Importance-driven isosurface decimation for visualization of large simulation data based on OpenCL

Yi Peng, Li Chen, Jun-Hai Yong

Importance-Driven Isosurface Decimation for Visualization of Large Simulation Data Based on OpenCL

Yi Peng, Li Chen, and Jun-Hai Yong

Abstract—For large simulation data, Parallel Marching Cubes algorithm is efficient and commonly used to extract isosurfaces in 3D scalar field. However, the isosurface meshes are sometimes too dense and it is difficult for scientists to specify the areas they are interested in. In this paper, we provide them a new way to define mesh importance for decimation using transfer functions and visualize large simulation data in case the normal visualization methods cannot handle due to memory limit. We also introduce a parallel isosurface simplification framework which uses pyramid peeling to extract the decimated meshes progressively without generating the original surface. Since the implementation uses OpenCL which is oriented to heterogeneous computing, our method can be applied to different parallel systems and scientists can see the visualization results while doing simulations. Finally, we evaluate the performances of our algorithm and use different scientific datasets to show the efficiency of our method.

Index Terms—Isosurface Visualization, Importance Driven, Transfer Function, Marching Cubes, Mesh Simplification, OpenCL

1 Introduction

Marching Cubes is widely used in isosurface visualization for large simulation data. Since for each cube, the computation is independent, the algorithm can be parallelized efficiently, which means scientists can use this method to analyze the results while doing simulation. However, since for one dataset domain experts often use different visualization methods, such as slicing, cutting plane, volume rendering and so on, the original volumetric data needs to be stored in memory to keep high performance, which means for isosurface rendering the meshes require additional memory. That is to say for large dataset the memory consuming should be considered, so it is quite necessary to do mesh decimation. Even though there are many mesh simplification methods such as edge collapse, vertex removal and polygon merging [1] which can preserve geometric features. It is still difficult for users to define mesh importance according to their interests. In order to solve this problem, we propose an importance-driven mesh decimation strategy, which provides users an interactive way to specify the importance of meshes according to features using transfer functions and allow them to directly see the feature distributions and evolutions on the decimated surfaces. The most important contributions we have made are listed below:

1) Using transfer functions to define the importance of meshes for isosurface generation interactively;
2) Simplifying isosurface before it is generated;
3) The entire algorithm is parallelized based on OpenCL which includes feature extraction, mesh generation and model rendering;

Our method consists of three parts. Data

- *Y. Peng (15pengyi@gmail.com) is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China. http://cgcad.thss.tsinghua.edu.cn/jackiepang/*
- *L. Chen (chenlee@tsinghua.edu.cn) and J.H. Yong (yongjh@tsinghua.edu.cn) are with School of Software, Tsinghua University.*

Pre-processing classifies the cubes and extracts user-specified features. Pyramid Construction builds the histograms and feature pyramids which contain decimation information. Pyramid Peeling generates and renders the decimated meshes in a top-down way.

These meshes are generated while being patched and refined. We also test the performance of our algorithm and use several datasets including medical datadsets and scientific datasets to evaluate our method. The results show that our method is efficient and interactive.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 defines the mesh importance which is used as the decimation criteria. Section 4 gives a detail introduction of our algorithm which is highly parallelized. Section 5 shows the results of our importance-driven decimation strategy based on several datasets. Section 6 summarizes the advantages of our method and outlines the future work.

2 Related Work

The research related to our method can be divided into three groups. First, our idea comes from importance-driven visualization method and this work is based on some latest research in Parallel Marching Cubes. We also benefits from traditional mesh simplification methods.

2.1 Importance-Driven Visualization Method

In scientific visualization, there are two ways to define data importance. The feature-based methods need to extract features from the original data while the visibility-based methods pay attention to user perceptions.

For feature-based methods, researchers focus on texture, size, importance curves for feature-temporal analysis [2], shape [3] and so on. They also do mesh deformation to reduce volumetric data based on voxel importance, which provides users a focus + context visualization [4].

From the point of users, visibility is introduced by Bordoloi et al. [5]. Further research computes the visibility histograms [6] and informational divergence [7] which is used to maximize the user-defined importance.

In our method, we use geometric features which are extracted from the original data and allow users to specify mesh importance using transfer functions. So users can interactively define the mesh importance and directly see the decimated meshes and the feature distributions on that isosurface.

2.2 Marching Cubes on GPU

Marching Cubes is introduced by Lorensen et al. [8] and there are two different ways to parallelize it. On one hand, Goetz et al. [9] extends Marching Cubes using vertex shader which is compatible to OpenGL fix function. After that, the work is improved by introducing preclassification and histogram pyramid. On the other hand, Smistad [10] implements the original algorithm in OpenCL, so that the algorithm runs entirely on the GPU, which means their work is oriented to general computing.

Our work is based on OpenCL since we want to unify the computing process and rendering process. So our visualization techniques can be easily integrated into different research fields.

