The Storage Independent Polygonal Mesh Simplification System

Hung-Kuang Chen[1], Chin-Shyurng Fahn[2], and Ming-Bo Lin[3]

[1] Dept. of Electronic Engineering, National Chin-Yi University of Technology, Taichung, Taiwan, ROC.
[2] Dept. of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C.
[3] Dept. of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C.
hankchentw@gmail.com, csfahn@ntust.edu.tw, mblin@et.ntust.edu.tw

Abstract. Traditional simplification algorithms operate either in the main memory or in the disk space. In this paper, we propose a novel storage independent polygonal mesh simplification system (SIPMSS). The new system offers an advanced memory utilization scheme and efficient storage-independent primitives that achieve constant main memory footprint and excellent runtime efficiency. It not only provides a very flexible platform for both in-core and out-of-core simplification algorithms but also permits very efficient integration of various types of simplification algorithms. The results presented in the section of experiments further approve its effectiveness and usefulness for both in-core and out-of-core of simplification.

1 Introduction

The polygonal mesh has been the de facto standard representation of 3D objects owing to its mathematical simplicity and the direct support of rendering hardware [1]. For this reason, the majority of graphics algorithms for manipulating surfaces are designed for polygonal meshes, and most contemporary methods for model acquisition start with the construction of polygonal meshes. However, the above representation has an inherent limitation to the realism of a rendered scene imposed by the maximum number of polygons that can be processed by the rendering hardware at an interactive rate.

To overcome such a limitation, one can either minimize the number of geometric/topological primitives of each object within an allowable range of approximation errors or maximize the use of limited rendering resources by adaptively adjusting the resolution of each object according to their contribution to the rendered image. The former can be achieved by applying polygonal mesh simplification to automatically generate an optimized approximation of an object. The latter is essentially the basic concept of the level of detail (LOD) scheme originally suggested by Clark [2], which usually requires the generation of LOD representations through the use of polygonal mesh simplification.

R. Shumaker (Ed.): Virtual Reality, HCII 2007, LNCS 4563, pp. 3–12, 2007.

Most traditional polygonal mesh simplification algorithms operate either in the main memory space or the disk space. Those who place their data inside the main memory space are called in-core algorithms; on the contrary, those who place their data in the external memory space such as disks are called out-of-core algorithms. In general, in-core algorithms featuring with high performance memory access efficiency mostly perform memory access intensive complex operations with a higher degree of optimization [3-6]. Hence, most of these algorithms are capable of producing very good quality outputs. However, owing to the limited main memory space, this type of algorithms is not able to deal with high-resolution scanned objects with a huge amount of polygons.

On the other hand, out-of-core algorithms mostly apply a spatial partitioning scheme to lower the memory access frequency [7,8]. Among these algorithms, the grid-based vertex re-sampling scheme has the best run-time efficiency. Nevertheless, it comes at the cost of inferior output quality when the input mesh is sampled with a low resolution grid. For high-resolution outputs using high-resolution grids, the runtime efficiency of such a method decreases radically. For instance, if an $n \times n \times n$ grid is used for re-sampling the mesh, both the time and space complexities are $O(n^3)$.

A compromise approach proposed by Garland et al. suggested a two-phase simplification that begins with a uniform-grid out-of-core vertex-clustering phase similar to the OoCS method proposed by Lindstrom et al. followed by an in-core iterative edge collapse phase [9]. In principle, the memory cost of their method is related to the grid resolution of the first phase that affects the quality of approximations in the first phase. Lower resolution may greatly reduce the quality of resulting approximations. However, higher resolution may require much more memory space. The increase of grid resolution may raise the memory cost by $O(n^3)$.

In this paper, we propose a novel storage independent polygonal mesh simplification system (SIPMSS). The new system offers an advanced memory utilization scheme and efficient storage-independent primitives that achieve constant main memory footprint and excellent runtime efficiency. It not only provides a very flexible platform for both in-core and out-of-core simplification algorithms, but also permits very efficient integration of various types of simplification algorithms.

