Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Cluster parallel rendering based on encoded mesh*

QIN Ai-hong†[1,2], XIONG Hua[1], PENG Hao-yu[1], LIU Zhen[1], SHI Jiao-ying[1]
([1]*State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, China*)
([2]*School of Computer and Information Technology, Shanxi University, Taiyuan 030006, China*)
†E-mail: qinaihong@cad.zju.edu.cn

Received Apr. 3, 2006; revision accepted Apr. 17, 2006

Abstract: Use of compressed mesh in parallel rendering architecture is still an unexplored area, the main challenge of which is to partition and sort the encoded mesh in compression-domain. This paper presents a mesh compression scheme PRMC (Parallel Rendering based Mesh Compression) supplying encoded meshes that can be partitioned and sorted in parallel rendering system even in encoded-domain. First, we segment the mesh into submeshes and clip the submeshes' boundary into Runs, and then piecewise compress the submeshes and Runs respectively. With the help of several auxiliary index tables, compressed submeshes and Runs can serve as rendering primitives in parallel rendering system. Based on PRMC, we design and implement a parallel rendering architecture. Compared with uncompressed representation, experimental results showed that PRMC meshes applied in cluster parallel rendering system can dramatically reduce the communication requirement.

Key words: Cluster, Parallel rendering, Rendering, Compression algorithm, Mesh coding, Mesh segmentation
doi:10.1631/jzus.2006.A1124 **Document code:** A **CLC number:** TP391-41

INTRODUCTION

Three dimensional (3D) meshes are widely used to represent 3D objects. With the rapid advances in digital acquisition technology, meshes with millions of vertices are becoming increasingly common. Because of memory constraints and lack of graphics power, visualizations of this magnitude are difficult or impossible to perform even on the most powerful workstations. Therefore, the need for parallel implementation is clear. Most previous work in high performance rendering has focused on building large, high-end computers with multiple tightly coupled processors. However, expensive hardware restricts it from being widely used. In recent years, PC clusters have become an attractive alternative to traditional high-end graphics workstations.

In PC cluster based parallel rendering system, the primitives (usually triangles) are distributed to different processors, in a sorting operation. Depending on at what stages of the graphics pipeline (Molnar *et al*., 1992) the sorting operation is performed, the parallel strategies can be classified as sort-first, sort-middle, and sort-last.

In the sort-first strategy (Mueller, 1995), the image space is partitioned into tiles. Each processor is responsible for the rendering calculations of the tiles to which it is assigned. Each 3D triangle is then distributed to the corresponding processor(s). Finally the image regions from all the processors are combined together to form the rendered image. As the sort-first architecture is generic enough to be applied to both sort-middle and sort-last architectures, we will concentrate on this technique in this paper.

In the sort-first architecture, communication requirement from the client to rendering servers is where the bottleneck is. One solution is to partition the model and distribute the data among PCs. Currently the dynamic, view-dependent partition strate-

* Project supported by the National Basic Research Program (973) of China (No. 2002CB312105), the National Natural Science Foundation of China (No. 60573074), the Natural Science Foundation of Shanxi Province, China (No. 20041040), Shanxi Foundation of Tackling Key Problem in Science and Technology (No. 051129), and Key NSFC Project of "Digital Olympic Museum" (No. 60533080), China

gies (Molnar *et al*., 1992; Mueller, 1995; Samanta *et al*., 1999; 2000a) demonstrate high efficiency. Another solution to this problem is to use a compressed representation during the transfer. As a result, when the viewpoint changes per frame, the set of triangles for each tile also changes accordingly. Therefore, the compression algorithm should be run to all the tiles per frame. Unfortunately, current triangle mesh compression algorithms are too computation intensive to be performed on-the-fly per frame. In addition, there are no partition algorithms that can subdivide the state-of-the-art compressed mesh during sorting. In fact, a scratching subdividing will completely destroy the original triangle mesh structure.

In this paper, we present a compression algorithm PRMC (Parallel Rendering based Mesh Compression) that focuses on compressing large models which will not be sorted in a single node. As we compress the models from the standpoint of view-dependent partition and sorting, there are two distinguishing advantages of our approach for the parallel rendering. First, the partitioning and sorting become more efficient because the number of submeshes is far smaller than that of the triangles. Second, using a compressed representation during the transfer significantly reduces the communication requirement.