2.3 Mesh Simplification

For traditional mesh simplification methods, incremental decimation methods such as edge collapse, vertex collapse, vertex removal and polygon merging are difficult for parallelism due to data sorting [1]. Only vertex clustering can be easily parallelized [11].

For marching cubes surfaces, early work reads adjacent slices of volumetric data and merges the mesh vertices or uses octree to simplify the main continuous surface [12]. Attali et al. [13] introduce tandem algorithm which extends the Marching Cubes by sweeping the dataset and extracting meshes on the accumulated partial surface. Dupuy et al. [14] extend this algorithm to a parallel system. However, these methods are hard to be implemented on GPU since they are streamed.

In this paper, we introduce a parallel decimation method based on mesh patching and refining, which benefits from crack patching [12] and vertex clustering.

3 Importance Definition

Before mesh simplification, we use both geometric properties and user-specified features to define mesh importance. The geometric properties are mainly based on gradient which is also used to control the quality of meshes. The user-specified features can interactively be changed through transfer function editor and be applied to the decimation process, so that the generated meshes are important to users.

3.1 Geometric Properties

For isosurface, three most important geometric properties are continuity, smoothness and topology. In this paper, we consider continuity and smoothness because the topology is changed during simplification.

Since Marching Cubes algorithm guarantees mesh quality for a same resolution, the main challenge is to maintain the continuity between two different resolutions. We follow the idea of [12] which performs crack patching in a top-down way. During this process, we use a histogram pyramid(see [10]) like an octree to record the patching information, so the finer meshes generated by a child cube are consistent to the coarser ones correspond to its parent cube. The parent-children relationship of cubes is defined by their relationship in the octree, which means a parent cube has 8 child cubes at the next resolution(finer resolution).

On the other hand, we choose the smoothness of gradient as one of the decimation criteria. For each cube c, we first compute its gradient using central difference and then calculate the smoothness feature s_c according to the gradients of its 8 child cubes as follow:

$$s_c^{(l)} = 1 - \left(1 + \frac{\sum_{i=1}^{N} n_{c_i} \left| \mathbf{g}_{c_i}^{(l-1)} \right|^2}{\sum_{i=1}^{N} n_{c_i}} - \left| \frac{\sum_{i=1}^{N} n_{c_i} \mathbf{g}_{c_i}^{(l-1)}}{\sum_{i=1}^{N} n_{c_i}} \right|^2 \right)^{-1} \tag{1}$$

where for volumetric data, $N = 8$ which is the total number of child cubes of a parent cube, l is the resolution level, $\mathbf{g}_c$ denotes normalized gradient vector and and n_{c_i} denotes the number of meshes generated by cube c. In implementation, for each cube we only store two gradient moments and one mesh number. So the cubes with lower smoothness are more likely to be decimated.

3.2 User-Specified Features

For user-defined features such as texture, pressure, velocity and so on, we first construct a feature pyramid in a same way as smoothness feature:

$$\mathbf{f}_c^{(l)} = \frac{\sum_{i=1}^{N} n_{c_i} \mathbf{f}_{c_i}^{(l-1)}}{\sum_{i=1}^{N} n_{c_i}} \tag{2}$$

where $N = 8$ see Equation. 1, $\mathbf{f}_c$ denotes the features which are domain-specific derivatives.

After that the features are mapped to importance values by transfer functions. So users can define mesh importance according to the features they are interested in interactively. In this paper, we provide users a one-dimensional transfer function editor to design the mapping functions as shown in Fig.1. However the mapping can easily be extended to multi-dimensional, if different domains are considered. In this transfer function editor, users can edit the control points to generate the mapping functions. For each control point, the horizontal coordinate indicates the corresponding feature which is obtained by blending geometric feature with user-specified features. The vertical coordinate and user-specified color denotes the importance and color of that feature respectively.

The advantages of using transfer functions to define mesh importance are obvious. First, this kind of interactions are widely used in scientific visualization. Also, it is easy to learn. For one-dimensional transfer function, users only need to add(delete) control points or just modify their positions and colors. Then the system will generate the mapping functions automatically. More importantly, the background histograms of features help users design the mapping functions, since they connect feature with color and mesh importance.

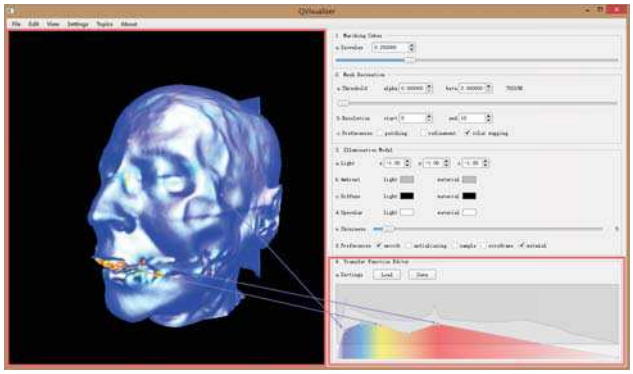


Fig. 1. One-dimensional transfer function editor which is used to specify mesh importance. The height of a control point defines the feature importance while its color corresponds to the color mapping functions used in surface rendering. The background curves show the distributions of data and user-specified feature.

