Constructing and Rendering of Multiresolution Representation for Massive Meshes with GPU and Mesh Layout

Yaping Zhang
Faculty of Computer Science & Information Technology, Yunnan Normal University, Kunming, Yunnan, China
Email: zhangyp.cs@gmail.com

Xu Chen
Department of Computer and Information Science, Southwest Forestry University, Kunming, Yunnan, China
Email: chenxu_gis@yahoo.com.cn

***Abstract*—Multiresolution technique is one of the most efficient approaches to improve the rendering performance, but its design and implementation for massive meshes are still very difficult. This paper researches and realizes constructing and rendering of multiresolution representation for massive meshes base on surface partition, which could provide vertex-grained local refinement and generate the optimal rendering quality. Our approach adopts dual hierarchy to represent the mesh. One is cluster hierarchy of progressive meshes for coarse-grained selective refinement. The other is vertex hierarchy built with progressive mesh in the cluster node to provide fine-grained local refinement. In order to promote the speed of local refinement, we introduce some data structures and dependency rules to realize parallel view-dependent refinement for vertex hierarchy by using GPU, which greatly reduces the load of CPU and enables it to prefetch data to hide I/O latency effectively. In addition, we propose a new mesh layout algorithm which reorders triangles contained by cluster node to reduce the average cache miss ratio and further improve the rendering speed.**

***Index Terms*—multiresolution representation, massive meshes, surface partition, GPU, mesh layout**

I. Introduction

Over the last decade, advances in model acquisition, computer-aided design (CAD), and simulation technologies have resulted in massive complex meshes [1]. We call these meshes as out-of-core meshes that largely overload the performance and memory capacity of state-of-the-art graphics and computational platforms, and so it is difficult to render them interactively. Multiresolution technique as one of the most efficient approaches to improve rendering performance, which needs to build a hierarchical structure implemented by mesh simplification or level of detail methods, can effectively reduce the geometric complexity of the model, and thus improve the rendering speed. However, there are many difficulties in the design and implementation of this technique for massive meshes. Firstly, as geometric data and auxiliary data structures of massive meshes can not be completely loaded into memory due to limited memory size, the traditional multiresolution modeling and rendering algorithm can not be directly applied to the massive meshes. Secondly, with limited bus bandwidth and CPU processing power, the construction process of multiresolution representation for massive meshes often takes much time, which is not conducive to system debugging and real-time applications. Therefore constructing and rendering of multiresolution representation for massive meshes has become a hot topic in the research area of computer graphics [2][3].

The modern Graphics Processing Unit (GPU) consists of many multiprocessors (MP) and additional memory, and uses the SIMT (Single Instruction Multiple Thread) parallel programming paradigm, which makes it have higher performance on handling graphics tasks than CPU [4]. Owing to the parallel architecture of GPU, the conventional algorithm, such as vertex hierarchy refinement, can not be applied directly to GPU.

Based on intensive study on massive mesh simplification and multiresolution techniques in recent years, we propose a novel approach for constructing and rendering out-of-core multiresolution representation for massive meshes based on surface partition, which could provide vertex-grained local refinement and generate the optimal rendering quality. In order to promote the speed of local refinement and rendering, we use GPU to perform parallel view-dependent refinement for vertex hierarchy, which greatly reduces the load of CPU and enables it to prefetch data to hide I/O latency effectively. In addition, we propose a new mesh layout algorithm to reduce the average cache miss ratio (ACMR) and further improve the rendering speed.

The remainder of this paper is organized as follows: Section Ⅱ briefly reviews previous work in this area, while Section Ⅲ introduces construction of multiresolution representation for massive meshes. Section Ⅳ presents out-of-core rendering. Some results and analyses are provided in Section Ⅴ. Finally we summarize our research and future work in Section Ⅵ.

doi:10.4304/jsw.8.8.1968-1975

II. Related Work

Considering the basic unit of level of detail, multiresolution representation for meshes can be divided into two categories: vertex hierarchy and cluster hierarchy. Each node in vertex hierarchy contains only one vertex, such as merge tree [5] constructed by Xia et al. This representation method was able to provide the most fine-grained level of detail, but for massive meshes, the overhead of selection and switching of level of detail was great. Each node in cluster hierarchy contains a number of triangles and vertices, which can reduce the switching cost of selection refinement and thus be suitable for representing massive meshes, such as [2][6].