RELATED WORK

Samanta *et al*.(1999; 2000a) developed a sort-first rendering system using a network of commodity PCs. In order to achieve a well-balanced system, they developed dynamic screen partitioning schemes, which in some cases could become the bottleneck. Moreover, the system is not scalable because the model had to be replicated in the main memory on each of the nodes of their cluster.

In subsequent work, Samanta *et al*.(2000b) developed a hybrid sort-first/sort-last parallel rendering algorithm. Their new approach could perform dynamic, view-dependent partition. Although the geometry is again replicated on each of the nodes, experiments implied that view-dependent partition strategy can efficiently cull the data outside the view-frustum and balance the rendering load on the nodes. So it is adopted widely by others.

In more recent work, Samanta *et al*.(2001) addressed the replication problem. They stored copies of the model only in k of the available n nodes, where $k<n$. Still, neither their preprocessing phase nor their rendering phase could handle a model larger than the total amount of main memory in their cluster.

The out-of-core preprocessing algorithm (Correa *et al*., 2002) created an on-disk hierarchical representation of the input model. Only the sections being processed will be kept in memory. The algorithm is easy to implement. But high transmission requirement poses a great challenge to its practical application.

Tulika and Tzi-cker method (Mitra and Chiueh, 2002) is somewhat different. They used compressed mesh in the parallel rendering architecture and gave a recipe to build a tiled encoded mesh through clipping the original compressed BFT mesh into tiled BFT mesh. Experiments showed that parallel rendering system based on tiled BFT mesh reduces one third of the communication requirement. But before the triangles clip checking, the compressed mesh must be decompressed. As the viewpoint changes per frame, the tiled BFT mesh also must change, which means both the decompression algorithm and the tiled compression algorithm should be run correspondingly during sorting.

To sum up, all the approaches above focus on reducing the communication requirement and improving rendering rate. Experiments showed that view-dependent and compressed-domain based partition and sorting strategy can significantly improve the rendering performance.

Although view-dependent partition strategy has been studied widely in recent years, little work has been done on the compressed algorithm for parallel rendering system. In this work, we propose a compression algorithm PRMC which can integrate both the view-dependent strategy and the compressed-domain strategy into the parallel rendering system.

PRMC

The state-of-the-art compression algorithms are too complicated to access normals and positions in the compressed domain. Here we use PRMC to solve this problem. PRMC segments the mesh into small submeshes and compresses each submesh separately.

The properties of the encoded submeshes, such as normal cones and bounding boxes are explicitly stored and can be accessed randomly. So a compressed submesh can substitute for a triangle in sorting.

Our proposed method consists of three phrases:

(1) Mesh segmentation;

(2) Submesh compression;

(3) Sorting compressed submesh.

The data flow of PRMC is shown in Fig.1 and explained in the following sections.

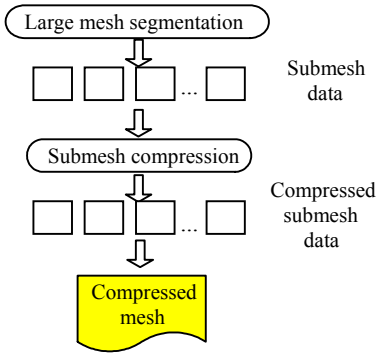

Fig.1 Data flow of PRMC algorithm

Mesh segmentation

1. Criterion of segmentation

As a matter of fact, mesh segmenting is much like graph partitioning problem. Graph partition partitions the vertices of a graph in p roughly equal clusters or patches, and the number of edges connecting vertices in different partitions is minimized. In fact, the graph partitioning problem is NP-complete. Here, we just attempt to segment the cluster to make it more reasonably suitable for view dependent partitioning and sorting. The concept "reasonably" is highly subjective and hard to qualify in the algorithm. Two basic criterions of reasonable segmentation are listed below:

First, each region of the submesh is "localized" somewhere in the mesh. Then, the partition should in general be balanced in the sense that each submesh should contain approximately the same count of geometric primitives.

Motivation is that submeshes within a localized region would share similar geometry. Count balance of geometric primitives ensures that every submesh can serve as one sorting primitive, which simples the implementation of sorting step. Both criterions may assure high sorting efficiency.

In most situations the ideal results are seldom provided. We cannot precisely find the global optimal partition. Only the proximate optimal result can be achieved.