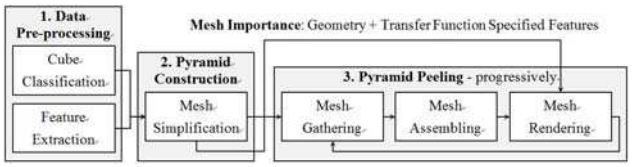


Fig. 2. Parallel Marching Cubes surfaces decimation framework. The framework includes three parts: pyramid initialization, construction and peeling. In the first stage, the cubes are classified and features including geometric properties and user-specified features are extracted. In the next construction stage, the isosurface meshes are generated while decimated, and in the last stage, meshes with different resolutions are rendered top-down progressively.

3.3 Mesh Importance

The final importance of a mesh combines the geometric properties and user-defined features. For each cube c, the importance I_c is defined as follow:

$$I_c^{(l)} = t\left((1-\alpha)\, s_c{}^{(l)} + \alpha \cdot |\mathbf{f}_c{}^{(l)}|\right) \tag{3}$$

where α is a blend factor and t is the transfer function specified by user. And in the next stage, the importance is used to decimate the meshes.

4 Mesh Simplification

Based on [10]'s work, our mesh simplification method is highly parallelized. Also we have made several improvements on it, such as lower memory cost and progressive rendering support.

4.1 Simplification Framework

The framework of our parallel simplification algorithm includes 3 stages as shown in Fig.2. All these stages are carried out in GPU.

The first stage computes both the geometric and user-defined features. After that, the construction stage build histogram and feature pyramids while recording the decimation information. The final stage performs a top-down mesh peeling. In the last stage, the meshes are rendered shortly after they have been assembled which means they are generated progressively. So that users can choose which resolution should be rendered. The advantages of decimating meshes when generating are less memory cost since we only need to store decimation information and final meshes instead of the full resolution meshes, faster rendering due to less meshes and progressive exploration.

4.2 Decimation Criteria

With mesh importance the decimation criteria can be defined as follow:

$$d_c^{(l)} = \begin{cases} 1 & I_c^{(l)} \le \beta \\ 0 & I_c^{(l)} > \beta \end{cases} \tag{4}$$

where β is a threshold used to control the degree of mesh decimation. When a parent cube is decimated, the same operation should be done for all the descendants. However, since for each resolution we need to compute the number of meshes, which means we will do a bottom-up scan, so we only need to record the decimation information while scanning the pyramid.

4.3 Data Pre-processing

For cube classification, we use the infrastructures provided by [10]. Although it leads to

topological ambiguity, we find that the performances are better to be measured when using the same look up table. Meanwhile, we also compute the gradient feature and extract user-defined features during the same time.

4.4 Pyramid Construction

In this stage, histogram and feature pyramids are constructed in a bottom-up way, so that cubes with higher resolution are processed earlier. For each resolution, we use smoothness and user-define features to compute the importance of cubes. After that, the importance is used to decide whether this cube should be decimated. If it is true, the meshes belong to this cube will be updated, which means this cube will be chosen as a representative of all its descendants.

4.5 Pyramid Peeling

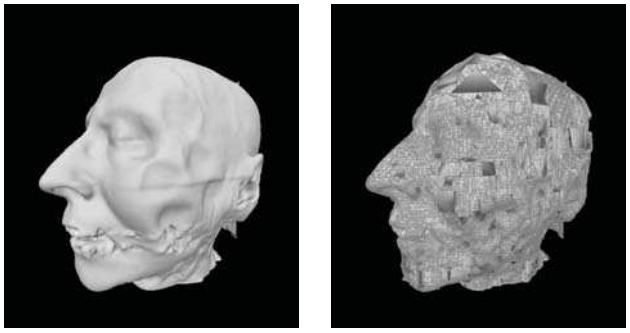
The last stage generates the meshes progressively, so the coarsest meshes are rendered first. For each resolution there is a corresponding peeling process which consists of three steps. The gathering step traverses the histogram pyramid to get the total number of meshes. The assembling step computes mesh positions, normals and color attributes. The last step just renders all the meshes. In the second step, there are three key techniques which are used to improve mesh quality.

4.5.1 Mesh Refinement

For meshes in different resolutions, cracks may be generated due to discontinuous meshes as shown in Fig.3.

