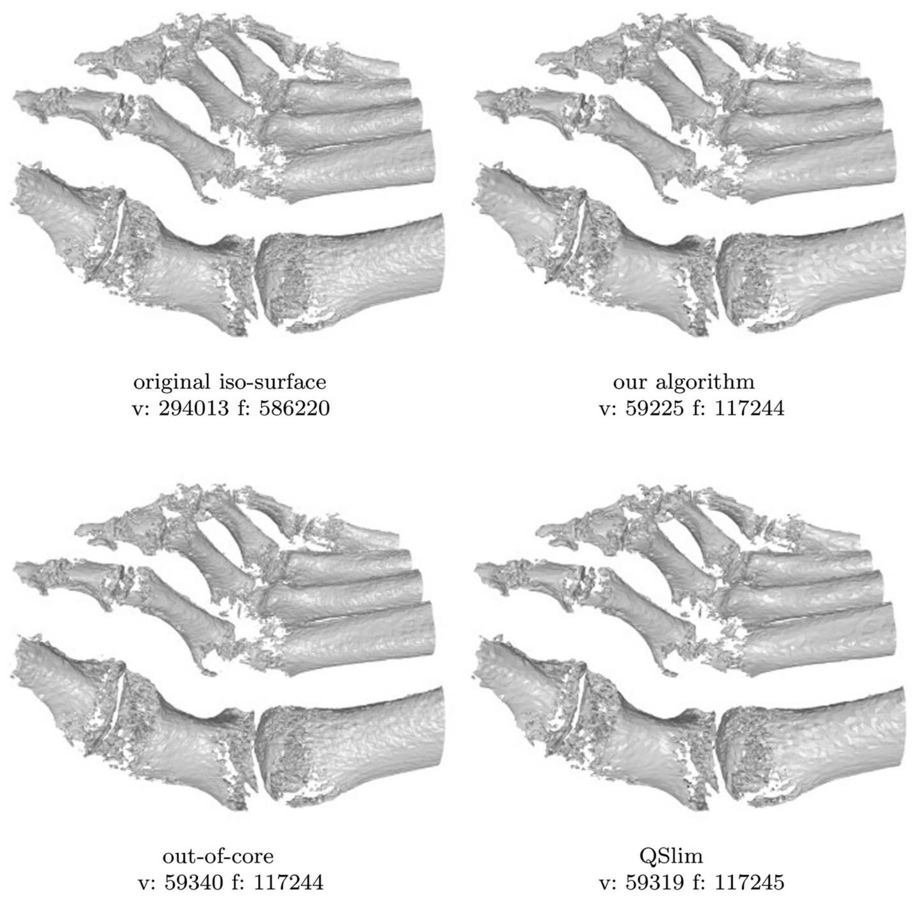

Fig. 6 Rendering results of the iso-surfaces extracted and simplified from the foot dataset using different algorithms with decimation rate 0.2. Because the decimation rate is rather moderate, the simplification quality of the three algorithms is similarly high, the rendered simplified iso-surfaces show little visual difference

Figure 7 gives the RMS error and wall clock time of the on-the-fly simplification on bunny dataset with $R = 0.1$ and 0.01, using different face increase (B). We found that when the decimation rate was rather low (0.1), the value of B made no notable impact on the simplification quality. However, when the decimation

