Feature Sensitive Out-of-Core Chartification of Large Polygonal Meshes

Sungyul Choe, Minsu Ahn, and Seungyong Lee

Dept. of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
San 31, Hyoja-dong, Pohang, 790-784, Korea
{ggalssam, atom, leesy}@postech.ac.kr

Abstract. Mesh chartification is an important tool for processing meshes in various applications. In this paper, we present a novel feature sensitive mesh chartification technique that can handle huge meshes with limited main memory. Our technique adapts the mesh chartification approach using Lloyd-Max quantization to out-of-core processing. While the previous approach updates chartification globally at each iteration of Lloyd-Max quantization, we propose a local update algorithm where only a part of the chartification is processed at a time. By repeating the local updates, we can obtain a chartification of a huge mesh that cannot fit into the main memory. We verify the accuracy of the serialized local updates by comparing the results with the global update approach. We demonstrate that our technique can successfully process huge meshes for applications, such as mesh compression, shape approximation, and remeshing.

1 Introduction

Recently large polygonal meshes acquired by 3D scanning devices have become widely available. Processing these large meshes may be difficult or even impossible with existing mesh processing tools running on a fixed-size main memory. To handle large meshes, out-of-core algorithms have been introduced, where the whole mesh is not loaded into the main memory at the same time.

A simple but effective approach for out-of-core processing is to divide a huge mesh into several small pieces that can fit into the main memory. Then a mesh processing tool can be applied to each piece with a small memory footprint. Out-of-core algorithms based on mesh cutting or clustering have been proposed for mesh simplification [1, 2, 3] and mesh compression [4, 5].

Mesh partitioning for out-of-core processing can be obtained by dividing a large polygonal mesh into small pieces using the coordinate axes or voxel grids. However, in this case, the partitioning result does not reflect mesh features, which may degrade the performance of the processing. Hence, a feature-sensitive out-of-core mesh chartification technique will be a useful tool to improve the results of out-of-core processing.

Mesh chartification has been an active research area and used for numerous applications, such as texture atlas generation [6, 7], shape simplification [8], and shape decomposition [9]. Mesh chartification decomposes a mesh into charts,

H.-P. Seidel, T. Nishita, and Q. Peng (Eds.): CGI 2006, LNCS 4035, pp. 518–529, 2006.

where each chart consists of faces with similar properties. Excellent chartification techniques have been proposed, most of which try to generate flat and compact charts with features aligned at the chart boundaries. Unfortunately, these techniques assume the whole mesh can be accessed for processing at the same time and are not directly applicable to large meshes that cannot fit into the main memory.

In this paper, we propose a feature sensitive out-of-core chartification technique. Our technique adapts previous mesh chartification methods [7, 8] based on Lloyd-Max quantization to partition large meshes with limited main memory. The previous methods globally update chartification at each iteration of the Lloyd-Max quantization. In contrast, our technique locally updates the current chartification by considering only one chart and its neighborhood at a time. The chartification of the whole mesh is gradually updated by repeating the local update. We verify with experiments that the results of our technique are as good as those of the previous global update. As application examples, we show that the chartification results can be effectively used for mesh compression, shape approximation, and remeshing.

2 Related Work

2.1 Mesh Chartification

Mesh chartification techniques can be roughly classified into cluster merging and region growing algorithms. However, all the algorithms assume that the whole mesh resides in the main memory and cannot be used for out-of-core processing of large meshes.

A cluster merging algorithm consists of two steps: pair selection and merging [10, 11]. After evaluating the merging cost of each cluster pair, the algorithm selects and merges the pair having the minimum cost. After merging, the cost for the new cluster is updated and the process is repeated.

Lloyd-Max quantization, which is a well-known partitioning algorithm, has been widely used for data clustering and quantization [12, 13]. Sander et al. [7] introduced the Lloyd-Max quantization algorithm to mesh chartification, and Cohen-Steiner et al. [8] applied Lloyd-Max quantization for shape approximation.

2.2 Out-of-Core Algorithms

Several out-of-core techniques have been developed for simplification of huge meshes. Hoppe [1] segments a given mesh into charts and simplifies each chart independently while preserving the boundary edges. After independent simplification of charts, charts are merged and boundaries are simplified. Lindstrom [2] simplifies large meshes by vertex clustering. This algorithm can simplify large meshes with small overhead, but the connectivity and topology information are not utilized and removed. Isenburg et al. [3] proposed a local simplification technique. At each step, a small part of the mesh is partially loaded and simplified.