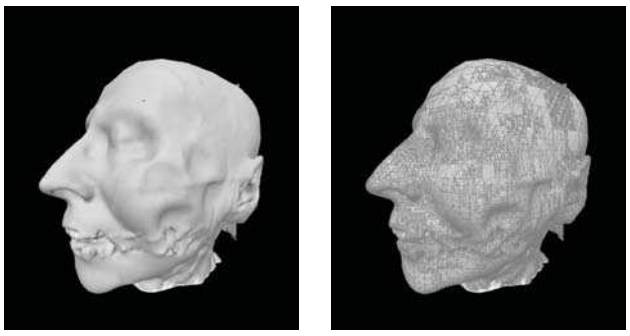
Since coarse meshes are prominent, we first improve their qualities by refining them according to detail information. That is to say, we need to re-compute mesh positions, normal vectors using the information of their descendants. In implementation, we use binary search to calculate the finest value in the full resolution.

Fig.4 shows the results of mesh refinement. Comparing Fig.4(a) with the original results shown in Fig.3(a), the results are not that different.



(a) original data (b) generated cracks

Fig. 3. Cracks generated by Marching Cubes Algorithm. (a) shows the original isosurface. (b) shows the meshes in different resolutions.



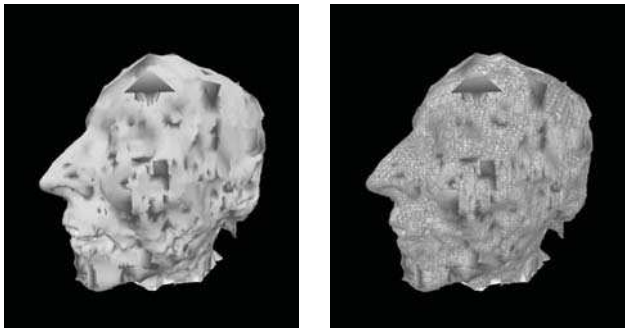
(a) refining results (b) refining results with wireframes

Fig. 4. Mesh refining results. (a) shows the results of mesh refinement technique. (b) draws the wireframes of the same model.

4.5.2 Crack Patching

In order to patch the cracks, we re-compute the finer meshes according their neighboring ancestors, which means for all the vertices on a finer mesh, we compute their new properties. For a vertex, we first find its 8 neighboring cubes. Then all of their ancestors will be checked in a top-down way, so coarser cracks are patched first. If there are several ancestors which may cause cracks, we choose one at the coarsest resolution. After that, we project this vertex to the edges generated by that ancestor. If there are several edges which are coplanar with this vertex, we choose one with the shortest distance. Fig.5 shows the results of crack patching.

From Fig.5 we can see that, the crack patching technique gets remarkable results, since the boundary consistency of meshes in different



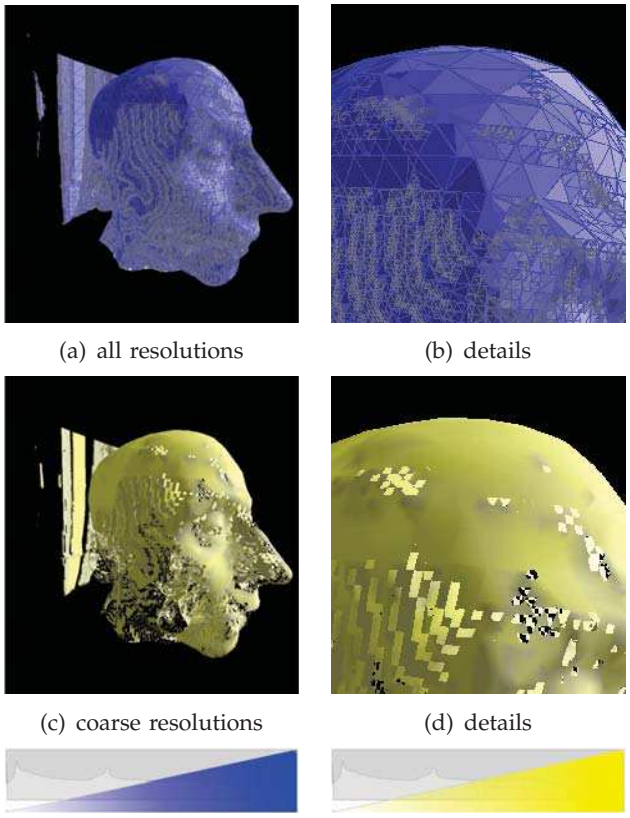
(a) patching results (b) patching results with wireframes

Fig. 5. Crack patching results. (a) shows the results of crack patching technique. (b) draws the wireframes of the same model.

resolutions is guaranteed due to the uniqueness of vertex projection, so the meshes are more continuous compared with Fig.3(b).

4.5.3 Progressive Rendering

We also render the meshes progressively. So users can choose which resolution to be rendered as shown in Fig.6.



