Semi-Regular Remeshing of Out-of-Core Meshes [1]

Minsu Ahn[†] **Igor Guskov**[‡] **Seungyong Lee**[†]

[†]*Dept. of Computer Science & Eng., POSTECH, Pohang, Korea*
[‡]*Dept. of Electrical Eng. & Computer Science, Univ. of Michigan, Ann Arbor, USA*

Abstract

This paper proposes an out-of-core method for creating semi-regular surface representations from large input surface meshes. Our approach is based on a streaming implementation of the MAPS remesher of Lee et al. [10]. Our remeshing procedure consists of two stages. First, a simplification process is used to obtain the base domain. During simplification, we maintain the mapping information between the input and the simplified meshes. The second stage of remeshing uses the mapping information to produce samples of the output semi-regular mesh. The out-of-core operation of our method is enabled by the synchronous streaming of a simplified mesh and the mapping information stored at the original vertices. The synchronicity of two streaming buffers is maintained using a specially designed write strategy for each buffer. Experimental results demonstrate the remeshing performance of the proposed method, as well as other applications that use the created mapping between the simplified and the original surface representations.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Curve, surface, solid, and object representations

1 Introduction

Modern 3D acquisition technology can easily produce huge surface meshes with high precision and fine detail. The medical visualization and computational simulation applications can also be sources of extremely large surface meshes. Data structures needed for efficient and complex processing of such meshes may not always fit into the main memory and a special class of *out-of-core* mesh processing algorithms has been introduced in the recent years.

Many of the huge surface meshes found in practical applications are relatively smooth and can therefore benefit from conversion into a semi-regular form [4, 9, 12]. Unfortunately, most of the currently available remeshing procedures cannot deal with extremely large meshes. In this paper, we introduce an out-of-core approach to remeshing.

As is common in many other semi-regular remeshing approaches, our out-of-core remeshing algorithm builds a base mesh which serves as the coarsest level of the multiresolution output mesh, and parameterizes the input mesh so that each input vertex corresponds to a point on the base mesh. Our approach is based on the MAPS algorithm of Lee et al. [10]. We use a streaming mesh strategy to enable out-of-core processing of large input meshes. That is, an out-of-core streaming mesh and the accompanying mapping information file are maintained in-sync during the simplification stage of the MAPS algorithm in order to build the parameterization map. The final resampling stage uses the parameterization map to produce a semi-regular surface mesh approximating the input surface.

Our main contribution is to introduce an out-of-core semi-regular surface remeshing algorithm that can establish a correspondence between a coarse base domain and a very large input surface mesh. We test our out-of-core remeshing procedure on a number of large meshes and explore the approximation and compression performance of our algorithm.

2 Related Work

There exist several approaches for out-of-core mesh processing, such as cutting the mesh into pieces [6], clustering mesh vertices [11], using external memory mesh data structures [2], and processing stream data [7, 14].

Recent out-of-core surface processing approaches use *streaming representations* [7, 14]. The basic idea of a streaming algorithm is that the input meshes stream from the input files without reading backward and a partial input mesh is stored in the main memory. Our approach also uses streaming, and the main challenge here is the synchronization of the two meshes (original and simplified) which are at two different levels of resolution.

Over the past ten years, several remeshing algorithms have been proposed. One group of approaches is based on simplification and subdivision techniques of 3D meshes [5, 8, 10]. Another approach uses a global parameterization of the input meshes for remeshing [1, 4, 12]. More recent algorithms directly manipulate the input meshes without global parameterization [13].

However, these techniques are the remeshing algorithms

[1] Extended abstract: The full paper was published at IEEE Visualization 2006.