To compress large scanned meshes, Ho et al. [4] introduced an out-of-core compression technique. The technique partitions an input mesh into several exclusive charts and compresses charts independently with the Touma-Gotsman's algorithm [14]. The technique also maintains the gluing information to attach neighbor charts during decompression. Isenburg et al. [5] proposed an out-of-core data structure which is composed of small clusters. During compression, only necessary clusters are loaded into main memory and unnecessary clusters are released.

3 Out-of-Core Chartification

3.1 Overall Process

In this paper, we propose an out-of-core algorithm for feature sensitive chartification of large meshes. Our algorithm is based on Lloyd-Max quantization for meshes [7], which consists of two steps: region growing and seed recomputation. The region growing step creates a set of charts from given seeds. The seed recomputation step computes a new seed for each chart. The two steps are repeated in tandem until the terminal conditions are satisfied. However, for the region growing, we have to maintain the merging costs of all faces to neighbor regions. Thus, if an input mesh is too large to fit into the main memory, we cannot execute chartification with such a global approach. To enable chartification of large meshes, we propose a local update scheme for Lloyd-Max quantization. The basic idea is to keep a partial mesh in the main memory for processing which contains a subset of charts.

The overall process of the proposed out-of-core chartification algorithm is illustrated in Fig. 1. At the beginning, each chart is stored independently in a file. At the iterative optimization stage, we read and keep a few charts in the main memory, which are the selected chart and its neighborhood. We then update the charts in main memory by region growing similar to the original Lloyd-Max quantization. After updating the charts, we write and remove them from the main memory. By repeating this local update with varying selected charts, we can simulate the global update of chartification with limited main memory.

To bootstrap such repeating updates, we need an initial chartification. Previous methods [7, 8] randomly generate initial chartifications. In this paper, to

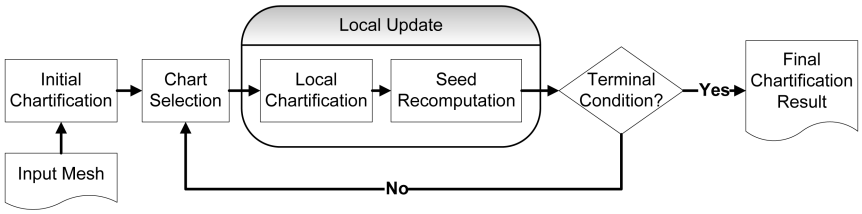


Fig. 1. Overall chartification process with local updates

achieve better results, we adopt the previous out-of-core chartification based on spatial subdivision [4, 5]. In addition, we execute post-processing to remove annuli charts along with charts that are too small or too big.

During the region growing step, a cost function is used to determine the conquered order of faces, and the shapes of charts are determined by the cost function. In this paper, we use cost functions proposed in [7] and [8]. To reflect mesh features in chartification, the cost functions incorporate the normal variations among faces. If we want other properties of charts, other cost functions can be used without changing the framework of the proposed algorithm.

4 Local Update Algorithm

4.1 Local Update

To update a chartification with Lloyd-Max quantization, the seeds are repositioned to the centroids of the current charts and the charts are recomputed with new seeds. In our out-of-core chartification method, the seed repositioning and chart recomputation are locally performed on a chart and its neighborhoods (see Fig. 2). With a local update, the boundary of the center chart is aligned with the features and the boundaries of neighbor charts are partially updated. In this paper, we call the previous algorithms [7, 8] the global update whereas our method is called the local update.

(a) before (b) after

Fig. 2. Local update of chartification: In (b), tiny black triangles are recomputed seeds

4.2 Update Ratio

In practice, for the global update scheme, a few iterations are sufficient for the convergence to optimal results [8]. Similarly, we use the number of iterations as the stopping condition for the local update scheme. However, it is difficult to count the number of updates in our framework. The boundaries of neighbor charts are partially updated in a local update and thus keeping track of update counts is not simple.

To resolve the problem, we define the *update ratio* of a chart. For a chart C, the update ratio is initially zero and increased by V_C with a local update, where

$$V_C = N_{pn}/N_n.$$

N_{pn} is the number of neighbor charts of C which has been involved in the local update. N_n is the total number of neighbor charts of C. For example, in Fig. 2, the update ratio of the center chart is increased by one and the update ratios of the neighbor charts are increased by the amounts less than one. If we want to obtain a result similar to that by n iterations of the global update, we repeat the local update until all charts have update ratios larger than n.

4.3 Chart Selection

In the iteration of local updates, the order of charts to be selected as the center chart influences the final result. A straightforward way is to select a chart whose update ratio is the minimum. However, such approach does not consider the effect of a local update on the neighbor charts and may cause over-updating of a neighbor chart if the chart already has a large update ratio. In this paper, we define the priority to determine the order of local updates as the average of update ratios of a chart and its neighbors. This priority considers the overall update ratio of a partial mesh that will be involved in a local update.