(a) all resolutions (b) details

(c) coarse resolutions (d) details

(e) color transfer function - a (f) color transfer function - b

Fig. 6. Progressive rendering results. (a) shows the meshes of all resolutions, in which higher lightness values are set to coarser meshes. (c) is their corresponding transfer functions. (b) shows the meshes at coarse resolutions and (d) is their transfer functions.

5 Results

In this section, we analyze the performance and memory cost of our algorithm and use several datasets to test our method.

5.1 Algorithm Performance

The performance of our method is shown in TABLE1. From this table, we can see that the time cost is strongly related to the number of decimated meshes. Larger decimation threshold leads to better performance. These tests are done on a standard PC: CPU Intel(R) Core i7-2670QM @ 2.20GHz, RAM 2 * 4.00GB @ 1333MHz, GPU Nvidia GeForce GT 555M @ 2GB and OS Windos 7 Ultimate SP1.

TABLE2 shows the memory costs of our method and Smistad's method [10]. Compared with their method, we use buffer objects instead of 3D textures and support progressive and multi-pass rendering which can deal with large data. Since their method relies on the maximum size of the data, they can not handle inhomogeneous data. Even though we need addition space to record decimation information and features, since our pyramid only relies on the total size of the data, we can still handle large datasets. If we only consider geometric property, the feature size will be four times the data size(three for gradient and one for mag-

TABLE 1
Algorithm Performance

Dataset	Parameters			Mesh Number			Data Pre-	Pyramid	Pyramid	Total
Name	Iso	α	β	Original	Decimated	%	processing	Construction	Peeling	Time
	0.25	0.00	0.0035	703196	175122	24.9	40ms	41ms	115ms	196ms
Head	0.25	0.00	0.0130	703196	58868	8.37	39ms	41ms	63ms	143ms
256x256x225	0.38	0.00	0.0040	827884	180752	21.83	40ms	41ms	97ms	178ms
	0.38	0.00	0.0120	827884	63606	7.68	39ms	40ms	45ms	124ms
	0.25	0.00	0.0015	612936	131598	21.47	35ms	36ms	67ms	138ms
Hurricane(18^{th})	0.25	0.00	0.0055	612936	44860	7.32	35ms	35ms	44ms	114ms
661x401x42	0.25	0.35	0.0225	612936	118344	19.31	36ms	35ms	59ms	130ms
	0.25	0.35	0.0240	612936	53498	8.73	36ms	35ms	38ms	109ms
	0.50	0.00	0.0150	460062	107732	23.42	103ms	83ms	136ms	322ms
Fluid Data(100^{th})	0.50	0.00	0.0350	460062	61096	13.28	103ms	82ms	80ms	265ms
2048x56x239	0.50	0.50	0.0400	460062	140120	30.46	104ms	83ms	93ms	280ms
	0.50	0.50	0.0500	460062	65320	14.2	104ms	82ms	63ms	249ms

TABLE 2
Memory Cost

Dataset	Parameters			SEL11 [10]		Our Method		
Discription	Iso	α	β	Mesh	Pyramid(Texture)	Mesh	Pyramid(Buffer)	Feature
Head	0.25	0.00	0.0035	72.4MB	18 MB	18.0MB	63.3MB	224MB
256x256x225x8b	0.25	0.00	0.0130	72.4MB	18 MB	6MB	63.3MB	224MB
14.0MB	0.38	0.00	0.0040	85.3MB	18 MB	18.6MB	63.3MB	224MB
-	0.38	0.00	0.0120	85.3MB	18 MB	6.6MB	63.3MB	224MB
Hurricane	0.25	0.00	0.0015	63.1MB	1188 MB	13.6MB	49.3MB	212MB
661x401x42x32b	0.25	0.00	0.0055	63.1MB	1188 MB	4.6MB	49.3MB	212MB
42.4MB	0.25	0.35	0.0225	63.1MB	1188 MB	12.2MB	49.3MB	212MB
18^{th}	0.25	0.35	0.0240	63.1MB	1188 MB	5.5MB	49.3MB	212MB
Fluid Data	0.50	0.00	0.0150	47.4MB	9509MB	11.1MB	121.4MB	520MB
2048x56x239x32b	0.50	0.00	0.0350	47.4MB	9509MB	6.3MB	121.4MB	520MB
104MB	0.50	0.50	0.0400	47.4MB	9509MB	14.4MB	121.4MB	520MB
100^{th}	0.50	0.50	0.0500	47.4MB	9509MB	6.7MB	121.4MB	520MB