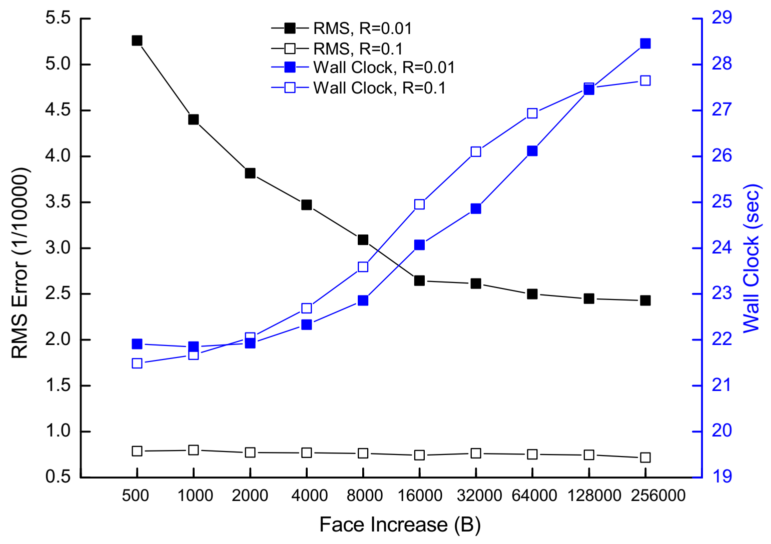

Fig. 7 The RMS error and wall clock time of the on-the-fly simplification on bunny dataset with $R = 0.1$ and 0.01 using different face increase (B)

rate was high (0.01), the impact of B's value became great. Based on our test, in order to retrieve a simplification quality comparable to in-core simplifications, one usually has to set a B value higher than the simplified iso-surface size. Using such a B value means that the algorithm will never jump out of the initial phase. When the decimation rate is high, the size of the simplified iso-surface is rather small; such a B value keeps the memory footprint low.

We used trilinear interpolation to create large out-of-core datasets from some of our in-core sets. The interpolated out-of-core datasets have the same effects with normal large datasets on performance evaluation. We also rendered the iso-surfaces extracted from some small interpolated datasets and found that these iso-surfaces had no notable visual difference from the original ones. They were mostly quite smooth with denser triangle tessellation. Therefore, the visual effects of the iso-surfaces extracted and simplified from these interpolated datasets demonstrate the simplification quality of our algorithm on large datasets. Table 5 shows the details of these interpolated large sets and the last two rows of Table 3 give their test results. We set B to 2,000,000 for the bunny* dataset and 500,000 for the head* dataset when running our on-the-fly algorithm. To better illustrate the simplification quality of our algorithm, we used a computer with 64 GB memory to run QSlim on the interpolated datasets. The rendering result is shown in Fig. 8. It should be noted that, though in-core simplification can be performed on a computer with such large memory, most regular users can only afford commodity PCs which are similar to the one that we perform most of our tests. Our algorithm is targeted to scenarios where the computer has just regular-size memory.

From the results on these interpolated large datasets, we can see a significant difference between our algorithm and the cascading of out-of-core extraction and simplification. In our implementation, the cutting-based simplification uses a simple spatial division to segment original mesh into pieces. Larger slices on the axes create smaller pieces, which leads to lower memory consumption and less execution time. However, it may also lead to poorer simplification quality. Hence, choosing the division of axes is more of a tradeoff between execution cost and quality. For the cutting-based algorithm, we allocate the target vertex count of decimation for each piece based on its vertex count of the original iso-surface. This fast-deriving approach may fail on some large iso-surfaces. On the other hand, our algorithm utilizes a global priority queue to preserve the overall decimation distribution over the iso-surface. As long as B's value is carefully chosen, the simplified iso-surfaces produced by our algorithm would have a more balanced tessellation over regions with different grades of details. This could be illustrated by Fig. 8. For the head* set, since the target vertex count is not small enough to make poorly simplified iso-surface occur, both algorithms produce high-quality simplified iso-surfaces. However, there are still some regions where the out-of-core algorithm produces unbalanced simplified triangles while our algorithm preserves the overall detail distribution. The lowest row of Fig. 8 illustrates one of these regions. From the figure, we can see that part of the inner torus of the head*