2. Segmentation algorithm

Researches have proposed many approaches so far. A famous polygon partition software packet is MeTiS (Karypis and Kumar, 1998). It employs k-way partitioning algorithm to partition the polygon into c total compact patches of balanced vertex counts. But the large model which cannot be loaded in memory will not be partitioned by MeTiS directly, such as Lucy from Stanford Computer Graphics Laboratory, Davids and the St. Matthew statue in the Digital Michelangelo Project at Stanford University, the Double Eagle model in Newport News Shipbuilding, etc. So we first subdivide the bounding box of the model into a regular grid of cubical cells. Then, every cell is partitioned by MeTiS to get balanced submeshes. Results are presented in Fig.2 (see page 1131).

Submesh compression

In this work we consider the strategy of connectivity and the geometry information compression. Normals, colors, and texture mapping vectors are compressed in the same way (Deering, 1995; Taubin and Rossignac, 1998).

The vertices in the submesh are classified into two distinguishing types, namely inner vertices and boundary vertices. A vertex is a boundary vertex if it is shared by triangles of different submeshes or if its 1-ring neighbors are homeomorphic to a half-disk. Otherwise it is an inner vertex.

As the inner vertices of one submesh are separated from other submeshes, they can be compressed individually. However, the boundary vertices supply important information to stitch submeshes back together. To keep the right connection among different rendering servers, some extra work should be done. In this work, we focus on the strategy of boundary vertices compression.

Connectivity compression

Many mesh compression algorithms have been investigated. In this work we modify an existing one to suit parallel rendering system. However, extending

them from solo computer to parallel system is not trivial because of the difficulty in keeping the right connection among the submeshes on different servers.

Here we care only for the lossless algorithm. This constraint prohibits the use of progressive approaches (Cohen-or *et al*., 1999; Pajarola and Rossignac, 2000; Alliez and Desbrun, 2001). Not only because the total disk space required by progressive meshes increases by twice that of the origins, random mesh access for refinement operations during decompression is also required.

There are many lossless mesh compression algorithms, but only those which are consistent with the two criteria below are suitable for parallel rendering system:

(1) The compression ratio is insensitive to the submeshes size;

(2) Efficient decoding.

The two constraints above prohibit the approaches such as Taubin's (Taubin and Rossignac, 1998) whose compression ratio depends on the model size. They also eliminate those algorithms whose connectivity and geometry decompression are done in separate passes (Senburg and Snoeyink, 2000; Szymczak *et al*., 2002), which is costly in efficiency.

Certainly there are some compression approaches (Gunhold and Strasser, 1998; Touma and Gotsman, 1998; Mitra and Chiueh, 2002) according with the rules above. In this work, we choose the simple and efficient degree coder algorithm (Touma and Gotsman, 1998) for triangle meshes and extend it to polygon in (Isenburg and Alliez, 2002). It traverses a triangle mesh by sequentially visiting the face adjacent to the gate of the active boundary. The strategy can deal with both the manifold mesh and the non-manifold mesh. But the "parallelogram" rule adopted in their coordinate compression will cost longer decoding time than Taubin's (Taubin and Rossignac, 1998) geometry compression algorithm. In this work we adopt Taubin's coordinate compression strategy in which a vertex's position can be predicted by its preceding vertices based on a predictor equation. Here we use the linear predictor. But nonlinear predictors are also contemplated in this scheme.

Submesh boundary compression

Some compression strategies on submeshes boundary are introduced in few Out-Of-Core mesh compression algorithms (Isenburg and Gumhold, 2003; Ho *et al*., 2001; Hoppe, 1998; Bernardini *et al*., 2002). Ho *et al*.(2001) compressed the submesh boundaries into the submeshes which they belong to, so most of the boundaries vertices are compressed more than once. In addition to high storage requirement, it also induces crack between the shared submeshes boundaries. One solution to this problem is to compress the boundary vertices as a whole. As a result, the whole submesh boundaries have to be stored on client and on every server for random access during decoding.

Our solution to this problem is to piecewise compress the boundary vertices (Fig.3, see page 1131). First, their geometry information is separately piecewise compressed. Then we compress their connectivity together with the submeshes they belong to. The indexes of boundary vertices are recoded in their corresponding compressed submeshes. Benefiting from a novel boundary buffer data structure that we constructed, the boundary can be rapidly piecewise accessed in compression domain. Some definitions are given below:

A boundary vertex that is shared by more than two boundary edges is named Saddle Point (Fig.3). Run is some successive boundary edges shared by adjacent submeshes, which start from a Saddle Point. Each submesh boundary has several Runs. Run Gate is the endpoint of the Run, which is usually a Saddle Point.