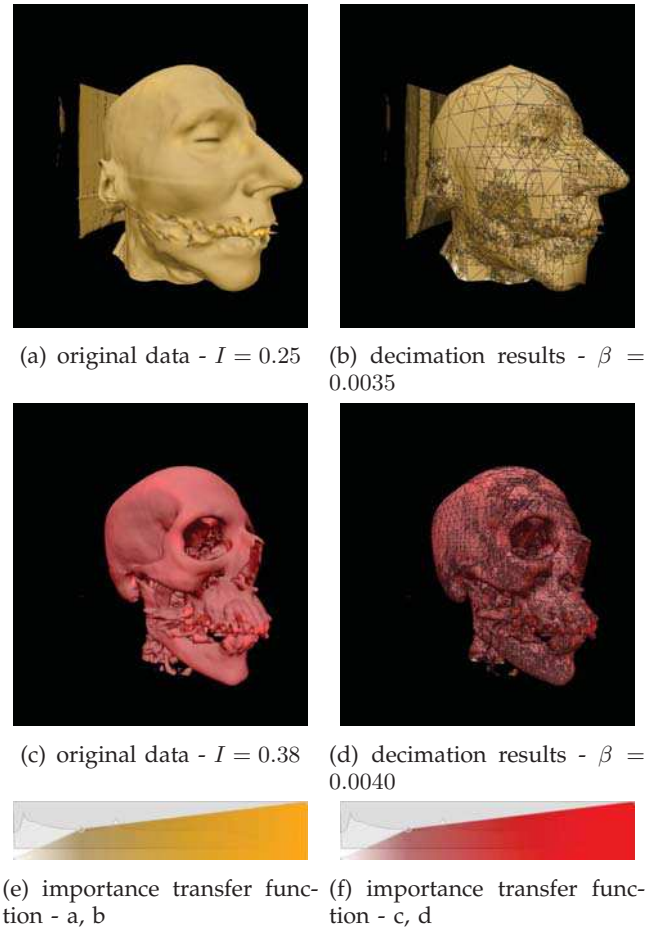
nitude), see "Head" dataset. If another scalar feature is used(see "Hurricane" dataset), this size can be five times the data size.

5.2 Mesh Decimation Results

We use both medical datasets and scientific datasets to evaluate our method. The evaluation includes two parts. The first part focuses on the geometric features, while the second part shows the decimation results combing geometric properties and user-specified features.

Fig.7 and Fig.8 show the results of mesh simplification using only geometric features. That is to say, the blending parameter α is set to zero. In Fig.7, the meshes around tooth have higher smoothness values, so they are preserved during simplification while the meshes on forehead are decimated. In Fig.8, meshes which are far from the eyes are decimated since they are smoother than the central meshes. So that the eye of the hurricane is preserved during mesh decimation.

Fig.9 and Fig.10 show the results of mesh simplification using user defined features. In Fig.9, the isosurfaces of moist are rendered and we use potential temperature as user-defined feature. Compare 8(a) with 9(a), we can see that the distributions of geometric and temperature features are quite different. By blending them with $\alpha = 0.35$, we get the results shown in 9(b), in which the meshes belong to right and bottom parts are decimated while the eye is preserved. In Fig.10, we use pressure feature while the original isosurface is about vof function which represents the density of cells. From 10(a) and 10(c) we can see that the geometric



(a) original data - $I = 0.25$ (b) decimation results - $\beta = 0.0035$

(c) original data - $I = 0.38$ (d) decimation results - $\beta = 0.0040$

(e) importance transfer function - a, b (f) importance transfer function - c, d

Fig. 7. Results of Head dataset with geometric features.

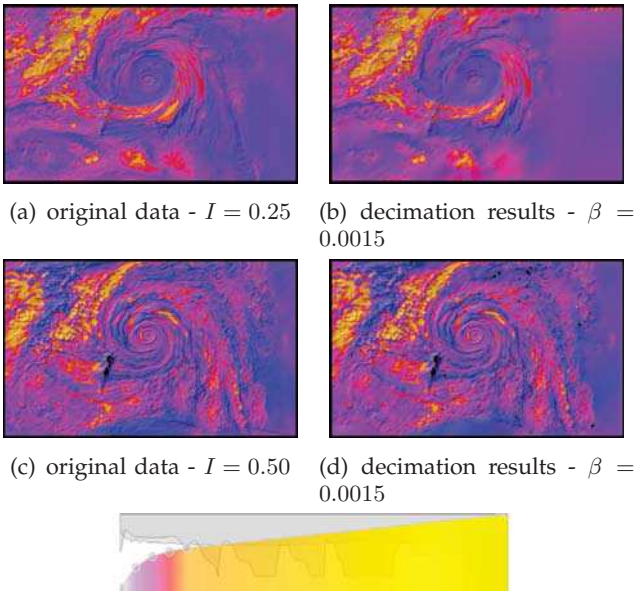
and pressure feature are slightly different. 10(f), 10(g), 10(h) show the decimation results. Since the central parts of vof and pressure features all have lower values, the central meshes in the final results are decimated.

From these figures, we can see that our method preserves both geometric feature and user-defined features well while performing mesh simplification.