4.4 Local Update with the k-Ring Neighborhood

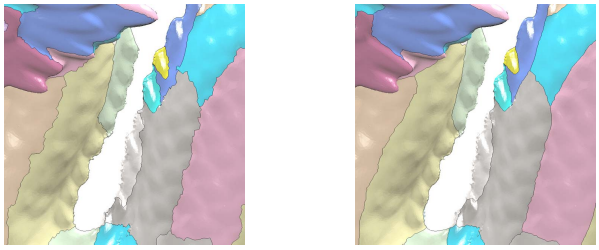
In Fig. 2, we only consider the center chart and its 1-ring charts for a local update. However, when the sizes of charts are relatively small, we can store a larger number of charts in the main memory. That is, we can process a center chart and its k-ring neighbor charts simultaneously. In this case, the priority to select the next chart to be processed can be computed as the average of the update ratios of the center and its k-ring neighbor charts, which constitute the region to be affected by a local update.

When we update a chart with its k-ring neighbor charts, the chart and its $(k-1)$-ring neighbor charts are fully updated and their update ratios are increased by one. That is, a larger number of charts are fully updated with this approach than with the local update of 1-ring neighborhood. Consequently, we can reduce the required number of iterations for local updates, which alleviates the overhead by file I/O and decreases the processing time.

4.5 Boundary Straightening

Some applications of mesh chartification need smooth and compact chart boundaries. To provide such results, we straighten chart boundaries after chartification. For each boundary shared by two adjacent charts, We find the shortest path from a corner vertex to the opposite one. To enhance the performance and maintain the shape features captured in the chartification process, a banded region is designated along the current boundary and the shortest path is calculated within the region. The banded region is defined by the k neighborhoods of the current boundary vertices.

Fig. 3 shows the effect of boundary straightening. Boundaries in Fig. 3(a) are created by chartification. Fig. 3(b) is the result of straightening, which shows smoothed boundaries.



(a) before (b) after

Fig. 3. Chart boundary straightening

4.6 Local Minimum

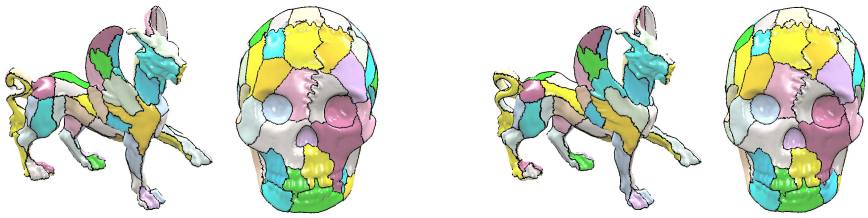
Usually chartification algorithms based on region growing suffer from local minima. During a chart update, a growing chart region cannot jump through high curvature features. Suppose several seeds reside in the same region which is surrounded by features in the chartification process. Then, the charts from the seeds cannot escape the region and we are left with unnecessary charts which could have merged into one chart. Similarly chartification cannot capture a feature if no seed is placed in a region surrounded by the feature. To avoid such local minima, Cohen-Steiner et al. [8] insert or delete charts incrementally and introduce region teleportation, which is similar to vector splitting for scattered data [15].

In this paper, we adapt the solutions in an out-of-core way. In the local update step, we simulate chart deletion by comparing the local chartification quality before and after deleting a chart. The quality of a chartification can be measured by the sum of face distances from the seeds, which will be discussed in Sec. 5.1. If the distance sum after chart deletion is less than before, we perform the incremental chart deletion. In that case, we also perform the incremental chart insertion, constituting a region teleportation. To insert a chart, we place a new seed on the face with the maximum region conquering cost.

5 Experimental Results and Applications

5.1 Chartification Comparison

Fig. 4 shows the comparison of the chartification results from global and local update schemes. For the global scheme, we globally update all charts five times from the initial chartification. For the local scheme, we apply the local update to all charts one by one and repeat this process five times. Note that in the local scheme, each chart is updated more than five times because a chart is changed when the chart itself or its neighbor is selected as the center chart of a local update. The cost function used for region growing is the one proposed in [7]. In Fig. 4, we can see that the chartification results of the local scheme are as good as the global scheme even though only local information is used for chart update.