Fig.3 shows four submeshes, S_1, S_2, S_3 and S_4, which are coloured differently. '*A*', '*D*', '*F*' are Saddle Points. Successive vertices "*ABCD*", "*DEF*" and "*FGHA*" are three Runs. '*A*', '*D*' and '*F*' are Run Gates.

Every Run is shared by at most two submeshes in a 2-manifold mesh. The boundary of a mesh is clipped into many Runs. Space coherence indicates that when a submesh is loaded, its Runs will probably be loaded; when a vertex in the Run is loaded, the adjacent vertices in the same Run will be loaded soon. Based on those rules, every Run can act as a loading primitive. Consequently, we compressed each Run individually.

Beginning from one of the Run Gates (we name this Run Gate the Active Run Gate (ARG)), vertices on it will be compressed successively. Positions of ver-

tex are compressed using predictor equation (Taubin and Rossignac, 1998). Predictors are the preceding nodes in a Run. ARG will not be compressed, for ARG acts as the first predictor of a Run. All the encoded Runs are stored in a boundary buffer file as shown in Fig.4.

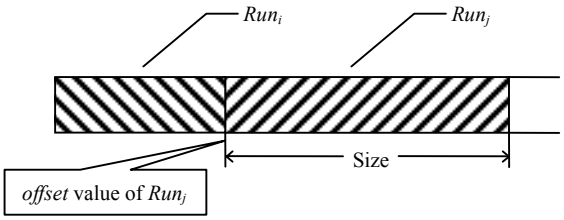

Fig.4 Representation of boundary buffer file

Every Run has an auxiliary CSGT (Client Submesh Gate Table) node, to index the Run in the boundary buffer file. It includes three elements: V_{ARG} (ARG's coordinate), *offset*, *size*. *offset* (Fig.4) is a Run's start position in boundary buffer file, while *size* indicates the encoded Run's length. CSGT enables us to feasibly access each encoded Run in the file randomly.

We also construct an index of each submesh boundary. It is represented as a link, named boundary link. Every node of the link has three elements: CSGT (introduced above), *neighbor_id* and *next_link*. *neighbor_id* is the neighbor submesh which shares this Run with (Fig.5). *next_link* points to next Run of this boundary.

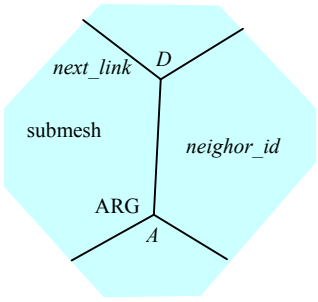

Fig.5 A bird's-eye represent of submeshes. Four gray patches specify four submeshes. Black lines indicate Runs

Fig.6 is a boundary link example of submesh S_1 in Fig.3. S_1 has three Runs. As a result, there are three nodes in S_1's boundary link. The first three items in every node are elements of CSGT. The fourth value is *neighbor_id*. It indicates that S_1 shares the first Run with S_2, shares the second with S_4 and the third with S_3.

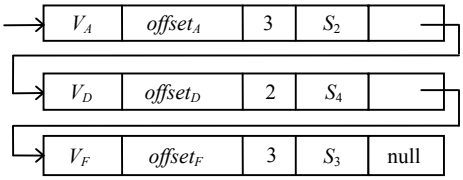

Fig.6 Represent of boundary link

To enhance the efficiency of view-dependented partitioning and sorting in parallel rendering system, the piecewise encoded mesh is organized in an auxiliary table CST (Client Submesh Table) with 10 elements: *visible*, *use*, *s_id*, *BoundingBox*, *CentrePoint*, *NormalCone*, *TriangleCounts*, *VerticesCounts*, *Pdata*, *Plink*. Here *visible* bit specifies if a mesh is visible for a particular view. *use* bit indicates whether this submesh is in a server's memory. *s_id* specifies the rendering server in which this submesh is sorted in. The other five successive elements are submesh parameters used in view-depended partitioning and sorting. *pdata* stores the file name of encoded submesh data, while *plink* points to its boundary link. Each CST node corresponds to one submesh.

To sum up, we piecewise compress everyone of the submeshes and Runs instead of the whole model. With two auxiliary data structures: CST and CSGT, which are index tables of compressed submeshes and Runs respectively, both submeshes and Runs can be accessed randomly in compression domain.

SYSTEM ORGNIZATION

We adopt sort-first architecture in our work. As shown in Fig.7, our system comprises a client PC, server PCs, and a display PC arranged in a three-stage pipeline. During the first stage, the client partitions the screen into tiles and assigns each of them to a different server. Next, every server renders all the graphics primitives assigned to it to form a subimage. As a consequence, the server sends the resulting pixels to the display PC. Finally, the display PC composites all the subimages in its frame buffer for display.

