Optimal Sequencing of the Polygonal Meshes on the Basis of the Geodesic Distance Calculation

Hung-Kuang Chen
Department of Electronic Engineering, National Chin-Yi University of Technology
hankchentw@gmail.com

Chi-Feng Chen
Department of Electronic Engineering, National Chin-Yi University of Technology
kid3541@hotmail.com

Abstract

Concurrent 3D model files mostly adapt traditional indexed faced format such as *ply* and *obj*. With the improvements of model scanning technology, the file size becoming larger and larger as the resolution of the resulted 3D meshes increases. To process large meshes of such, we usually cannot load the complete mesh into the main memory; instead, a technique called out-of-core processing is commonly needed. However, such technique usually requires considerable amount of additional time consumed by the overhead incurred from the disk I/Os. Acceleration of such disk traffics through advanced caching is possible, which requires data coherence. In traditional indexed faced mesh, the data accesses are usually incoherent, i.e., the distribution of mesh data is sparse, which makes subsequent processing become more complicated and inefficient.

In this paper we addressed this issue, and propose a new method for optimal sequencing of the mesh data to optimize the data access coherence by the computation of geodesic distances. The experimental results have shown that the access of the optimized 3D model file achieves higher consistency and efficiency. In addition, with the optimized file, the memory footprint and the hard disk access time are significantly reduced.

Keywords: out-of-core, geodesic distance, optimal sequencing

1. Introduction

As a consequence of the advancements of the 3D scanning technology, the representations of the 3D objects are becoming more and more realistic and sophisticated, which directly leads to the enormous size of very large mesh. In such circumstance, the file is too large to fit in the main memory. To deal with such problem, a number of techniques are developed [1-9]. For such methods, the distribution of data has significant impact to the efficiency. Better access locality or data coherence is desirable, which may greatly reduce the number of disk accesses and thus improves the data access efficiency.

As shown in the Figure 1(a), traditional triangular indexed face meshes keeps at least an array of vertices coordinates and an array of triplet of indices to the vertex array. However, the sequencing of the vertices and faces usually are not optimized to increase the locality. The access to a local neighborhood of the mesh usually requires random accesses to the vertex/face array that may cross a wide range of the disk file.

To solve this problem, we may rearrange the allocation of the vertex and face data for a given access sequence to optimize the locality of references. For this purpose, in this paper, we proposed a method for optimizing the sequence of a given 3D triangular mesh on the basis of the calculation of the geodesic distance. The resulting meshes of our method may have better locality of reference and smaller memory footprint in comparison with original input.

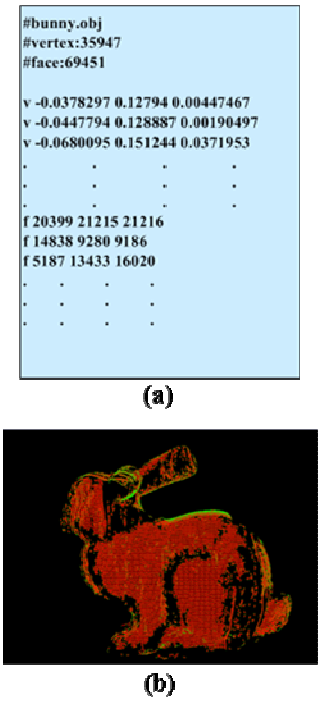

Figure 1. (a) A traditional indexed face mesh file (.OBJ); (b) the rendered image

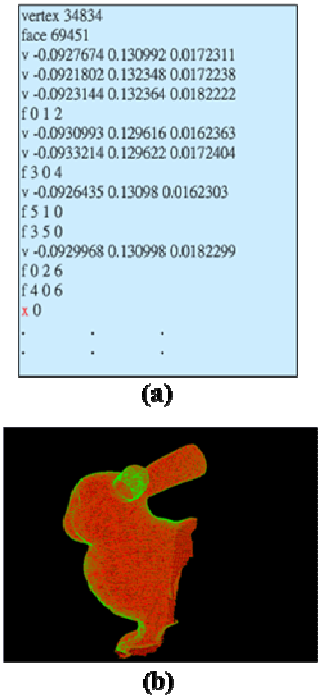

Figure 2. (a) The optimized streaming mesh file; (b) the visualization of the data accesses

2. Related Works

To deal with large datasets, a commonly used technique called out-of-core processing keeps only the part of currently processed datasets in the main memory space and stores the complete datasets in external memory space. To better optimize such techniques, the accesses to the complete dataset are usually on the basis of a block. If most of the data accesses are local to a block, the efficiency will be greatly improved.