According to objects divided, the cluster hierarchy can mainly be divided into two types: space partition cluster hierarchy and surface partition cluster hierarchy. The space partition cluster hierarchy divides bounding volume of the model. Lindstrom [7] used a sparse octree decomposition of space over a uniform rectilinear grid, and exploited vertex clustering on a rectilinear octree grid to coarsen and create a hierarchy for the mesh. The run-time component then traversed this hierarchy and produced an adaptive mesh that could be displayed interactively. Simplification, level-of-detail hierarchy construction were performed entirely on disk, and used only a small, constant amount of memory, whereas the run-time system pages in only the rendered parts of the mesh in a cache coherent manner. The limitation of this approach was that the multiresolution surface constructed could not provide the fidelity of the original mesh. Cignoni et al. [6] used a regular conformal hierarchy of tetrahedra to spatially partition the model. Each tetrahedral cell contained a precomputed simplified version of the original model, represented using cache coherent indexed strips for fast rendering. The representation was constructed during a fine-to-coarse simplification of the surface contained in diamonds (sets of tetrahedral cells sharing their longest edge). Appropriate boundary constraints were introduced in the simplification to ensure that all conforming selective subdivisions of the tetrahedron hierarchy lead to correctly matching surface patches. For each frame at runtime, the hierarchy was traversed coarse-to-fine to select diamonds of the appropriate resolution given the view parameters. Shaffer et al. [8] presented an external memory multiresolution surface representation for massive polygonal meshes, which also used a uniform grid to sample the mesh and build an external memory octree by a bottom-up merging process. The bottom level of this octree encoded the original surface, which formed the finest level of resolution. The construction phase required only two passes over the input mesh plus external sorts of the vertices and faces. The sorts ensured coherent access of the processed mesh data from disk

Space partition method is easy to implement, but the quality of division is not good, which usually uses the vertex clustering to simplify the divided sub-meshes. Surface partition method is in accordance with the model topology to partition model surface, which produces sub-meshes with better uniformity and flatness, and commonly uses edge collapse to simplify the sub-meshes. Surface partition cluster hierarchy can generate better rendering results, but it is complex to implement. Clustered Hierarchy of Progressive Meshes (CHPM) [9] is currently the best one among known construction and rendering algorithms of multiresolution representation based on the surface partition for massive models. The method represented the model as a clustered hierarchy of progressive meshes and used the cluster hierarchy for coarse-grained selective refinement and progressive meshes for fine-grained local refinement. However, refinement of progressive meshes is ordered, which means CHPM requires many more unnecessary vertex splits and renders more triangles to meet the view-dependent criteria compared to a vertex hierarchy usually when rendering. From the table of runtime performance given by CHPM, we can find that about 90% of the total time is used to rendering. Based on the above analysis, performing selective refinement with the vertex hierarchy instead of linear progressive mesh may be a good method to minimize the number of triangles rendered. However, the selective refinement for vertex hierarchy is time-consuming, which can lead to higher running time.

The construction process of multiresolution representation for massive meshes often takes much time, which is not conducive to system debugging and real-time applications. There are two main solutions. (1) The use of PC clusters. For example, Cignoni et al. [6] built a tetrahedron hierarchy with 1, 4, 8, and 14 workers. Overall processing times ranged from about 3K-4K triangles/s for 1 CPU to 15K-30K triangles/s for 14 CPU. Zhang et al. [2] built the external memory octree for lucy model proposed by reference [8] with 4 workers, the speedup of processing time was 2.08:1. Although parallel construction of multiresolution representation for massive meshes using PC clusters can effectively shorten the preprocessing time, the flexibility of deployment has been limited. (2) The use of GPU. DeCoro et al. [10] used geometry shader of GPU to achieve real-time simplification based on vertex clustering, and gained acceleration ratio of 15-25 compared to CPU implementation. Ji et al. [11] used GPU parallel computing power to accelerate view-dependent multiresolution hierarchy refinement algorithm based on LOD atlas texture of geometric image, the GPU-accelerated implementation achieved more than an order of magnitude performance gain over CPU version. Two rendering passes using this method were included. During the first pass, the level of detail selection was performed in the fragment shaders. The resultant buffer from the first pass was taken as the input texture to the second rendering pass by vertex texturing, and then the node culling and triangulation could be performed in the vertex shaders. The approach could generate adaptive meshes in real-time. However, with the limits of the vertex shaders, merging vertices to suppress the T-junction might cause the generation of degenerated triangles. Hu et al. [12, 13] realized parallel view-dependent refinement of progressive mesh, which was the first vertex hierarchy refinement algorithm based on GPU. Such fine-grain

control has previously been demonstrated using sequential CPU algorithms. However, these algorithms involve pointer-based structures with intricate dependencies that cannot be handled efficiently within the restricted framework of GPU parallelism. By introducing new data structures and dependency rules, Hu et al. realized fine-grain progressive mesh updates as a sequence of parallel streaming passes over the mesh elements. But it only handled the models which can be loaded into main memory.