2 System Overview

To provide a storage independent environment for various types of polygonal mesh simplification algorithms, we extend the idea from our cache-based approach [10] by

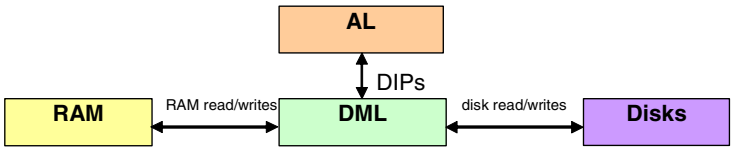

Fig. 1. The storage independent polygonal mesh simplification system

dividing the simplification system into two layers: the *algorithm layer* (AL) and *data management layer* (DML). A block diagram of this system is shown in Fig. 1.

The simplification algorithms are implemented in the AL where a set of *device independent primitives* (DIPs) is supported by the DML regardless of the physical location of the data. To enable device independent simplification, the DML has to manage the allocation of the storage space, accesses to different types of storage devices, and configures the operation mode according to the memory requirement of the simplification layer and the available system memory space. A detailed discussion of the two layers is given in the following two subsections.

2.1 The Initialization Flows

Prior to the start of the simplification, the AL begins with the requests sent to the DML through the DIPs describing the basic data structures, usable disk directories, and the amount of available system memory. On the basis of such a profile, the DML decides the operation mode by comparing the estimated main memory cost with the amount of allocated main memory space. If the allocated main memory space is larger than the estimated memory cost, the simplification is performed in core; conversely, it will be operated under the cache-based out-of-core mode.

2.2 The Operation Modes

The new system is able to operate under two operation modes: the in-core and out-of-core operation modes. The system operates under the in-core operation mode only when the memory requirement of the AL is lower than the available main memory space; otherwise, it runs under the out-of-core operation mode. During in-core operations, the DML puts all the data of the AL in the main memory buffer. Accesses to the data are re-directed to the main memory blocks.

By contrast, under the out-of-core mode, the DML allocates all the data to the available disks and uses the available system memory as cache blocks. During the simplification, the AL carries out the simplification using the read/write DIPs supported by the DML. To serve the requests, the DML first searches the required data from the cache blocks in the main memory space. If the search completes successfully, the required data is sent from the cache blocks in the main memory space; otherwise, the request is re-directed to the disks and the required data is read from the disk files. The two basic operation states are illustrated in Fig. 2.

Since the memory cost of the AL decreases as the mesh simplifies, it is possible to allow in-core simplification after a series of simplifications. Hence, the simplification can be performed in the combination of in-core and out-of-core modes where the simplification starts with the out-of-core mode then switches to the in-core operation mode when the memory cost of the AL is less than the size of available in-core buffers. Such combination is enabled by periodically issuing a request to reconfigure the operation mode by the AL, which may further improve the runtime efficiency.

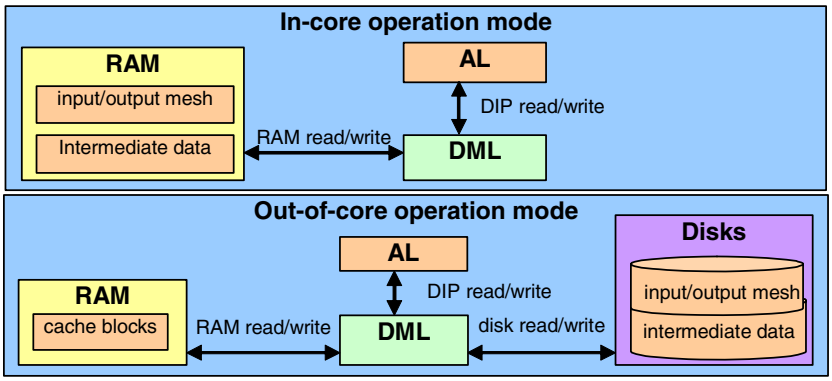

Fig. 2. The two operation modes: the in-core operation and out-of-core operation modes

3 The Data Management Layer

The data management layer (DML) provides a set of device-independent primitives to the algorithm layer (AL), which abstracts the allocation, de-allocation, and read/write accesses to all the data sets.