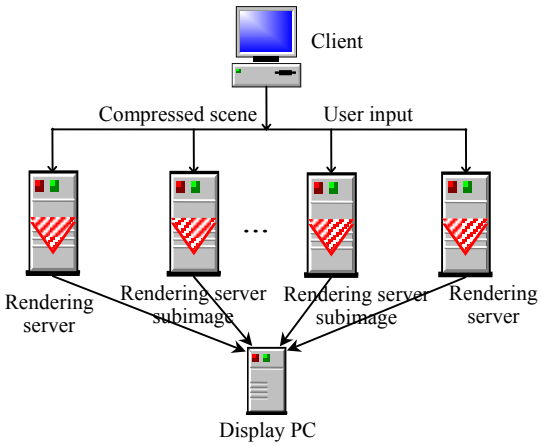

Fig.7 Sort-first architecture

Different from existing systems, the data of input and transmitting of this architecture are PRMC compressed meshes consisting of many compressed submeshes. After sorting the compressed submeshes into rendering servers, decoders are implemented transparently. Fig.8 shows a parallel rendering strategy based on PRMC.

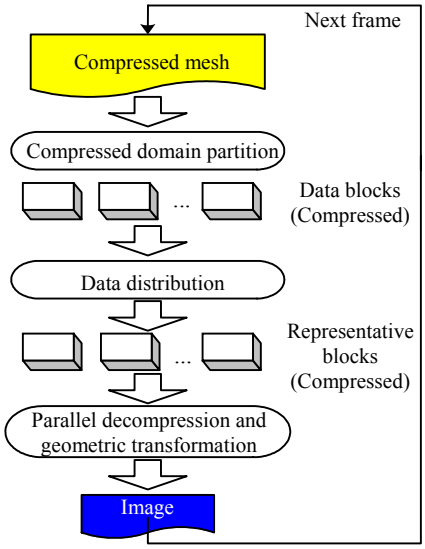

Fig.8 Data flow of parallel rendering

ENCODED-DOMAIN SORTING

The parallel rendering system sorts the encoded submeshes from client to the rendering servers (pc_1, pc_2, …, pc_n). As an encoded submesh is a set of co-spatial triangles that share similar geometry, it may assure higher sorting efficiency than sorting original triangles one by one. Moreover, the count of compressed submeshes is much less than that of original triangles, which will dramatically decrease the computing cost of sorting.

The sorting will be conducted as the viewpoint changes. That is, sorting the compressed submeshes (without decompression) that can be visible at current viewpoint, to occlude other submeshes behind it along the viewing direction and outside the visible region. Here we adopt a conservative predictive occlusion culling.

There are three steps in encoded-domain sorting:

First, client searches CST table to find some visible submeshes that are not on the rendering servers. Then, client gets the file names of the encoded submeshes and the positions of Runs. Finally, client loads the encoded submeshes and Runs on rendering servers according to the work load partition strategy of the parallel system.

If the vertices of shared boundary have already decompressed on the same rendering servers, they need not be transmitted on the server and be decoded again. Only its index needs to be recoded in both shared submeshes.

Decoding is transparently implemented when the first few bytes of the compressed submeshes are loaded on rendering servers. The geometry information of boundary vertices should be decoded before the inner vertices, because submeshes must access the corresponding boundary vertices during decoding. Data structures of decoded vertices and triangles are the same as that of the aforementioned mesh structures, which ensure PRMC compression cause little effect on the parallel architecture. Pseudo-code for a simple sorting algorithm on PRMC mesh is listed in Table 1 and Table 2. In the submesh sorting algorithm, $(v'_{i_1}, v'_{i_2}, ..., v'_{i_s})$ is loaded from $p_2.offset$ to $p_2.offset$ +*size* in the CSGT.

To eliminate redundant transmission of shared boundary, we construct a temp auxiliary SST (Sever Submesh Table) on servers. Every SST node corresponds to a submesh on this server now and consists of a submesh's original index on client and a link pointing to server submesh's boundary link, whose node consists of a copy of the Run's CSGT and the Run's start position in server's vertex table.