Most modern computers adopt hierarchies of memory levels. Each level of memory serves as a cache for the next level, such as cache-main memory-disk hierarchy. Lower levels are larger in size and farther from the processor and have slower data access times, typically disk. Data transfer is performed whenever there is a cache miss between two adjacent levels of the memory hierarchy. For massive meshes, its data is stored in disk initially. With the difference of access speed between memory levels, usually several orders of magnitude, we need to optimize the layout of mesh to minimize the ACMR during rendering. Lin et al. [14] proposed a simple yet effective algorithm for generating a sequence for efficient rendering of 3D polygonal meshes based on greedy optimization, which was one of the classic mesh layout algorithms. Their strategy was to associate each vertex with a cost value, which was tailored to reduce cache misses. Specifically, the vertex with the minimum cost would be picked as the focus vertex. For each face connecting to the focus vertex that had not been rendered, its vertex(es) would be pushed into buffer and it would be rendered and output. The cost metric of this method was the combination of three weighting coefficients. However, how to set these weights for any given mesh was not straightforward and its computational complexity was more than $O(t)$ (t represents the number of triangles contained in the input mesh). Sander et al. [15] simplified the cost metric proposed by reference [14], and only considered the position of the vertices in the cache as factor. Computational complexity of the algorithm was close to $O(t)$, but ACMR was slightly lower than that of reference [14].

III. Constructing of Multiresolution Representation for Massive Meshes

We represent the model as dual hierarchies (Fig. 1): a cluster hierarchy of progressive meshes for coarse-grained selective refinement and a vertex hierarchy for progressive meshes contained in the cluster node for fine-grained local refinement. The construction process includes cluster hierarchy generation, building vertex hierarchy adapted to parallel view-dependent refinement, and mesh layout.

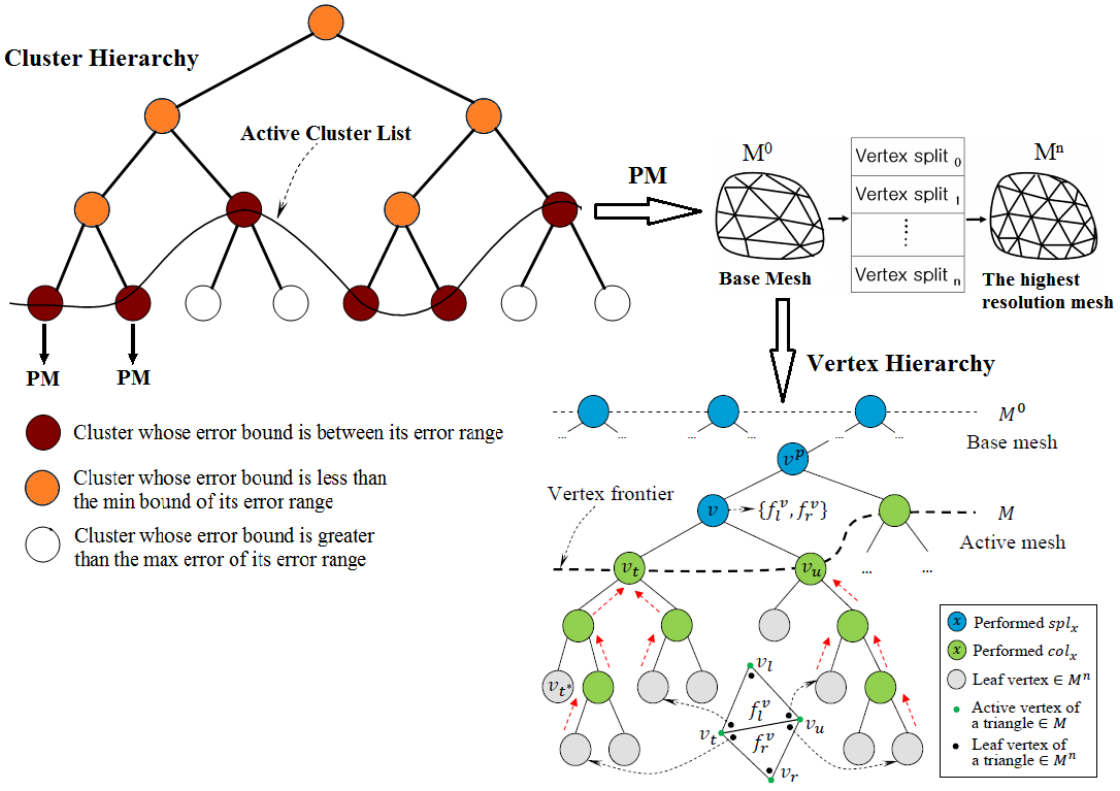


Figure 1. Dual hierarchy

A. Cluster Hierarchy Generation

Cluster hierarchy generation proceeds in three steps: First, we decompose the model into clusters, which are spatially localized portions of the input mesh. The generated clusters should be nearly equally sized in terms of number of triangles for several reasons. This property is desirable for out-of-core mesh processing to minimize the memory requirements. Moreover, enforcing spatial