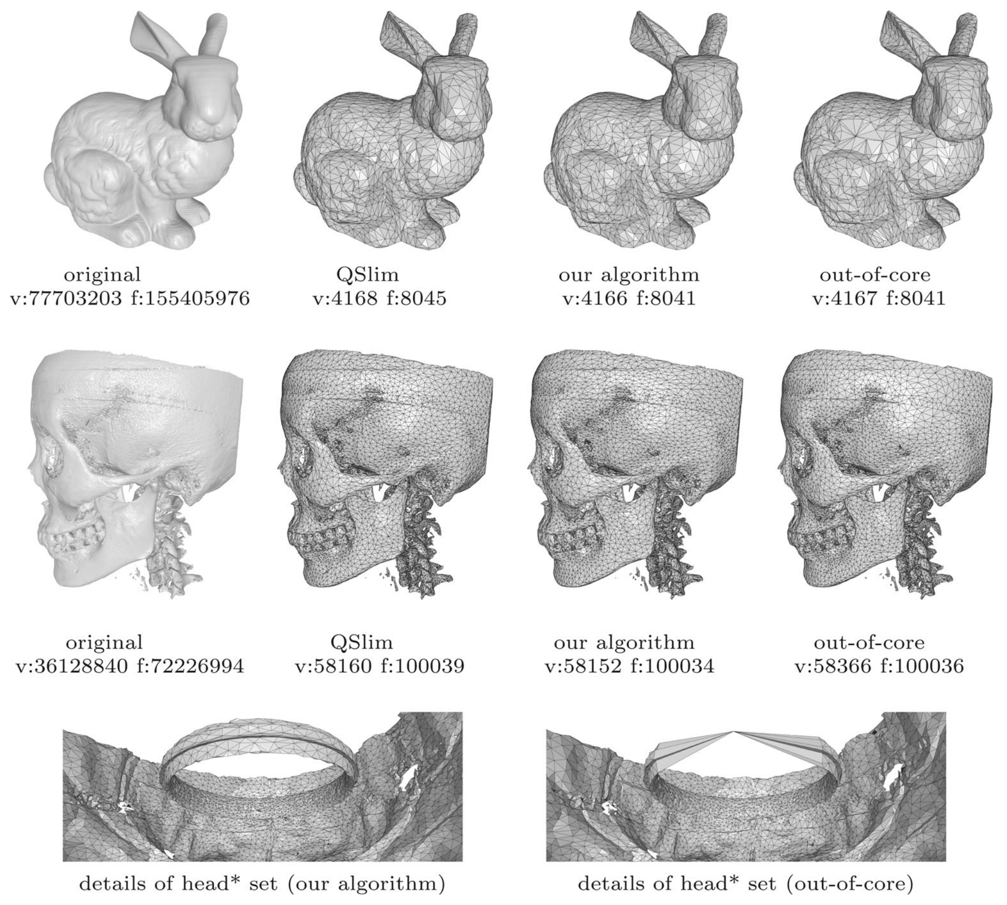

Fig. 8 Rendering results of bunny* and head* iso-surfaces extracted and simplified using our on-the-fly algorithm, the out-of-core algorithms, and the QSlim algorithm (on a computer with 64 GB memory). Our algorithm processes the dataset as a whole and produces a more balanced tessellation than the out-of-core simplification which decimates the mesh pieces separately

Table 4 Execution time measured by separating the extraction, simplification, and I/O operations

Dataset	Algorithm	Gentime(s)	Simptime(s)	Iotime(s)
Bunny	Ours	10.86	13.72	0.02
	Out-of-core	10.78	13.11	8.65
Head	Ours	20.32	32.48	0.04
	Out-of-core	19.92	32.02	20.42
Bunny*	Ours	8482.63	1789.11	2.87
	Out-of-core	8112.97	1543.90	849.79
Head*	Ours	1503.83	719.65	0.37
	Out-of-core	1453.65	650.79	332.41

* Upsampled datasets compared with the original sets

set is oversimplified by the cutting-based algorithm. The reason may be that, the piece of mesh segmented by the algorithm containing that region has too few triangles, which leads to the number of decimated vertices allocated to be small. However, the low target vertex count does not match the surface complexity (which is, on the contrary, high). As a result, oversimplified regions are produced. The root cause for this is the break of global decimation sequence introduced by the mesh segmentation. For the bunny* set, because the decimation rate is high considering the size of the dataset, the difference of simplification quality of the two algorithms is obvious. The bunny* iso-surface simplified by the cutting-based algorithm has quite poor quality, with over decimated triangles in the central area, while the simplification quality of our algorithm is still satisfying. Another thing to mention is that, for both of the two datasets, our algorithm produces simplified iso-surfaces comparable to the QSlim algorithm. Though producing better simplification quality, our algorithm consumes much less memory than the out-of-core algorithm based on Table 3.