3.1 The Device Independent Primitives

We intend to provide three classes of primitives:

- *The operation mode configuration primitives*: they are employed to determine or reset the operation modes.
  - **int Config**(ProfileT <profile>) : it calculates the maximum memory cost according to the table of data structures in the initial system profile and determines the operation mode according to the following rules: let the bound to the main memory cost be *B* and the memory cost is *C*. The DML operates under the in-core simplification mode if $C \le B$; otherwise, it operates under the out-of-core mode.
- *The storage allocation/de-allocation primitives*: they are employed to set up the buffers and files for the data structures used by the simplification layer.
  - **int OpenBlock**(**int** <element size>, **int** <no of elements>): it allocates a file or a memory block of <element size>×<no of elements> bytes and returns the block ID.
  - **void CloseBlock**(**int** <block ID>): it releases the memory and file of the block specified by the block ID.
- *The data read/*write *primitives*: they are used for accessing the content of these data structures either from the buffers or from the disks.
  - **void read**(**int** <block ID>, **int** <element size>, **int** <element index>, **char** *<destination>): it reads an element from a file or a main memory block specified by the block ID.
  - **void write**(**int** <blockid>, **int** <element size>, **int** <element *index*>, **char** *<source>): it writes an element to a file or a main memory block specified by the block ID.

4 The Algorithm Layer

The AL is essentially an implementation of the simplification algorithm using the DIPs supported by the data management layer. To show how the new system works, we have implemented two types of polygonal mesh simplification algorithms on this new system, namely Algorithm I and Algorithm II. The former, Algorithm I, is an implementation of a linear-time in-core iterative edge collapse algorithm proposed in our precious work [11]. The latter, Algorithm II, is a variation of the famous OoCS algorithm performing out-of-core uniform-grid vertex re-sampling [7].

The memory footprint of Algorithm I is large; hence, in its original implementation, the simplification of large meshes requires an enormous amount of system memory. Algorithm II is designed for efficient out-of-core simplification; however, it is output sensitive and assumed to accept the STL format rather than the common index-faced format. Both of them adopt the quadric error metrics proposed by [4].

With our new system, both Algorithms I and II are possible to perform simplification on large meshes with constant main memory footprint. Furthermore, with the automatic detection and adjustment of the operation modes, we do not need a precise estimate of the output size to allow the simplification in the second phase when the multiphase simplification approach is applied.

4.1 Algorithm I– A Linear-Time Iterative Edge Collapse Algorithm

Algorithm I is an in-core iterative full edge collapse polygonal mesh simplification algorithm. In the main, the algorithm is composed of three stages: the preprocessing stage, the simplification stage, and the output stage. In the preprocessing stage, the vertex rings are constructed through a single pass over the faces of the input mesh. Upon processing each face, the face index is inserted to the incident face lists of the three vertices. Following the preprocessing stage, the simplification stage and the output stage are repeatedly performed until the termination condition is satisfied. Every pass of the loop outputs a new level of approximation. Thus, if k iterations are executed, the algorithm may produce k LODs.

4.2 Algorithm II– A Uniform-Grid Vertex Clustering Algorithm

The second type of algorithms is designed for radical out-of-core simplification [7], which comprises three phases: vertex re-sampling, face re-sampling and grid quadric calculation, and the calculation of representative vertices. The first phase re-samples all the vertices with a uniform grid. This gives the mapping between a vertex and a grid ID.

After the re-sampling of vertices, the algorithm re-samples the faces by outputting only those faces whose three vertices are sampled to distinct grids. In the meantime, it also calculates grid quadrics by summing all the face quadrics sampled to each grid. In the finale phase, an optimal placement of the representative vertex of each grid is calculated by solving the linear system presented by the grid

quadrics. In the original work, a set of rules is suggested to prevent from numerical errors. We do not employ those rules in this implementation. Rather, we simply use one of the sampled vertices of a grid if the solved position is out of the range of such a grid.

5 Experimental Results

The storage independent polygonal mesh simplification system (SIPMSS) has been implemented with C++ codes on a PC equipped with an Intel Pentium 4 2.2 GHz CPU, 1 GB RAM, and a RAID disk system of two IDE 7,200 rpm hard drives. The operating system and the compiler are the Microsoft Windows 2000 and Visual C++ 6.0, respectively. The test models are downloaded from the Stanford Scanning Repository. A summary of the test models is listed in Table 1.

Table 1. A summary of the test models

Model	Dragon	David (2mm)	Lucy
Number of vertices	437,645	3,614,098	14,027,872
Number of faces	871,414	7,227,031	28,055,742

To determine whether the system is successful, we may evaluate the new system from three perspectives. First, we have to verify if our new system indeed improves the performance of the out-of-core algorithm (OoCS) by comparing the running times of the OoCSx with those of Algorithm II. Second, we also have to prove that the implementation of the in-core algorithm, Algorithm I, is able to simplify large meshes with a constant amount of main memory footprint. Third, how does the multiphase approach work in the new system? To provide a basis of comparison, the running times of the HQMS [11] and OoCSx [8] for the simplification of the three meshes are shown in Table 2.