6 Conclusion

In this paper, we introduce an importance-driven parallel mesh simplification method which has three main advantages:

1) mesh importance definition based on transfer functions allows users to interactively see the distributions of features with their interests;

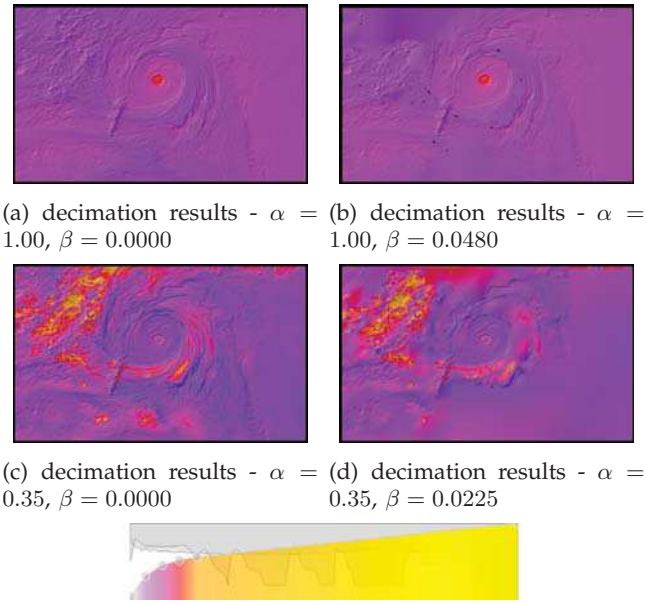


(a) original data - $I = 0.25$ (b) decimation results - $\beta = 0.0015$

(c) original data - $I = 0.50$ (d) decimation results - $\beta = 0.0015$

(e) importance transfer function

Fig. 8. Results of Hurricane dataset with geometric features.



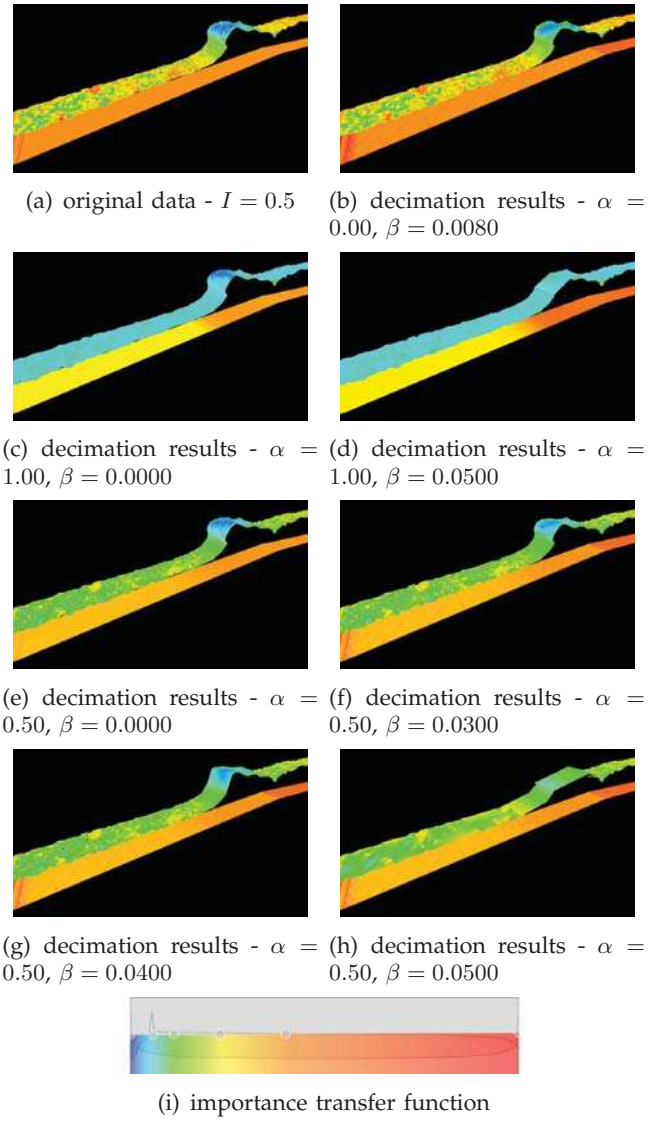
(a) decimation results - $\alpha = 1.00$, $\beta = 0.0000$ (b) decimation results - $\alpha = 1.00$, $\beta = 0.0480$

(c) decimation results - $\alpha = 0.35$, $\beta = 0.0000$ (d) decimation results - $\alpha = 0.35$, $\beta = 0.0225$

(e) importance transfer function

Fig. 9. Results of Hurricane dataset with user defined features.