locality and uniform size provides higher performance for selective refinement. The decomposition occurs in several passes to avoid loading the entire input mesh at once. These clusters facilitate out-of-core access to the mesh for the remaining steps [9]. Next, we construct the cluster hierarchy using a graph partitioning algorithm [16] which can make it with nearly equal cluster size, high spatial locality, well-balanced structure and minimum shared vertices. The algorithm represents each cluster as a node in a graph, weighted by the number of vertices. Clusters are connected by an edge in the graph if they share vertices or are within a threshold distance of each other. The edges are weighted by the number of shared vertices and the inverse of the distance between the clusters, with greater priority placed on the number of shared vertices. The cluster hierarchy is then constructed in a top-down manner by recursively partitioning the graph into halves considering the weights, thus producing a binary tree. Finally, we build the progressive meshes (PM) for each cluster by applying "half-edge collapses." After creating the PM, the error range of the cluster is computed and expressed as a pair: (*min bound, max error*). The *max error* is the error value associated with the base mesh (M^0) and the *min bound* is the error value associated with the highest resolution mesh (M^n). When proceeding to the next level up the hierarchy, the mesh within each cluster's PM is initialized by merging the base meshes of the children. Since the intermediate clusters should be nearly the same size as the leaf level clusters, each cluster is simplified to half its original face count at each level of the hierarchy [9].

At runtime, we maintain an active cluster list (ACL), which represents a front in the cluster hierarchy containing the clusters of the current mesh (as shown in Fig. 1), and perform coarse-grained selective refinement on this list.

B. Vertex Hierarchy Generation

Vertex hierarchy can be built easily according to the linear progressive mesh. However conventional vertex hierarchy refinement algorithms involve pointer-based structures with intricate dependencies that cannot be handled efficiently within the restricted framework of GPU parallelism, we introduce some data structures to realize fine-grain vertex hierarchy updates of progressive mesh [12]. The method maintains a set of static structures used to store vertex hierarchy of progressive mesh, a set of dynamic structures encode the active, selectively refined mesh. A unique aspect is that the active mesh is fully specified by a stream of vertices. This stream contains all vertices "above" the active frontier in the vertex hierarchy, such as all the blue nodes called active vertices in Fig. 1. Splitting each active vertex will create two active faces. The indices of the vertices in every pair of faces $\{f_l, f_r\}$ are obtained by retrieving the indices of the leaf vertices in the same faces in M^n and searching up the hierarchy for the coarsest vertices in collapsed states. The algorithm performs a set of parallel streaming passes to update the vertex stream as the view parameters change, and to create an index buffer for rendering [12].

During the PM construction, for each col_v, each removed face f_l and f_r is adjacent to two other mesh faces, $\{f_{n0}, f_{n1}\}$ and $\{f_{n2}, f_{n3}\}$ respectively, as shown in Fig. 2. As a result of edge or vertex-pair collapses a triangle may "foldback" on itself or changes its normal by about π (as shown in Fig. 3). We refer to this as a mesh fold-over or just a foldover [18].

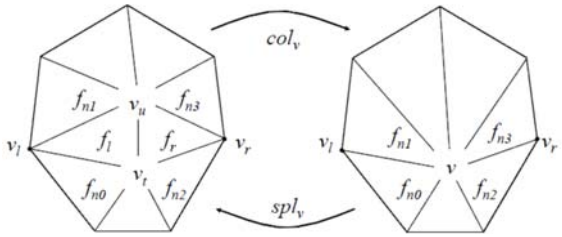


Figure 2. The neighborhood around a split/collapse operation [12].

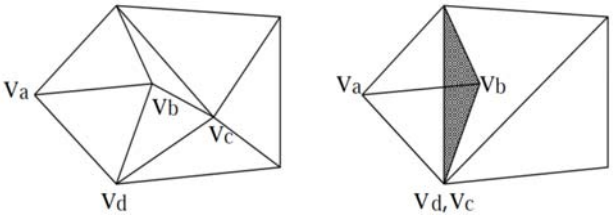


Figure 3. Foldover case [18]

To prevent foldovers of the triangles in the mesh, the splits and collapses must adhere to dependency rules. The explicit rules [17] check for the presence and adjacency of these four faces in the current selectively refined mesh. Specifically the rules are as follows:

(i) A split spl_v is legal if the faces $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$ all exist in the current selectively refined mesh.