Table 2. The running times of the HQMS and OoCSx by seconds

Algorithm	Model		
	Dragon	**David (2mm)**	**Lucy**
HQMS	7	66	-
OoCSx[1]	22	284	1,414
[1]Our implementation of the OoCSx.			

The performance of the new system under the in-core operation mode running the Algorithms I, II, and the two-stage approach are respectively presented in Table 3 where the results are compared with their original version and the QSLIM V2.0.

Table 3. The running times of Algorithms I, II, and their two-stage combination in the in-core mode by seconds

Algorithm		Model		
		Dragon	**David (2mm)**	**Lucy**
Algorithm I (o. s.=10K Δs)[1,2,3]		10	81	2,284[4]
Algorithm II (m. g. d. =50)[5]		5	38	148
Two-stage	**Stage 1: (m. g. d. =300)[5]**	6	33	125
	Stage 2: (o. s.=10K Δs)[1,2,3]	3	2	3
[1]o. s. = output size.; [2]1K=1,000.; [3]Δs = triangles; [4]The system memory is not enough for in-core mode simplification; [5]m. g. d. = maximum grid dimension.				

The test of the out-of-core mode is conducted by restricting the main memory size to be below 8 MB, which is much less than the main memory size of a desktop PC that usually has more than 256 MB main memory space. Under such a strict condition, the simplification of all the three models is executed using the out-of-core operation mode. The test results are given in Table 4.

Table 4. The running times of Algorithms I, II, and their two-stage combination in the out-of-core modes by seconds

Algorithm		Model		
		Dragon	**David (2mm)**	**Lucy**
Algorithm I (o. s.=10K Δs)[1,2,3]		23	362	2,931
Algorithm II (m. g. d. =50)		6	58	217
Two-stage	**Stage 1: (m. g. d. =300)[4]**	20	49	219
	Stage 2: (o. s.=10K Δs)[1,2,3]	8	5	10
[1]o. s. = output size; [2]1K=1,000.; [3]Δs = triangles; [4]m. g. d. = maximum grid dimension.				

To show the integration of different types of algorithms is possible, an experiment is carried out to test the performance of applying a two-stage integration called Algorithm Two-Stage. The Algorithm Two-Stage adopts the idea from the multiphase approach proposed in [7] and is a two-stage method integrating Algorithm I and II by applying Algorithm II at the first stage followed by Algorithm I.

In addition to the running times, the rendered images of the simplified Lucy meshes are demonstrated in Fig. 3.

According to Tables 2-4, Algorithm II takes only 217 seconds to simplify the Lucy mesh with 8 MB RAM while its original memory insensitive version, OoCSx, takes 1,414 seconds. Hence, the running time of the out-of-core algorithm improves

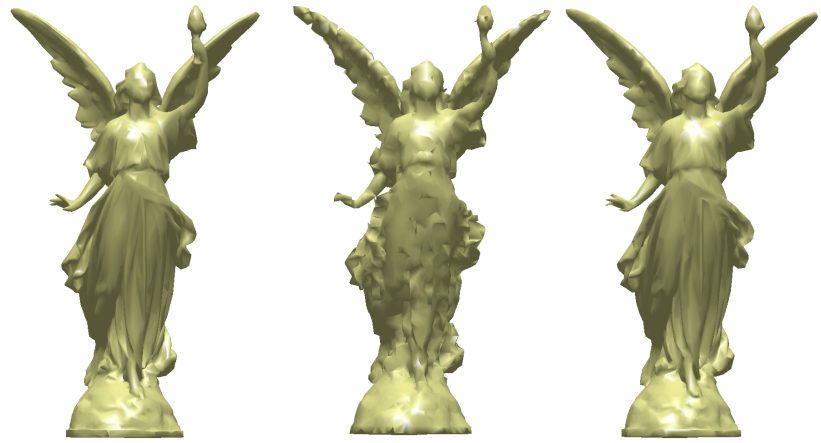

Fig. 3. The simplified Lucy meshes generated by Algorithm I, II, and Two-Stage

significantly with our new system. Also from Tables 2-4, we may notice that Algorithm I, the implementation of the HQMS algorithm, is able to simplify the Lucy mesh with only 8 MB main memory, which approves our statements as expected.