One of conventional out-of-core techniques called the mesh cutting begin with cutting the input large mesh into smaller pieces of submeshes acceptable for conventional in-core techniques; afterwards, the submeshes are processed individually. The results are then combined in the final stage. Examples such as those proposed in [1] for mesh simplification and [2] for mesh compression. In such methods, the datasets must be properly segmented and reindexed to achieved the desirable result. Difficulties also arises from the search of a proper mechanism to segment the input mesh and to merge the individual results while maintaining consistent boundary between the resulting submeshes.

Some other techniques such as those proposed by [5] and [8] reside in advanced external memory data structures. However, these techniques also share the same aforementioned re-indexation problems.

In addition, another newer technique proposed in [3] suggests streaming the meshes. In their work, they have demonstrated the concept of mesh streaming in the application of mesh simplification. In their approach, a small but variable size of buffer holds the incoming and in-process part of mesh in the core memory and outputs the finished parts directly to the disk file. A mesh simplification algorithm proposed in [6] made uses of such idea and suggested improving its efficiency by optimizing the mesh sequence with rearrangement of vertex and face data. With the optimized sequence, the accesses to the external datasets are contiguous and confined to a local region of constant size, which solves the problem resulted from the dynamic-sized buffer of the method proposed in [3].

Inspired by the concepts proposed by [4], in this paper, we proposed a method for the optimization of the mesh sequence on the basis of the calculations of the geodesic distance. With our method, the better optimal polygonal meshes can be got.

3. The Mesh Sequence Optimization Algorithm

In this context, we have assumed the input mesh to be a common indexed-face triangular mesh. Our algorithm accepts such input and converts it to an optimized streaming result based on the calculation of geodesic distance in the framework similar to [4].

As illustrated in the Figure 3, the new algorithm consists of four stages. In the first stage, the vertices and faces are separated from the input mesh and the degree of each vertex is calculated for the later use in optimizing the layouts of the vertices and the triangles. After the optimized layouts of the vertices and triangles are computed, the computations are finalized by interleaving the vertices with the faces to create an optimized streaming mesh to the disk.

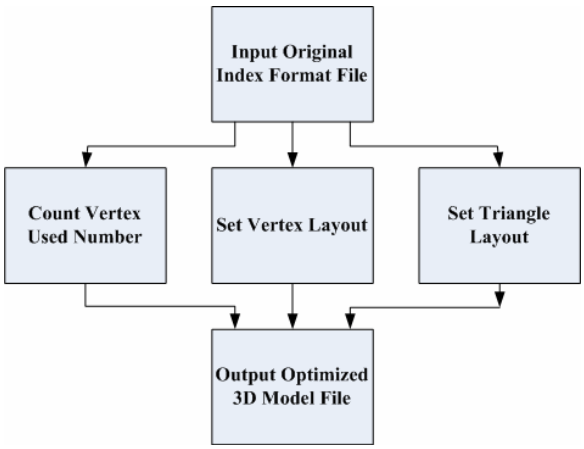

Figure 3. The flowchart of the new algorithm.

3.1 Initialization

In the first stage, a record is allocated and initialized for each vertex during the reading of the input mesh, which comprises the following fields.

k_v: the key value of the vertex v.
d_v: the vertex connectivity of v.
i_v: the new index of the vertex v in the input file;
(x_v, y_v, z_v): the geometric coordinates of the vertex v.

For each triangle, a record comprising the following four fields are computed.

k_t: the key of triangle t,
(v_1, v_2, v_3): respectively represents the vertex indices of the three vertex of the triangle.

3.2 Vertex Layout

As suggested by [4], to solve for k_v and k_t, one must assume that either the vertex or triangle layout is explicitly specified.

On the basis of the original triangle layout of the input mesh, we may find an optimized vertex layout by sorting the vertices according to the sort key k_v where a function $k_v=f(v)$ is defined to given a unique sort key for each vertex with respect to a particular sequencing criteria. If a preordered mesh where the vertex information must be known before the presence of the first triangle referring to this vertex is preferred, the sort key k_v is computed as follows.

$$k_v = \min\{ f(v) \mid v \in t\} \tag{1}$$

To get the vertex layout, we have tried two types of $f(v)$. The first one suggests a geometrical sort on the incrementing order of the coordinate of X-axis, which implies that the key value of each vertex is calculated by letting

$$f(v) = x_v, \quad \text{if} \quad v = (x_v, y_v, z_v) \in R^3 \tag{2}$$

The second way is by a topological sort according to a given geodesic function $G(v)$. Instead of measuring the distance along the surfaces, we suggest simplifying the distance calculation by finding the single source all destination shortest paths.