Table 5 Details of the interpolated datasets

Name	Orig res	Interp res	iso	v	f
Bunny[*]	512 × 512 × 361	4089 × 4089 × 3961	2000	77.703.203	155,405,976
Head[*]	500 × 500 × 476	1997 × 1997 × 1901	100.4	36,128,840	72,226,994

[*] Upsampled datasets compared with the original sets

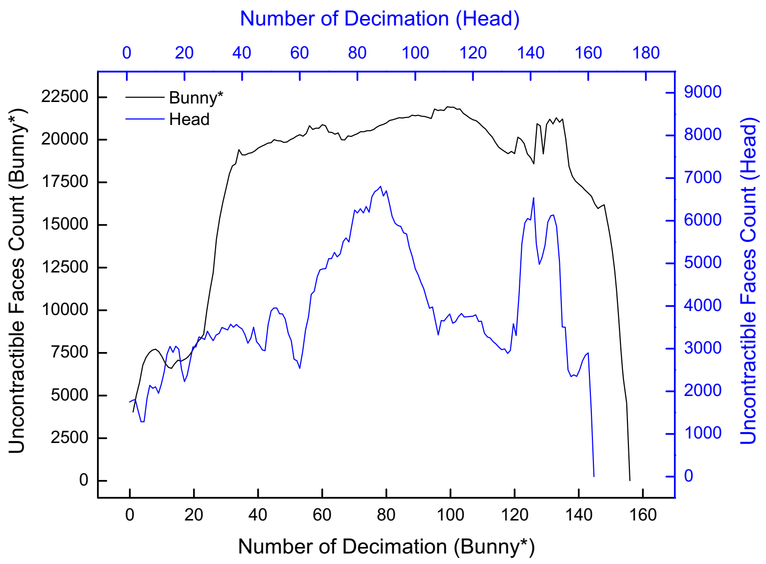

Fig. 9 Count of uncontractible faces before each decimation when processing the head and bunny* dataset using our on-the-fly simplification, with $R = 0.2$ and $B = 30,000$ for head, $R = 0.00005174$ and $B = 2,000,000$ for bunny*. Decimations are numbered by the sequence to which they take place

To better illustrate the speedup in I/O cost of our algorithm with respect to the out-of-core algorithm, we measured separately the extraction, simplification, and I/O time cost on some datasets for the two algorithms in Table 4. It should be noted that since these three kinds of operations are scattered in each iteration of the two algorithms, in order to conduct the measurement, we had to query the start and end time at different portions of the source codes in each iteration. This should have some impacts on the overall execution time. On the other hand, the measurement method in Table 3 only makes system time calls at some critical points of execution (like the start and end of a program), so that the time measured should be treated as a more precise approximation to the execution time. However, the results in Table 4 still provide a reliable reference for time cost spent in different operations. From the table, we can see that the I/O time cost of our algorithm is much less than the out-of-core algorithm. Because the major part of time is consumed by simplification and extraction, the speedup of our algorithm is quite considerable (Table 5).

Figure 9 shows the count of uncontractible faces before each decimation when processing the head and bunny* dataset using our on-the-fly simplification. Note that for the two datasets, the maximal ratio with respect to all extracted faces is 0.14 %. This also proves the low-footprint feature of our algorithm.

6 Discussion

One inconvenience of our on-the-fly algorithm implementation is that, user has to run the whole on-the-fly extraction and simplification process once some parameters for the algorithm are changed, even though the iso-value stays the same. In contrast, using the cascading of out-of-core algorithms, if the iso-value of interest is the same, one can perform the out-of-core extraction once and has different parameters for the simplification using the same generated mesh. A solution for this is to store the iso-surface generated by our extraction and reconstruction approach, with a special mesh format interleaving vertices and faces and explicitly claiming finalization of vertices similar to the streaming meshes (Isenburg and Lindstrom 2005). A stream simplification can then be performed utilizing the finalization information to determine vertex modifiability.

7 Conclusion

We presented an on-the-fly simplification algorithm for iso-surfaces extracted from large volume sets. It processes the dataset only once, runs fast, and has low memory footprint. The simplification quality of our algorithm is as good as some high-quality out-of-core simplification algorithm and also comparable to in-core algorithm. We tested the performance of our algorithm by comparing it with the widely used in-core algorithm QSlim and an out-of-core simplification algorithm on various datasets. The key of our algorithm is the per-cube vertex modifiability detection, which provides a general solution for the on-the-fly processing of large iso-surface.