(ii) A collapse col_v is legal if f_l is currently adjacent to $\{f_{n0}, f_{n1}\}$ and f_r is currently adjacent to $\{f_{n2}, f_{n3}\}$.

Unfortunately, test (ii) involves maintaining face adjacencies, which is difficult in a parallel algorithm.

The implicit dependencies rely on the enumeration of vertices generated after each collapse. If the model has *n* vertices at the highest level of detail they are assigned vertex-ids *0, 1, … , n-1*. Every time a vertex pair is collapsed to generate a new vertex, the id of the new vertex is assigned to be one more than the greatest vertex-id thus far. This process is continued till the entire vertex hierarchy has been constructed. The implicit dependencies are as follows:

(I) Vertex-Pair Collapse: A vertex-pair can be collapsed if the vertex-id of their parent is less than the vertex-ids of the parents of the collapsed boundary vertices.

(II) Vertex Split: A vertex *v* can be safely split at runtime if its vertex-id is greater than the vertex-ids of all its neighbors.

The explicit rules [17] incur extra memory and are not suitable for framework of GPU parallelism, whereas the implicit rules [18] are too restrictive and require many more unnecessary vertex splits to meet the view-dependent criteria. Additionally, they all require relatively complex runtime tests. We use an approach

[12], which follows the same refinement flexibility as the explicit rules, but with a more compact representation inspired by the implicit rules, and most importantly it is well adapted to GPU stream processing due to its simplicity. The approach is to perform a simpler check that involves computing and storing two vertex indices (v_{lmax} and v_{rmax}) instead of four face indices.

$$v_{l\max} = \max(c_v(f_{n0}), c_v(f_{n1}))$$

$$v_{r\max} = \max(c_v(f_{n2}), c_v(f_{n3}))$$

Where $c_v(f)$ is a non-ancestral vertex split that creates f. More precisely, $c_v(f)$ is the vertex x whose split creates face f, unless $f \in M^0$ or x is an ancestor of v, in which cases $c_v(f) = 0$. The vertex hierarchy is linearized in memory, with vertices assigned indices in the reverse order that they were collapsed. Thus the leaf vertices are consecutive and last, and can be distinguished from non-leaf vertices solely by their index. For any vertex v, the ordering also implies that $v > v^p$ (the parent of v). At runtime, given the side vertices v_l and v_r in M, we can check legality as follows [12]:

(I) A split spl_v is legal if $v_l > v_{lmax}$ and $v_r > v_{rmax}$.

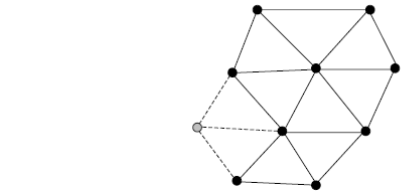
(II) A collapse col_v is legal if $(v_l)^p < v$ and $(v_r)^p < v$.

C. Mesh Layout

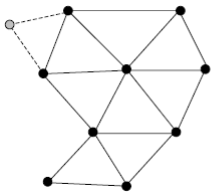
Due to the amount of data of multiresolution representation for massive meshes is too large, only part of the multiresolution structure can be loaded into memory during rendering. Because of the difference of access speed between the different storage media, usually several orders of magnitude, we should minimize the number of cache misses to improve the overall performance. In this paper, considering the first-in-first-out (FIFO) cache models, we propose a new algorithm to reorder triangles contained by cluster node to reduce the ACMR during rendering effectively. The algorithm selects the vertex as the focus which can output the maximum number of triangles when pressing it into cache. It is similar to the greedy algorithm, which chooses the operation to minimize increment of the cache miss ratio.

Fig. 4 gives several examples of the selection of the focus vertex for mesh layout, wherein the black solid dot indicates that the vertex is in the cache, the hollow white circle indicates that the vertex is not yet pressed into the cache. In Fig. 4(a), if the white vertex is pressed into the cache, it can output two triangles, the ACMR is 1/2 (mismatches a vertex, and outputs two triangles); in Fig. 4(b), if the white vertex is pressed into the cache, it can output a triangle, the ACMR is 1/1 (mismatches a vertex, and outputs one triangle); in Fig. 4(c), if any one white vertex is pressed into the cache, it can not output any triangle until another one is pressed into cache, which can output a triangle, therefore the ACMR is 2/1 (mismatches two vertexes, outputs a triangle). Consequently, based on the current cache miss ratio, the algorithm selects the white vertex in Fig. 4(a) as the focus to make the cache miss with the lowest ratio, thus to reduce the overall average ACMR.