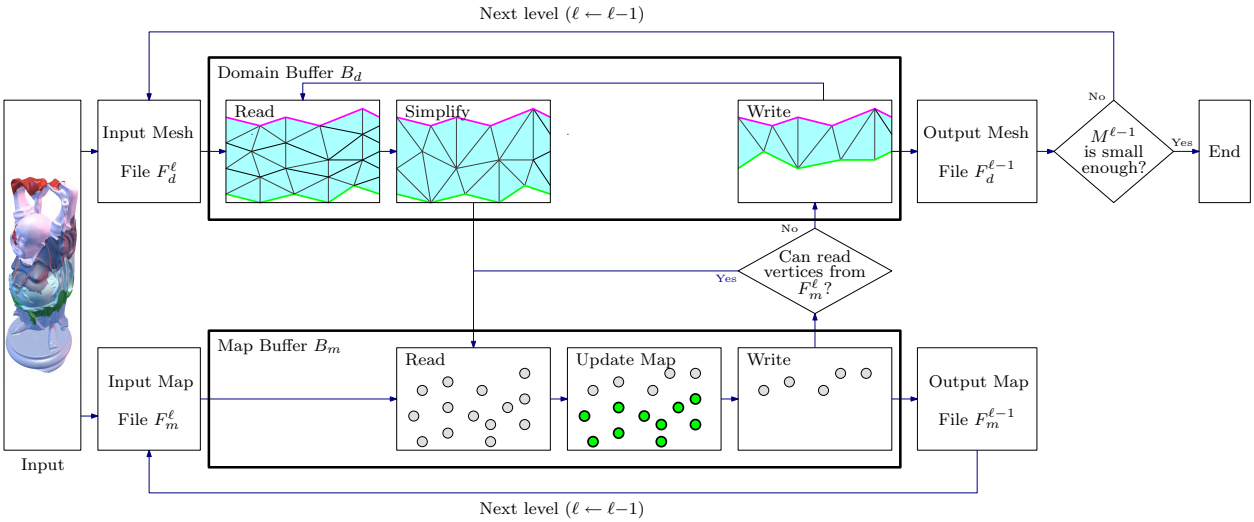


Figure 1: Overall parameterization process operates on the simplified domain mesh and the vertex file storing the parametric mapping information. We repeat three steps in each buffer; read, simplify, and write partial meshes in the domain buffer and read, update, and write the mapping information in the map buffer.

for in-core meshes which assume that input meshes are sufficiently small to fit into the main memory. Our goal is to produce an out-of-core remeshing procedure for very large input meshes.

3 Overall Process

Our out-of-core semi-regular remeshing technique is based on the MAPS algorithm [10]. However, we cannot directly apply this algorithm for remeshing when a given mesh M is an out-of-core mesh that cannot fit into the main memory. Since the data structures for M and its parameterization information cannot be stored in memory, we have to revise each step of the MAPS algorithm. For simplification, we adapt a streaming-based approach [14] where a mesh can be simplified iteratively with a fixed-size buffer and a streaming file. To maintain and update the parametric mapping information, we store the information in a map file and stream it through a buffer in a synchronized way with simplification. The vertex position resampling for a semi-regular mesh is achieved by traversing the parameterization stored in the final map file.

Fig. 1 illustrates the overall process for obtaining the parameterization. In the process, the original mesh $M = M^L$ is successively simplified to a series of homeomorphic meshes M^ℓ with $0 \leq \ell \leq L$ until the base mesh M^0 is derived. For simplification and parameterization information updates, we use two buffers: the *domain buffer* and the *map buffer*. The domain buffer is used to contain a partial mesh of M^ℓ when we simplify M^ℓ to $M^{\ell-1}$ using mesh streaming. The map buffer is used to store and update the parameterization information for the part of M^ℓ which is being simplified. Each simplified mesh M^ℓ is stored in a *mesh file* F_d^ℓ. The parameterization information that maps the vertices of M onto the surface of M^ℓ is stored in a *map file* F_m^ℓ.

As a result of the parameterization process, the mapping information from the given mesh M onto the base mesh M^0 is obtained. In the resampling process, we first determine the vertex set and connectivity of a semi-regular mesh by recursively subdividing the base mesh M^0. The vertex positions of the semi-regular mesh are obtained by resampling the shape of M using the mapping information.

4 Parameterization

In the parameterization process, we establish the mapping between the input mesh M^L and the base mesh M^0. Initially, the parameterization is the identity; each vertex of M^L is mapped onto itself. When we simplify the mesh M^ℓ to $M^{\ell-1}$, we should compute the mapping of the vertices onto the surface of $M^{\ell-1}$ from the mapping onto M^ℓ.

For the parameterization update, we follow the strategy proposed in MAPS [10]. We use vertex removal for simplification and use a conformal map for a bijection map in 2D between the 1-ring neighborhood of a removed vertex and its retriangulation after vertex removal. However, we should adapt the strategy to out-of-core processing because the mapping information for all the vertices of M cannot be kept in the main memory. This is the reason why we use the map buffer B_m to keep and process parts of the mapping information in a streaming fashion.

When we simplify the mesh M^ℓ to $M^{\ell-1}$, the buffer B_m may not contain all the vertices that are mapped onto the simplified region because some of the vertices may be read after simplification. In this case, we should store the conformal map of the removed vertex in a hash table H_{2D} and use it later to update the vertices that will be read after

simplification.