Furthermore, simplification results shown in Tables 3 and 4 prove that the two-stage integration is not only possible but also successful. With 512 MB system memory, Algorithm Two-Stage is capable of simplifying the Lucy mesh in the in-core mode using 128 seconds. With 8 MB RAM, it successfully simplifies the Lucy mesh using only 229 seconds without manual intervention.

6 Discussions and Conclusions

In this paper, we have addressed the issue of high-performance large mesh simplification by presenting a storage independent platform for the implementation and integration of various types of in-core and out-of-core polygonal mesh simplification algorithms. The proposed approach successfully ensures a constant main memory cost, unbounded input mesh size, high external memory access efficiency, and permits tight integration of various types of polygonal mesh simplification algorithms. The results presented in the experiments section further approve both its effectiveness and usefulness for the two types of simplification.

In summary, the experimental results verify our statements with the following facts:

- The Algorithm I, or the implementation of an in-core iterative edge collapse method (HQMS) on the SIPMSS, is capable of simplifying large meshes with a constant main memory cost.

- The Algorithm II, or the implementation of an out-of-core grid-based re-sampling method (OoCS) on the SIPMSS, has significantly better runtime efficiency than its original version.
- The integration of Algorithms I and II, or the two-stage approach, having the advantages of high runtime efficiency and good output quality proves that the SIPMSS is a very good system for integrating various types of simplification algorithms.
- Despite the aforementioned achievements, a number of issues are to be studied in the future.
- For some data structures of variable lengths, they could not be supported by our system. Hence, it is not feasible to port these algorithms directly to our system.
- The policy for optimum allocation of buffers to each data structure has not been studied in this paper. The amount of allocation is decided by the weight of the data structure, which is simply given by the ratio of its file size to the average file size.

Acknowledgments. We would like to thank the Stanford 3D Scanning Repository for providing all the test models. This work was supported by the National Science Council of Republic of China under the contract number NSC-95-2218-E-046-003.

References

1. Lubke, D., Reddy, M., Cohen, J.D., Varshney, A., Watson, B., Huebner, R.: Level-of-Detail for 3D Graphics. Morgan Kaufman, San Fransisco, California (2003)
2. Clark, J.H.: Hierarchical geometric models for visible surface algorithms. Communications of the ACM 19(10), 547–554 (1976)
3. Hoppe, H.: Progressive meshes. In: Proceedings of the SIGGRAPH ’96, vol. 30, pp. 99–108 (1996)
4. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proceedings of the SIGGRAPH ’96, vol. 30, pp. 209–216 (1996)
5. Lindstrom, P., Turk, G.: Evaluation of memoryless simplification. IEEE Transactions on Visualization and Computer Graphics 5(2), 98–115 (1999)
6. Wu, J., Kobbelt, L.: A stream algorithm for the decimation of massive meshes. In: Proceedings of the Graphics Interface ’03, pp. 185–192 (2003)
7. Lindstrom, P.: Out-of-core simplification of large polygonal models. In: Proceedings of the SIGGRAPH ’00, vol. 34, pp. 259–270 (2000)
8. Lindstrom, P., Silva, C.T.: A memory insensitive technique for large model simplification. In: Proceedings of the IEEE Visualization ’01, pp. 121–126 (2001)
9. Garland, M., Shaffer, E.: A multiphase approach to efficient surface simplification. In: Proceedings of the IEEE Visualization ’02, pp. 117–124 (2002)
10. Chen, H.K., Fahn, C.S., Tsai, J.P., Lin, M.B.: A novel cache-based approach to large polygonal mesh simplification. Journal of Information Science and Engineering 22(4), 843–861 (2006)
11. Chen, H.K., Fahn, C.S., Tsai, J.P., Chen, R.M., Lin, M.B.: A linear time algorithm for high quality mesh simplification. In: Proceeding of the IEEE 6th International Symposium on Multimedia Software Engineering, pp. 169–176 (2004)

12. Chen, H.K., Fahn, C.S., Tsai, J.P., Chen, R.M., Lin, M.B.: Generating high-quality discrete LOD meshes for 3D computer games in linear time. Multimedia Systems 11(5), 480–494 (2006)
13. Carr, R.W., Hennessy, J.L.: WSClock– A simple and effective algorithm for virtual memory management. In: Proceedings of the Symposium on Operating Systems Principles ’81, pp. 87–95 (1981)