3.3 Set Triangle Layout

Afterwards, a compatible triangle layout can be computed by sorting the faces according to the new indices of the vertices and the sort key k_t computed as follows.

$$k_t = \max\{I_v \mid v \in t\} \tag{3}$$

3.4 Finalization and Output

After optimizing the vertex and triangle layouts, a streaming mesh is derived by traversing the faces according to the new layout. Upon visiting a triangle, the newly referred vertex is introduced prior to the output of the visiting face.

The visited triangle face is then output to the stream as a triplet of the three indices to the vertices in the in-core vertex buffer. Afterwards, for each referred vertex, the reference count specified by d_v is decremented. When the count of a referred vertex reaches zero, the vertex is then removed from the in-core vertex buffer.

4. Experimental Results

We may evaluate an optimized stream intuitively by examining its layout diagram or by the two statistics suggested by [4].

The first is called the *front width*, or *width*, of the mesh, which is essentially the maximum size of the in-core vertex buffer.

The second is called the *front span*, or *span*, of the mesh, which intuitively measures the longest period of the vertex life in the in-core vertex buffer or the maximum difference of the vertex indices. Hence,

$$width = \max_i\{|F_i|\}, \tag{4}$$
$$span = \max_i\{\max F_i - \min F_i + 1\}, \tag{5}$$

where F_i is the evolving set of active vertices.

The test meshes are collected from the Internet. The statistics of these meshes are listed as follows.

Table 1. The statistics of the test meshes.

Model	# of vertices	# of triangles
Bunny	35,947	69,451
Armadillo	172,974	345,944
Dragon	437,645	871,414

The results listed in Table 2 show the widths and spans of the optimized streaming meshes by our method. In Table 2, V and T respectively represent the results by merely optimizing the vertex and the triangle layouts. In addition, X and G represent the sorting criteria: X means that the sorting is on incrementing order of the X-axis coordinates of the vertices and G is on the geodesic distances.

Table 2. The evaluation of the results (width/span)

Model	V	T	X	G
Bunny	9133/ 34550	3119/ 34641	412/ 1503	579/ 687
Armadillo	51951/ 172714	16553/ 171974	950/ 2753	1532/ 2320
Dragon	4586/ 54825	3847/ 434398	1330/ 9191	2759/ 3862

For better results, it is desirable to have the width and span of the optimized mesh as low as possible. From Table 2, it is obvious that X and G generates much better results relative to V and T, which directly implies that X and G use much less memory.

5. Conclusion

From the experimental results, it is evident that the new method is able to effectively optimize an input mesh and generate optimized streaming mesh by interleaving the vertices and the faces as those suggested by [4].

With much lower width and span, the optimized sequence has much smaller memory footprint and better data coherency that may significantly improves the efficiency of common out-of-core techniques by reducing the number of disk accesses and I/O cache miss.

References

[1] H. Hoppe. "Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering." Visualization ’98, 35–42.

[2] J. Ho, K. Lee, and D. Kriegman. "Compressing Large Polygonal Models." Visualization ’01, 357–362

[3] J. Wu and L. Kobbelt. "A Stream Algorithm for the Decimation of Massive Meshes." Graphics Interface ’03, 185–192."

[4] M. Isenburg and P.Lindstrom. "Streaming Meshes." In Proc. of IEEE Visualization (2005), pp. 231–238.

[5] M. Isenburg and S. Gumhold. "Out-of-Core Compression for Gigantic Polygon Meshes." SIGGRAPH 2003, 935–942.

[6] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. "Large Mesh Simplification using Processing Sequences." Visualization ’03, 465–72.

[7] M. Isengurg, P. Lindstrom, and J. Snoeyink. "Streaming compression of triangle meshes." In Proceedings of the 3rd Eurographics symposium on Geometry,2005.

[8] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. "External Memory Management and Simplification of Huge Meshes." IEEE Transactions on Visualization and Computer Graphics, 9(4):525–537, 2003.

[9] Y. Chiang, J. El-Sana, P. Lindstrom, and C. Silva. "Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics." In IEEE Visualization Tutorial T4, 2002.