Similarly, the domain buffer B_d contains a part of the simplified mesh M^ℓ. When we read a vertex v_m of M into the map buffer B_m, we may not find the face of M^ℓ onto which v_m is mapped in the domain buffer B_d. This occurs when the face has not yet been read or has been removed by *half-edge collapse*. If the face has not yet been read, we keep the vertex v_m in B_m until we read the face. To check whether a face of M^ℓ has been removed, we maintain the indices of removed faces in a hash table H_f. The hash tables temporarily store the information related to the removed vertex until we finish updating all mapping information around the removed vertex.

5 Buffer Maintenance

In the parameterization process, we use two limited-size buffers, the domain and map buffers. When we use a limited-size buffer, the main concern is how to maintain the buffer. To avoid buffer overflow, we must write and remove the processed part before we read the next part. However, we cannot write and remove faces in an arbitrary order. A face f in the domain buffer B_d cannot be written and removed until we completely update the mapping information for the vertices of M which are mapped onto f. Similarly, a vertex v in the map buffer B_m becomes writable after we have updated its mapping information.

If no vertex in the map buffer B_m finds its mapping face in the domain buffer B_d and no face in B_d is writable, then we can neither remove any vertex nor face from the buffers. In this case, other vertices and faces cannot be read from the files and the parameterization process cannot be continued. To prevent this situation, we manage the mesh parts contained in the buffers to overlap each other. If the mesh parts in the buffers have some overlapping regions, a vertex in B_m will find the mapping face in B_d for the mapping information update, which helps the faces in B_d become writable.

6 Resampling

After the parameterization process, we can obtain a semi-regular mesh by recursively subdividing the base mesh M^0 and resampling the positions of new vertices from the given mesh M. However, since the mesh M cannot fit into the main memory, we need an out-of-core algorithm for resampling the new vertex positions.

For resampling, we first decompose the triangles into sets T_i of triangles that are contained in the same faces $f_i \in M^0$. In Fig. 2, the large triangle represents a face in M^0 and the two small triangles are from M. Also in each set T_i, triangles are sorted externally from bottom to top. Finally, we sequentially read triangles and check whether each triangle contains a sampling point or not. When a triangle contains a sampling point, the position of the sampling point is determined by barycentric coordinates.

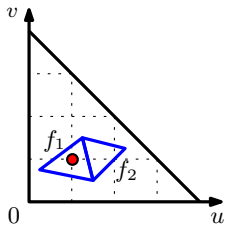


Figure 2: Resampling of vertex positions

7 Experimental Results and Applications

7.1 Semi-Regular Remeshing

We first explore the semi-regular remeshing performance of our algorithm on several models. Fig. 3 shows the remeshed models with zoomed-in details showing the structures of the resulting semi-regular meshes at level 4, with each regular patch individually colored. For all of the examples, we achieve good visual approximation quality and a bijective mapping. Table 1 presents the key statistics of the input models with the remeshing time, output semi-regular mesh complexity, and errors measured by the Metro tool [3].

Our streaming algorithm has the sizes of the main memory buffers as parameters. In all of our experiments, we set the maximum face count for the domain buffer B_d, $\mathrm{Max}_f(B_d)$, to be 400K and the maximum vertex count for the map buffer B_m, $\mathrm{Max}_v(B_m)$, to be 10K. Our method with these sizes of buffers used less than 230MB of RAM for all of our experiments. Fig. 4 shows the memory footprint for the Lucy model. In Fig. 4, we used only 31MB of RAM when $\mathrm{Max}_f(B_d) = 50\text{K}$ and $\mathrm{Max}_v(B_m) = 10\text{K}$.

7.2 Map Conversion

For a very large input model, the parameterization process can be split into two stages. We first run our out-of-core parameterization procedure on the large input mesh and create an intermediate base mesh together with a mapping between the intermediate mesh and the original input mesh. Then, we use an in-core parameterization procedure to parameterize the intermediate base mesh onto the coarse base mesh. Using this idea, for example, we can create a spherical parameterization of a large input mesh. Fig. 5 shows a geometry image and a normal-map image of the given out-of-core mesh by the spherical parameterization [12].