Table 1 Sorting algorithm

```
function sorting(VIEWPOINT e)
begin
repeat until all the submeshes in the CST are checked
    Get the information of submesh_i
    Check submesh_i.normal_cone
    Calculate visibility of submesh_i under e
    if visible(submesh_i)=true
        Calculate the server's id pc_j for submesh_i
        if use(submesh_i)=false
            Set s_id(submesh_i)=pc_j
            Sort submesh_i into pc_j
        else if s_id(submesh_i)≠pc_j
            Invalidate submesh_i in s_id(submesh_i)
            Modify s_id(submesh_i)=pc_j
            Sort submesh_i into pc_j
end
```

Table 2 Submesh sorting algorithm

```
function submesh_sort(integer i, integer j)
begin
  p1=submesh_i.plink
  p2=p1.nextlink
  repeat until p2=NULL
    begin
    shared=p1.neighbor_id
    if use(shared)=false or s_id(shared)≠pc_j
      begin
      Load compressed boundary vertex (v'_i1, v'_i2, ..., v'_is)
          of submesh_i into pc_j
      Decode the (v_i1, v_i2, ..., v_is) of (v'_i1, v'_i2, ..., v'_is)
      end
      Set incident data structure in the CST, SST
      p2=p2.nextlink
    end
  Load patch_i.pdata into pc_j
  Decode patch_i.pdata and set incident data structure
end
```

RESULTS

We test PRMC with several models listed in Table 3. Pre-processing test was conducted in a windows PC with 1 GB memory and a 2.4 GHz Pentium IV processor. Table 4 shows some results of pre-processing step.

Table 3 Models

Model	Vertices	Triangles	Compression ratio
Horse	48485	96966	5.6
Armadillo	172974	345944	6.2
Dragons	437645	871414	7.63
Buddha	543652	1087716	8.8
Lucy	14027872	28055740	16.5

The primitive's counts in submeshes are very much balanced. Table 4 shows that our primitive's counts balance is below 1.2, which is sufficient for our needs. Consequently, a balanced rendering loading will be easily achieved.

Table 4 Characterize of the mesh

Mesh name	Armadillo	Buddha	Lucy
Size (M)	6.59	40.6	520
Segments	4	9	16
Submeshes	500	1005	5080
Primitive balance	1.20	1.14	1.13
Total segmentation & METIS partition time (s)	6.1	10.73	1797
Compression time (s)	2.7	8.5	1370.5
Sorting & transmission triangles per ms	8346	8920	27553
Decoding triangles per ms (on one server)	2165	2804	3730
Compression ratio	6.2	8.8	16.5

The pre-processing is fast. It takes several seconds to partition buddha model into large segments and then into small submeshes (Table 4). It needs more time to segment Lucy model because it is too large to be loaded in the memory and need to read and write disk several times, which consumes a lot of time.

The compression speed is high. Because our large model is segmented into many submeshes, it can be parallel compressed in cluster, which is faster than that of compressing the whole model directly. The decoding speed of our algorithm is also high. In practice, the time consumed by decoding is less than that shown in Table 4 as the decoding is implemented almost parallel on 15 rendering servers. So, to some extent the time consumed by decoding can be negligible.

PRMC compression ratios are shown in Fig.9. It ranged from 1:4 to 1:17 when we chose 16 bits per coordinate. In fact, the compression ratio of PRMC can be improved more than twice by integrating entropy encoding into it. As a result, the compression ratio will come to 8~34. But further compressing the encoded mesh by entropy encoding will possibly bring a little longer decoding time.

The compression ratio varies with model size. Because boundary vertices require some auxiliary

space to store CSGT and CST, the compression ratio of boundary vertices is smaller than that of inner vertices. It means the larger the model is, the higher the compressed ratio will be achieved if the count of submeshes is the same.

Our compression ratio is some higher than that of Ho *et al*.(2001) who compressed the entire boundary vertex twice. In (Ho *et al*., 2001), Lucy's compression ratio is 15.2 at 16 bits per coordinate quantization even it is further compressed by arithmetic entropy encoder.

Compared with (Touma and Gotsman, 1998; Isenburg and Alliez, 2002), a kind of traditional in-the-core mesh compression algorithm which encodes mesh as a whole, our compression ratio is some lower when mesh is segmented into too many small submeshes. But when the count of Runs is small enough to be ignored, PRMC's compression ratio will approximate to them.

Algorithm in (Isenburg and Gumhold, 2003) is an out-of-core large mesh compressed algorithm. It out-of-core compresses a large mesh as a whole. So the encoded mesh is much similar to that of a traditional in-the-core compression algorithm. Their compression ratio is a little bit higher than ours.