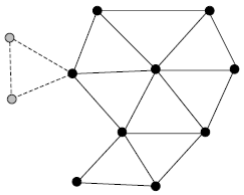
If the above cost metric is used directly, it may cause the new vertex pressed into the cache constantly. However for some vertices earlier pressed into cache, because the ACMR related to their adjacent vertices (not in the cache) is not the minimum value of the global search, they can not be accessed and may be pressed out cache soon. When later accessing these vertices, we need to press them into cache again. To solve this problem, our cost metric function takes into consideration the location of vertex in cache. The basic idea is that, if there are a number of candidate vertices with the minimum ACMR, the algorithm selects a candidate vertex adjacent to the vertex which is the first one to enter the cache as the current focus, and outputs all of its connecting triangles. This can ensure to access the vertex which will leave the cache as early as possible. As shown in Fig. 5, for candidate vertices a and b, they also can output two triangles and the ACMR is the same as 1/2, regardless of which vertex pressed into the cache. But since the vertex 2 enters cache earlier than the vertex 3, which means the vertex 2 will also leave the cache earlier, the algorithm choose the vertex a as the focus vertex.



(a) Two triangles can be output



(b) One triangle can be output



(c) No triangle can be output

Figure 4. The selection of the focus vertex for mesh layout

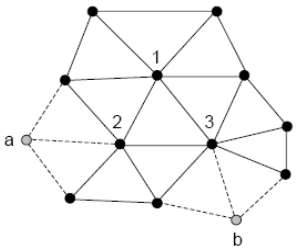


Figure 5. The position of vertices in the cache

Ⅳ. OUT-OF-CORE RENDERING

The entire representation is stored on the disk. We load the coarse-grained cluster hierarchy and keep a working set of cluster into main memory. We perform coarse-grained refinement at the cluster hierarchy and fine-grained refinement at the vertex hierarchy. The algorithm introduces a frame of latency in the rendering pipeline in

order to fetch the newly visible clusters from the disk and avoid stalls in the rendering pipeline.

The algorithm maintains an active cluster list (ACL), as shown in Fig. 1, which is a cut of clusters in the hierarchy representing the scene. During each frame, we refine the ACL based on the current viewing parameters. Specifically, we traverse the ACL and compute the *error bound* for each cluster. Each cluster on the active front whose *error bound* is less than the *min bound* of its *error range* (see Section Ⅲ(A)) is split because the highest resolution mesh cannot meet the *error bound*. Similarly, sibling clusters that have a greater *error bound* than the *max error* of their *error range* are collapsed. As for local refinement for active clusters, the algorithm firstly checks for desirable edge collapses and vertex splits, and updates the vertex states accordingly; then updates and maintains the stream of active vertices based on the updated states; and finally generates the index buffer using the set of split vertices and the updated frontier implied from the states [12].

To improve rendering speed, we transfer geometry data of clusters in the current ACL into GPU memory and let GPU responsible for fine-grained refinement and rendering for the vertex trees within the clusters. When we update clusters in the ACL by performing cluster-collapse and cluster-split operations, the children and parent clusters are activated. But the data of these clusters may not be loaded into GPU memory, which can stall the rendering pipeline. To prevent these stalls, whenever a cluster is added to the ACL, we use a separate thread to prefetch its parent and children clusters into main memory. When a cluster in the ACL needs to split, we don't split it temporarily if its children nodes are still not loaded into main memory. But if its children nodes already in main memory, we transfer the data into GPU and leave the space for other nodes prefetched. Similarly, we don't perform cluster-collapse temporarily if the parent node is still not in main memory. Otherwise we move the data from main memory into GPU. In order to save storage space, we only keep two levels of the hierarchy above and below the current ACL in memory.

At the same time, in order to efficiently process a large number of pre-fetch requests, we prioritize them based on the prediction of level of detail updating and of the user's operation, and insert them into the priority queue, which makes the request with higher priority be completed more quickly.