7.3 User-Guided Mesh Editing

The created correspondence between the original and the base domain meshes can be used for simple editing of the surface. In particular, the user can remove parts of the base mesh and have the corresponding parts of the original surface removed as well. For example, Fig. 6(a) shows the artifacts unrelated to the input shape. Since we have a bijection between the original mesh M and its parametric domain M^0, we can remove the artifacts as shown in Fig. 6(c).

models	original # vert.	base # vert.	param. time	resample time	E_r (10^{-4})	archive size (byte)	bit rate (bit/v)	E_r (10^{-4})
david	274,831	204	3m 28s	2m 16s	0.95	87,442	2.54	1.09
happy Buddha	541,366	204	7m 43s	3m 27s	0.89	88,243	1.30	1.75
david (head)	1,931,794	64	27m 59s	24m 27s	1.21	67,957	0.30	2.23
XYZ RGB dragon	3,609,455	22	56m 28s	51m 59s	0.86	125,691	0.28	1.41
XYZ RGB statuette	4,999,996	246	1h 18m 37s	1h 10m 13s	1.07	174,217	0.28	2.37
lucy	14,027,868	214	3h 56m 38s	2h 56m 49s	0.69	97,612	0.06	1.25

Table 1: Statistics for the remeshed models: The first six columns are the model name, the number of vertices in the original mesh, the number of vertices in the base domain, the parameterization time, the resampling time, and the L^2 error of the remeshed model (w.r.t. the bounding box diagonal of the original model). The remesh data are reported for the level five of the output semi-regular mesh. The last three columns show the compression statistics using the progressive geometry coder [9]; both the archive size and the bit per vertex are shown, followed by the relative L^2 error of the decompressed model. The default compression settings with 12 bit-planes were used in the PGC software [9] with Butterfly wavelets.

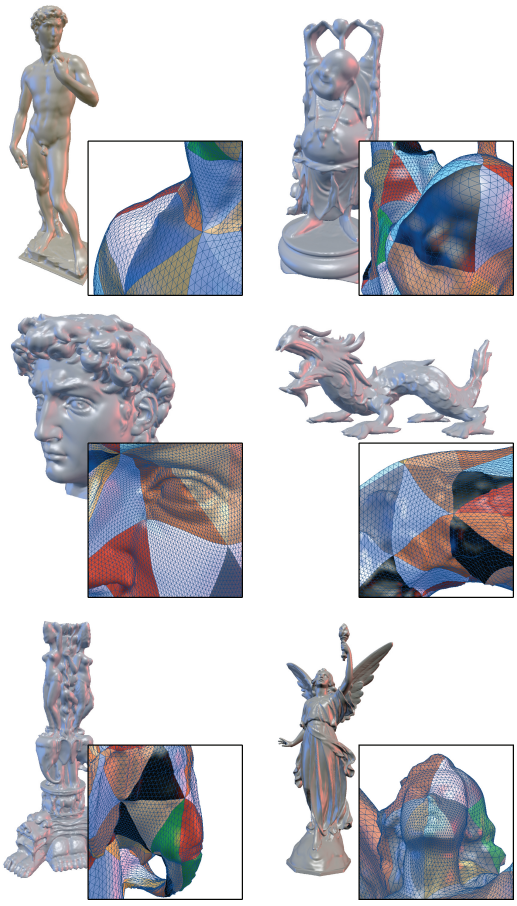


Figure 3: Remeshing examples

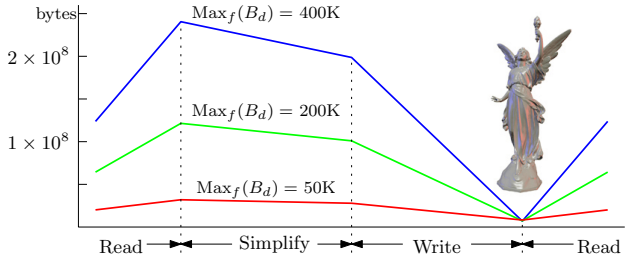


Figure 4: Memory footprint for the Lucy model with the different sizes of the domain buffer B_d and the map buffer size of $\mathrm{Max}_v(B_m) = 10K$

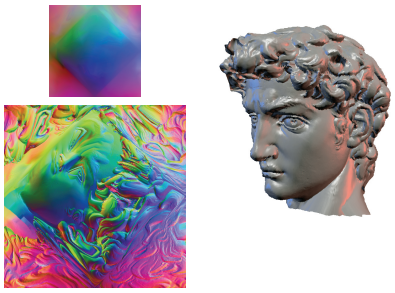


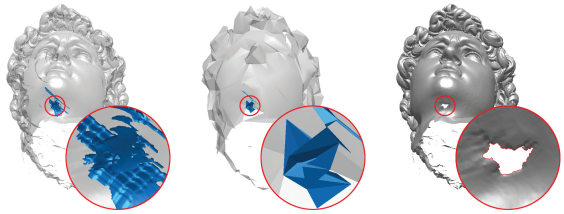
Figure 5: David head: geometry image (257x257), normal-map image (512x512), and normal-mapped reconstruction

7.4 Shape Compression

We use the progressive geometry compression method from [9] to convert our output semi-regular meshes into archives that can be used for storage and transmission. Fig. 7 and Table 1 present the compression results for several of our models. Since our method is a derivative of the MAPS approach, we expect to achieve similar compression performance to [9]. Therefore, we can proceed from the huge irregular input mesh to a small compressed archive storing an approximation of the same surface with low error.