Although PRMC is not a compression algorithm with very high ratio compared with Touma's and Isenburg's, parallel rendering system employing PRMC's encoded mesh has much less rendering redundancy than that by loading the whole compressed mesh. For example, in Fig.10a, if each server loads all those visible encoded meshes which are assigned to it,

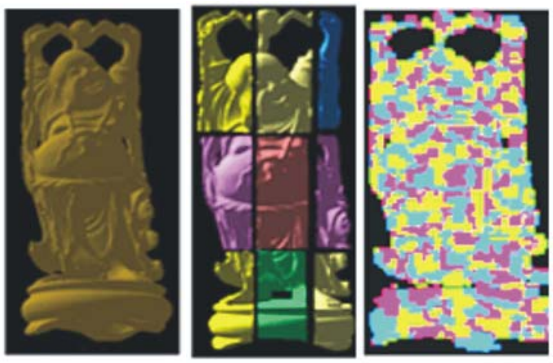

Fig.2 Partition results: Original model (a); The original model is divided into large segments (b); Partitioning large segments into small submeshes with almost balanced size (c). There are 1005 submeshes in the buddha model

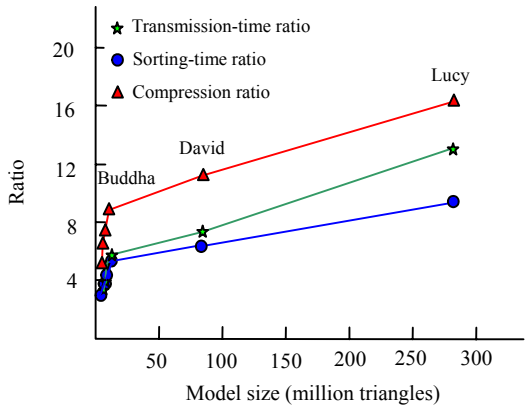

Fig.9 Average compression ratio, rate ratio of sorting and transmission varied with model size

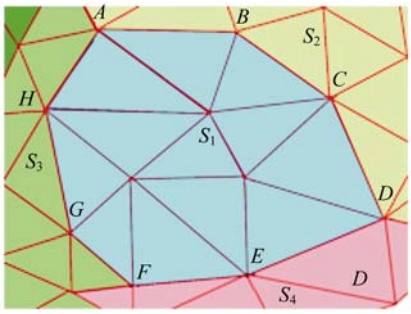

Fig.3 Representation of submesh

(a)

(b)

Fig.10 Two frames of rendering results on a cluster parallel system with 15 PCs. (a) Buddha; (b) Lucy

there may be 5 times more loading requirement of Touma's and Isenburg's encoded meshes than loading PRMC encoded meshes.

As a whole, compared with in-the-core or out-of-core mesh encoded algorithms for solo computer, PRMC sacrifices a little space to improve rendering speed. It can avoid the heavy redundancy brought about by them and achieve good balance between communication rates and rendering rates. As a consequence, high rendering performance will be achieved.

We demonstrate our rendering results in a cluster of 15 rendering servers with 1 G interconnecting network bandwidth. Every rendering server has two Intel XEON 2.4 GHz CPUs and 1 GB memory. The rendering results of Buddha and Lucy are showed in Figs.10a and 10b. Our parallel system employs both view-dependent partition and PRMC, each server just loads those visible encoded submeshes which are assigned to it. At the right side of Fig.10b, only one-fourth encoded data of Lucy need to be loaded. A lot of redundancy is eliminated.

In addition, although PRMC designed for sort-first parallel system which transmits data from cluster to server's memory, it also shows excellent performance when data are stored replicated in the disk of servers. PRMC mesh can be transmitted faster from disk to memory than uncoded mesh and brings much less redundancy than wholly compressed mesh.

CONCLUSION AND FUTURE WORK

In this paper, we introduced a mesh compression algorithm PRMC which is suitable for view-dependent parallel rendering. We also demonstrate a sort-first parallel architecture which uses PRMC mesh. Experimental results showed that PRMC encoded mesh can dramatically reduce the communication bandwidth requirement in the parallel rendering pipeline.

This is the first step towards studying how to design compressed meshes for parallel rendering system. In the future, we plan to investigate a more efficient sort-first parallel architecture based on PRMC. We will study other kinds of mesh segment algorithm, for example, to segment mesh into small patches with little curvature and balanced counts, which may further enhance loading efficiency because primitives in a patch with less curvature have the same visibility at a viewpoint. We also plan to design parallel rendering system based on encoded dynamic scene.