Acknowledgments This work was supported in part by the National Natural Science Foundation of China (61272225, 61572274, 91515103, 51261120376). We would like to thank the Stanford Computer Graphics Laboratory for the bunny dataset, http://www.volvis.org web site for the foot and teapot datasets, and Michael Garland for providing the QSlim implementation.

References

Akio D, Koide A (1991) An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. IEICE Trans Inf Syst 74(1):214–224

Attali D, Cohen-Steiner D, Edelsbrunner H (2005) Extraction and simplification of iso-surfaces in tandem. In: Eurographics symposium on geometry processing 2005, The Eurographics Association, pp 139–148

Bernardini F, Rushmeier H, Martin IM, Mittleman J, Taubin G (2002) Building a digital model of michelangelo's florentine pieta. Comput Gr Appl IEEE 22(1):59–67

Brodsky D, Pedersen JB, (2003) A parallel framework for simplification of massive meshes. In: Parallel and large-data visualization and graphics (2003) PVG 2003. IEEE symposium on, IEEE, pp 17–24

Cignoni P, Rocchini C, Scopigno R (1998) Metro: measuring error on simplified surfaces. Comput Gr Forum, Wiley Online Libr 17:167–174

Cignoni P, Montani C, Rocchini C, Scopigno R (2003) External memory management and simplification of huge meshes. Vis Comput Gr IEEE Trans 9(4):525–537

Garland M, Heckbert PS (1997) Surface simplification using quadric error metrics. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., pp 209–216

Hoppe H (1996) Progressive meshes. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, ACM, pp 99–108

Isenburg M, Lindstrom P (2005) Streaming meshes. In: Visualization (2005) VIS 05. IEEE, IEEE, pp 231–238

Isenburg M, Lindstrom P, Gumhold S, Snoeyink J (2003) Large mesh simplification using processing sequences. In: Proceedings of the 14th IEEE visualization 2003 (VIS'03), IEEE computer society, p 61

Ju T, Losasso F, Schaefer S, Warren J (2002) Dual contouring of hermite data. In: ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002, ACM, NY, USA, vol 21, pp 339–346

Lindstrom P (2000) Out-of-core simplification of large polygonal models. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., pp 259–262

Lindstrom P, Silva CT (2001) A memory insensitive technique for large model simplification. In: Proceedings of the conference on visualization'01, IEEE computer society, pp 121–126

Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. ACM Siggr Comput Gr ACM 21:163–169

Manson J, Schaefer S (2010) Isosurfaces over simplicial partitions of multiresolution grids. Comput Gr Forum, Wiley Online Libr 29:377–385

Newman TS, Yi H (2006) A survey of the marching cubes algorithm. Comput Gr 30(5):854–879

Nielson GM, Hamann B (1991) The asymptotic decider: resolving the ambiguity in marching cubes. In: Proceedings of the 2nd conference on visualization'91, IEEE Computer Society Press, pp 83–91

Peng Y, Chen L, Yong JH (2014) Importance-driven isosurface decimation for visualization of large simulation data based on openCL. Comput Sci Eng 16(1):24–32

Prince C (2000) Progressive meshes for large models of arbitrary topology. Master's thesis, University of Washington

Rossignac J, Borrel P (1993) Multi-resolution 3D approximations for rendering complex scenes. Springer, Berlin

Schaefer S, Warren J (2005) Dual marching cubes: primal contouring of dual grids. Comput Gr Forum, Wiley Online Libr 24:195–201

Schroeder WJ, Zarge JA, Lorensen WE (1992) Decimation of triangle meshes. ACM Siggr Comput Gr ACM 26:65–70

Shirley P, Tuchman A (1990) A polygonal approximation to direct scalar volume rendering. In: Proceedings of the 1990 Workshop on Volume Visualization, ACM, NY, USA, vol 24, pp 63–70. http://doi.acm.org/10.1145/99307.99322

Van Gelder A, Wilhelms J (1994) Topological considerations in isosurface generation. ACM Trans Gr (TOG) 13(4):337–375

Wu J, Kobbelt L (2003) A stream algorithm for the decimation of massive meshes. Gr Interface 3:185–192