8 Conclusion and Future Work

In this paper, we have presented the first out-of-core method for semi-regular remeshing of large polygonal models. It proceeds in a manner similar to the MAPS approach, first constructing a base domain by simplification, and then resampling the surface using the created parameterization.



(a) (b) (c)

Figure 6: Noise removal: The noisy region is shown in the original mesh on the left (a) and in the simplified mesh in the middle (b). After the region in the simplified mesh has been removed, it can also be removed from the original mesh as shown on the right (c).

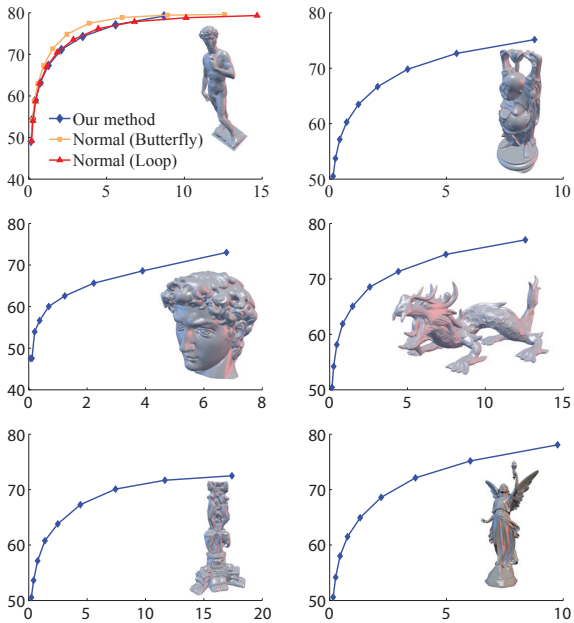


Figure 7: Rate-distortion curves: PSNR [9] is shown on the vertical axis vs. file size in 10K bytes on the horizontal axis. The curves at the top-left corner show our result compared to the Normal Mesh [5] results with Butterfly and Loop wavelet transforms.

Our method is implemented in a streaming framework and proceeds by synchronously streaming the input and the simplified version of the same surface.

Our remeshing algorithm works best for creating a mapping between a huge input mesh and its simplified version. Many processing operations on the simplified version of the mesh can then be "transported" by the stored map to the corresponding operations on the original surface. We illustrated our approach for map conversion and simple mesh editing for noise removal, in addition to semi-regular remeshing and shape compression.

Acknowledgments

This research was supported in part by the Korean Ministry of Information and Communication through the ITRC support program.

REFERENCES

[1] P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing. *ACM Trans. Graphics*, 21(3):347–354, 2002.

[2] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Trans. Visualization and Computer Graphics*, 9(4), 2003.

[3] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[4] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. *ACM Trans. Graphics*, 21(3):355–361, 2002.

[5] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder. Normal meshes. In *Proc. ACM SIGGRAPH 2000*, pages 95–102, 2000.

[6] J. Ho, K. C. Lee, and D. Kriegman. Compressing large polygonal models. In *Proc. IEEE Visualization 2001*, pages 357–362, 2001.

[7] M. Isenburg and P. Lindstrom. Streaming meshes. In *Proc. IEEE Visualization 2005*, pages 231–238, October 2005.

[8] A. Khodakovsky, N. Litke, and P. Schröder. Globally smooth parameterizations with low distortion. *ACM Trans. Graphics*, 22(3):350–357, 2003.

[9] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proc. ACM SIGGRAPH 2000*, pages 271–278, 2000.

[10] A. W. Lee, W. Sweldens, P. Schroder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *Proc. ACM SIGGRAPH 1998*, 1998.

[11] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proc. ACM SIGGRAPH 2000*, pages 259–262, 2000.

[12] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graphics*, 22(3):340–349, 2003.

[13] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Proc. Eurographics Symposium on Geometry Processing*, pages 17–28, June 2003.

[14] J. Wu and L. Kobbelt. A stream algorithm for the decimation of massive meshes. In *Proc. Graphics Interface 2003*, pages 185–192, June 2003.