Ⅴ. RESULTS

We have implemented our algorithm on a dual 3.0 GHz PC, with 2GB of RAM and a NVIDIA GeForce 9800 GTX+ GPU with 512MB of video memory. Table 1 shows construction performance of multiresolution representation for several massive meshes, including the number of clusters in the entire hierarchy, the total amount of data, and preprocessing time for cluster decomposition, hierarchical simplification and mesh layout. Since cluster hierarchy generation consumes little time, which is included in the total time for cluster decomposition. Vertex tree construction time is included in hierarchical simplification time. Hierarchical simplification is the most time-consuming step of the construction algorithm. This is mainly because the algorithm needs to calculate quadratic error metric of each vertex within the cluster to construct tree hierarchy for progressive mesh, and merge the base meshes of children clusters as the highest resolution mesh approximation of parent cluster. With the deepening of the cluster level, the time of hierarchical simplification increases significantly. Since our algorithm uses GPU to realize parallel view-dependent refinement of vertex tree within the cluster, which can get the least triangle sets to meet the current viewpoint parameter required, we don't need to divide the original mesh into thousands of small clusters like CHPM [9]. Thai Statue model in Table 1, for example, is divided into 256 leaf nodes in our algorithm and the entire hierarchy contains 511 cluster nodes, each cluster contains about 20K vertices. Table 2 shows the rendering performance for several massive meshes, which includes the number of cluster nodes selected under current viewpoint parameters, the number of triangles rendered and the rendering time. The screen projection error is set to one pixel. Fig. 6 shows the rendering results of the corresponding model. For the number of triangles rendered, the rendering effect of the model is satisfactory with an interaction rate of about 20fps.

Ⅵ. CONCLUSION AND FUTURE WORK

There has been an explosion in the size of 3D meshes during recent years, in part due to the drastic improvements in resolution and accuracy of data acquisition devices, such as laser range and CT/MRI scanners. Because the extremely high fidelity of gigantic meshes, they are applicable in many fields, such as culture heritage protection, digital museum and virtual human project. And thus high-quality rendering for these gigantic meshes has been the pursuing goal of computer graphics. Multiresolution representation based on space partitioning methods only consider the geometric information of the original mesh without concern for its topology, its rendering quality is relatively poor. This paper researches and realizes constructing and rendering of multiresolution representation for massive meshes base on surface partition, which can support vertex-level mesh refinement and provide the best rendering results. Our approach provides dual hierarchies. One is cluster hierarchy of progressive meshes for coarse-grained selective refinement. The other is vertex hierarchy which built with progressive mesh in the cluster node to provide fine-grained local refinement. In order to promote the speed of local refinement, we take advantage of GPU performing parallel view-dependent refinement for vertex hierarchy, which greatly reduces the load of CPU and enable it to prefetch data to hide I/O latency effectively. In addition, we propose a new algorithm for mesh layout to reorder the base mesh and further promote the rendering performance.

The efficiency of our algorithm is derived from the parallel execution of GPU and CPU, multi-thread data prefetch supported by dual-core CPU. Now it is a hot

research topic in recent years to achieve a wide range of applications with the processing power of GPU, such as Zheng et al. [19] uses GPU general-purpose computing and CUDA technology on RRTM (Rapid Radiative transfer model) module in Global and Regional Assimilation and Prediction System. The optimization results indicate that a 14.3 × speedup is obtained compared to CPU implementation. We believe that it is the development trend to mine parallelism of traditional algorithm, adopt parallel processing unit in GPU pipeline, as well as assembled PC cluster or GPU clusters for parallel processing.

TABLE I.
PREPROCESS TIMINGS AND STORAGE REQUIREMENTS FOR TEST MODELS

Modal	Dragon	Thai Statue	Lucy
The number of vertices	3,609,600	4,999,996	14,027,872
The number of triangles	7,219,045	10,000,000	28,055,742
Original size (MB)	130	180	508
The number of clusters	255	511	1,023
Total size (MB)	416	576	1,625
Cluster decomposition (min)	2.7	4.4	18
Hierarchical simplification	35	46	138
Mesh layout (min)	1.5	3	5.8
Processing time (min）	39.2	53.4	161.8

TABLE II.
RENDERING PERFORMANCE

Modal	Num triangles	Num clusters	Rendered triangles	Rendering time (ms)
Dragon	7,219,045	8	307,216	12
Thai Statue	10,000,000	12	376,532	19
Lucy	28,055,742	12	529,960	41

(a) Dragon

(b) Thai Statue

(c) Lucy

Figure 6. Rendering results

ACKNOWLEDGMENT

This work was supported by applied basic research programs of Yunnan Province (2010CD047) and the National Natural Science Funds (61262070).