ACKNOWLEDGEMENT

Special thanks to Mai Huang and Qin He for production help. The Happy Buddha and Lucy are courtesy of the Stanford Computer Graphics Laboratory.

References

Alliez, P., Desbrun, M., 2001. Progressive Encoding for Lossless Transmission of 3D Meshes. Proceeding of SIGGRAPH'01, p.198-205.

Bernardini, F., Martin, I.M., Mittleman, J., Rushmeier, H., Taubin, G., 2002. Building a digital model of Michelangelo's Florentine Pietà. *IEEE Computer Graphics & Applications*, **22**(1):59-67. [doi:10.1109/38.974519]

Cohen-or, D., Levin, D., Remez, O., 1999. Progressive Compression of Arbitrary Triangular Meshes. Visualization'99 Conference Proceedings, p.67-72.

Correa, T.W., Klosowski, T.J., Silva, T.C., 2002. Out-of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays. Proceedings of PGV 2002 4th Eurographics Workshop on Parallel Graphics and Visualization, p.89-96.

Deering, M., 1995. Geometric Compression. Proceeding of SIGGRAPH'95, p.13-20.

Gunhold, S., Strasser, W., 1998. Real Time Compression of Triangle Mesh Connectivity. Proceeding of SIGGRAPH'98, p.133-140.

Ho, J., Lee, K., Kriegman, D., 2001. Compressing Large Polygonal Models. Visualization'01 Conference Proceedings, p.357-362.

Hoppe, H., 1998. Smooth View-dependent Level-of-detail Control and its Application to Terrain Rendering. Visualization'98 Conference Proceedings, p.35-42.

Isenburg, M., Alliez, P., 2002. Compressing Polygon Mesh Geometry with Parallelogram Prediction. Visualization'02 Conference Proceedings, p.141-146.

Isenburg, M., Gumhold, S., 2003. Out-of-core Compression for Gigantic Polygon Meshes. *ACM Trans. Graphics (SIGGRAPH'03)*, **22**(3):935-942. [doi:10.1145/882262.882366]

Karypis, G., Kumar, V., 1998. METIS: Version 4 A Software Package for Partitioning Unstructured Graphs. University of Minnesota, http://glaros.dtc.umn.edu/gkhome/views/metis.

Mitra, T., Chiueh, T., 2002. Compression-Domain Parallel Rendering. Proceedings of the International Parallel and

Distributed Processing Symposium IPDPS'02.

Molnar, S., Eyles, J., Poulton, J., 1992. PixelFlow: High-speed rendering using image composition. *Computer Graphics (SIGGRAPH'92)*, **26**(2):231-240. [doi:10.1145/142920.134067]

Mueller, C., 1995. The Sort-First Rendering Architecture for High-Performance Graphics. Proceedings of the ACM Symposium on Interactive 3D Graphics. Monterey, California, USA, p.75-84

Pajarola, R., Rossignac, J., 2000. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, **6**(1):79-93. [doi:10.1109/2945.841122]

Samanta, R., Zheng, J., Funkhouser, T., Li, K., Singh, J.P., 1999. Load Balancing for Multi-projector Rendering Systems. SIGGRAPH/Eurographics Workshop on Graphics Hardware, p.107-116.

Samanta, R., Funkhouser, T., Li, K., Singh, J.P., 2000a. Sort-first Parallel Rendering with a Cluster of PCs. SIGGRAPH'00 Technical Sketches, p.260.

Samanta, R., Funkhouser, T., Singh, J.P., 2000b. Hybrid Sort-first and Sort-last Parallel Rendering with a Cluster of PCs. SIGGRAPH/Eurographics Workshop on Graphics Hardware, p.97-108.

Samanta, R., Funkhouser, T., Li, K., 2001. Parallel Rendering with k-way Replication. Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics, p.75-84.

Senburg, M., Snoeyink, J., 2000. Face Fixer: Compressing Polygon Meshes with Properties. Proc. SIGGRAPH'00, p.263-270.

Szymczak, A., Rossignac, J., King, D., 2002. Piecewise regular meshes: construction and compression. *Graphical Models*, **64**(3-4):183-198. [doi:10.1006/gmod.2002.0577]

Taubin, G., Rossignac, J., 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics*, **17**(2):84-115. [doi:10.1145/274363.274365]

Touma, C., Gotsman, C., 1998. Triangle Mesh Compression. Graphics Interface'98 Conference Proceedings, p.26-34.