2) mesh decimation before generation costs less memory and leads to higher rendering performance, which also support progressive mesh generation;
3) oriented to general computing ensures that our method can be easily integrated into other parallel systems;



(a) original data - $I = 0.5$ (b) decimation results - $\alpha = 0.00$, $\beta = 0.0080$

(c) decimation results - $\alpha = 1.00$, $\beta = 0.0000$ (d) decimation results - $\alpha = 1.00$, $\beta = 0.0500$

(e) decimation results - $\alpha = 0.50$, $\beta = 0.0000$ (f) decimation results - $\alpha = 0.50$, $\beta = 0.0300$

(g) decimation results - $\alpha = 0.50$, $\beta = 0.0400$ (h) decimation results - $\alpha = 0.50$, $\beta = 0.0500$

(i) importance transfer function

Fig. 10. Results of Fluid dataset with user defined features.

We also evaluate the performance of our algorithm and use several datasets to show the efficiency of our method.

Other future work includes improving current method to eliminate topological ambiguity, enhancing the performance of pyramid peeling stage since it is not so satisfactory, integrating more features and designing better interface for users to design transfer functions.

ACKNOWLEDGMENTS

We would like to thank Hongsen Liao, Min Wu, Zhu Zhu, Kan Wu for their enthusiastic help and helpful suggestions.

This research is supported by National Science Foundation of China(61272225, 51261120376), Chinese 863 Program(2012AA041606, 2012AA040902), Chinese 973 Program(2010CB328001) and the Important National Science & Technology Specific Projects (2011ZX02403).

REFERENCES

[1] D. Luebke. A developer's survey of polygonal simplification algorithms. *Computer Graphics and Applications, IEEE*, 21(3):24 –35, may/jun 2001.

[2] C. Wang, H. Yu, and K.-L. Ma. Importance-driven time-varying data visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1547–1554, nov.-dec. 2008.

[3] Y. Peng and L. Chen. Transfer function design using shape features in medical visualization. *Journal of Computer-Aided Design and Computer Graphics*, 23(1):78–84, jan. 2011.

[4] Y.-S. Wang, C. Wang, T.-Y. Lee, and K.-L. Ma. Feature-preserving volume data reduction and focus+context visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):171–181, feb. 2011.

[5] U. Bordoloi and H.-W. Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494, oct. 2005.

[6] C. Correa and K.-L. Ma. Visibility histograms and visibility-driven transfer functions. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):192–204, feb. 2011.

[7] M. Ruiz, A. Bardera, I. Boada, I. Viola, M. Feixas, and M. Sbert. Automatic transfer functions based on informational divergence. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):1932–1941, dec. 2011.

[8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987.

[9] F. Goetz, T. Junklewitz, and G. Domik. Real-time marching cubes on the vertex shader. In *Proceedings of Eurographics*, volume 2005, pages 1–4, 2005.

[10] E. Smistad, A. Elster, and F. Lindseth. Fast surface extraction and visualization of medical images using opencl and gpus. In *The Joint Workshop on High Performance and Distributed Computing for Medical Imaging (HP-MICCAI)*, 2011.

[11] M. Vetter and S. Olbrich. Scalability issues of in-situ visualization in parallel simulation of unsteady flows. *Competence in High Performance Computing 2010*, pages 177–190, 2012.

[12] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *Visualization '96. Proceedings.*, pages 335–342, 27 1996-nov. 1 1996.

[13] D. Attali, D. Cohen-Steiner, and H. Edelsbrunner. Extraction and simplification of iso-surfaces in tandem. In *Proceedings of the third Eurographics symposium on Geometry processing*, pages 139–148, Aire-la-Ville, Switzerland, 2005. Eurographics Association.

[14] G. Dupuy, B. Jobard, S. Guillon, N. Keskes, and D. Komatitsch. Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm. *Computer-Aided Design*, 42(2):129–138, feb. 2010.

Yi Peng is currently a Ph.D. student in Department of Computer Science and Technology at Tsinghua University. He received his B.S. in school of software from the Tsinghua University, China, in 2010. His research interests include data visualization, computer graphics and parallel computing.

Li Chen received the PhD degree in visualization from Zhejiang University in 1996. Currently, she is an associate professor in the institute of CG&CAD, School of Software, Tsinghua University. Her research interests include data visualization, mesh generation and parallel algorithm.

Jun-Hai Yong is currently a professor in School of Software at Tsinghua University. He received his B.S. and Ph.D. in computer science from the Tsinghua University, China, in 1996 and 2001, respectively. He held a visiting researcher position in the Department of Computer Science at Hong Kong University of Science & Technology in 2000. He was a post doctoral fellow in the Department of Computer Science at the University of Kentucky from 2000 to 2002. He obtained a lot of awards such as the National Excellent Doctoral Dissertation Award, the National Science Fund for Distinguished Young Scholars, the Best Paper Award of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, the Outstanding Service Award as Associate Editor of the Computers & Graphics Journal by Elsevier, and several National Excellent Textbook Awards. His main research interests include computer-aided design and computer graphics.