REFERENCES

[1] E. Gobbetti, D. Kasik, and S. Yoon, “Technical strategies for massive model visualization,” In Proceedings of the 2008 ACM symposium on Solid and physical modeling, pp.405-415, 2008.
[2] Yaping Zhang, Xiong Hua, Xiaohong Jiang and Shi Jiaoying, “Out-of-Core Constructing and Interactive Rendering of Multiresolution Representations for Massive Meshes,” Journal of Computer-Aided Design & Computer Graphics, vol.20, No.9, pp.1126-1131, 2008. (in Chinese)
[3] Yaping Zhang, Xiong Hua, Xiaohong Jiang and Shi Jiaoying, “A Survey of Simplification and Multiresolution Techniques for Massive Meshes,” Journal of Computer-Aided Design & Computer Graphics, vol.22, No.4, pp.559-568, 2010. (in Chinese)
[4] Qingkui Chen, Haifeng Wang, Songlin Zhuang,Bocheng Liu, “Parallel Algorithm of IDCT with GPUs and CUDA for Large-scale Video Quality of 3G,” Journal of Computers, vol. 7, No. 8, pp.1880-1886, 2012
[5] J. C. Xia, and A. Varshney, “Dynamic view-dependent simplification for polygonal models,” In Proceedings of the 7th Conference on Visualization. New York, ACM Press, pp.327-334, 1996.
[6] P. Cignoni, F. Ganovelli, E. Gobbetti, et al, “Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models,” ACM Transactions on Graphics, vol.23, No.3, pp.796-803, 2004.
[7] P Lindstrom, “Out-of-core construction and visualization of multiresolution surfaces,” In Proceedings of the 2003 symposium on Interactive 3D Graphics, New York, USA, 2003: 93-102
[8] E. Shaffer and M. Garland, “A multiresolution representation for massive meshes,” IEEE Transactions on Visualization and Computer Graphics, vol.11, No.2, pp.139-148, 2005.
[9] S. E. Yoon, B. Salomon, R. Gayle and D. Manocha, “Quick-VDR: Out-of-Core View-Dependent Rendering of Gigantic Models,” IEEE Transactions on Visualization and Computer Graphics, vol.11, No.4, pp.369-382, 2005
[10] C. DeCoro and N. Tatarchuk, “Real-time mesh simplification using the GPU,” in Proceedings of Symposium on Interactive 3D Graphics and Games, New York, USA, pp.161-166, 2007
[11] Junfeng Ji, Enhua Wu, Sheng Li, Xuehui Liu, “View-dependent refinement of multiresolution meshes using programmable graphics hardware,” Visual Comput, vol.22, pp.424–433,2006
[12] L. Hu, P. V. Sander, and H. Hoppe, “Parallel view-dependent refinement of progressive meshes,” In Proceeding(s) of ACM Symposium on Interactive 3D Graphics and Games, pp.169-176, 2009.
[13] L. Hu, P. V. Sander, and H. Hoppe, “Parallel View-Dependent Level-of-Detail Control,” IEEE Transactions on Visualization and Computer Graphics, vol. 16, No. 5, September/Octorber 2010.
[14] G. Lin and PYY Thomas, “An improved vertex caching scheme for 3d mesh rendering,” IEEE Transactions on Visualization and Computer Graphics, vol.12, No.4, pp.640-648, 2006.

[15] P. V. Sander, D. Nehab, and J. Barczak, "Fast triangle reordering for vertex locality and reduced overdraw," In Proceedings of SIGGRAPH 2007, New York, USA, ACM, 2007:89-98
[16] G. Karypis and V. Kumar, "Multilevel k-Way partitioning scheme for irregular graphs," Parallel and Distributed Computing, 1998.
[17] H. Hoppe, "View-dependent refinement of progressive meshes," In Proceedings of ACM SIGGRAPH, pp. 189-198, 1997.
[18] J. El-Sana and A. Varshney, "Generalized view-dependent simplification," In Proceedings of Eurographics, pp. 83-94, 1999.
[19] Fang Zheng, Xianbin Xu, Dongdong Xiang, Zhuowei Wang, Ming Xu, Shuibing He, "GPU-Based Parallel Researches on RRTM Module of GRAPES Numerical Prediction System," Journal of Computers, vol. 8, No. 3, pp.550-558, March 2013

Yaping Zhang was born in Yunna Province, China in 1979. She received the MS degree in computational mathematics from Yunnan University in 2005, and PhD degree in Computer Science from Zhejiang University in 2010, China. She is a lecturer in the Computer Science Department of Yunnan Normal University, Kunming, Yunnan, China. Her research concentrates on real-time computer graphics and parallel computing.

Xu Chen was born in Yunna Province, China in 1973. He received the MS degree in Computational mathematics from Yunnan University in 2004, and PhD degree in ecology from Sun Yat-sen University in 2011, China. He is currently an associate professor in Department of Computer and Information Science, Southwest Forestry University, Kunming, Yunnan, China. His research interests include remote sensing image processing and GPU Computing.