(a) global update (b) local update

Fig. 4. Comparison of global and local update schemes

Table 1. Comparison of chartification quality: The quality was measured by the average of the face cost sums of charts

# iterations	global update scheme					local update scheme				
	1	2	3	4	5	1	2	3	4	5
feline	8.09	5.90	5.47	5.35	5.39	5.48	5.47	5.46	5.32	5.27
skull	73.34	63.91	61.24	59.39	59.06	58.18	57.49	57.35	57.28	57.34

Table 2. Statistics of local updates with different k-ring neighborhoods

model	# faces	# charts	k-ring	# local updates	max # vertices in memory	total # loaded faces	processing time
dragon	800,000	300	1-ring	524	44,508	9,058,446	11m 36s
			2-ring	151	139,234	7,262,000	9m 54s
happy Buddha	1,082,760	600	1-ring	976	17,324	11,534,091	14m 3s
			2-ring	277	42,834	9,387,228	12m 27s
xyzrgb dragon	7,218,906	600	1-ring	949	109,125	76,532,952	2h 12m 13s
			2-ring	273	201,488	60,236,558	2h 2m 31s

Table 1 numerically compares the chartification quality between the global and local update schemes. The chartification quality is measured by the average of the face cost sums of charts. Once the chartification has been finished, each face has the cost (distance) from the seed of the chart it belongs to. The sum of the face costs of all charts is minimized when Lloyd-Max quantization is converged [16]. Hence, we can consider the average of the face cost sums of charts as the measure of the convergence for the iterative chart updates. Table 1 shows that the local updates result in smaller values of the measure, which is natural when we recall that a chart is updated more often with local updates than with global updates for the same number of iterations.

In Table 2, we also compare the chartification results of the 1-ring and 2-ring update strategies. In the experiment, the local update process was continued until the update ratios of all charts were larger than five. Table 2 shows that the 2-ring update strategy utilizes the main memory more efficiently and as a result, the file I/O overhead and processing time are decreased. The computation time was measured on a Windows PC with a Pentium D 3GHz CPU and 1GB memory.

5.2 Mesh Compression

Our out-of-core chartification technique can be used for effectively compressing large polygonal meshes. Choe et al. [17] proposed a framework for random accessible mesh compression, where a necessary mesh part can be decompressed without decoding other parts. By combining our out-of-core chartification technique with the framework, we can apply random accessible compression to large meshes that cannot fit into main memory.

To achieve a good compression ratio in the random accessible mesh compression framework, two properties of charts are important: planarity and compactness. Planar charts enable effective prediction in geometry encoding and shorter chart boundaries help achieve a better compression ratio. Hence, for the chartification for random accessible mesh compression, we use the cost function proposed in [7]. In Fig. 5, the first column shows the chartification results for the compression framework. In the experiments, local updates were performed until the update ratios of all charts were larger than five.

Table 3. Statistics of compression examples: The compression ratios were obtained with the random accessible mesh compression framework [17]

model	# vertices	# charts	compression ratio (bit/v)					
			with spatial subdivision			with our approach		
			connect.	geometry	total	connect.	geometry	total
dragon	400,000	271	2.04	16.71	18.75	1.99	16.58	18.57
happy Buddha	541,366	581	2.57	20.99	23.56	2.55	20.73	23.28
xyzrgb dragon	2,933,046	578	0.99	5.86	6.85	0.93	5.71	6.64
lucy	14,027,868	1,184	2.07	16.06	18.13	2.05	15.67	17.72

Table 4. Timing data of mesh compression examples: The chartification was obtained with the cost function proposed in [7]

model	# vertices	# charts	initial chartification	iterative local update	compression
dragon	400,000	271	5m 9s	10m 43s	1m 20s
happy Buddha	541,366	581	6m 50s	13m 12s	1m 40s
xyz rgb dragon	2,933,046	578	1h 55m	2h 30m	7m 45s
lucy	14,027,868	1,184	3h 15m	8h 55m	32m 30s

The compression results for several models are summarized in Table 3. To show the effect of chartification on the compression framework, we compare the compression results from the chartification obtained by spatial subdivision and the chartification obtained by our technique. We can see that the compression ratio is improved for every example because our chartification is better in terms of planarity and compactness of charts. Table 4 shows the timing data for mesh chartification and compression, which were measured on a Windows PC with a Pentium D 3GHz CPU and 1GB memory.

Fig. 5. Chartification and shape approximation results: The first column shows the chartification results for compression, the second column shows the chartification results for shape approximation, and the third column shows approximation meshes

5.3 Shape Approximation

Cohen-Steiner et al. [8] proposed an excellent shape approximation technique which obtains an approximation mesh from the original mesh using mesh

Table 5. Timing data of shape approximation examples: The chartification was obtained with the cost function proposed in [8]

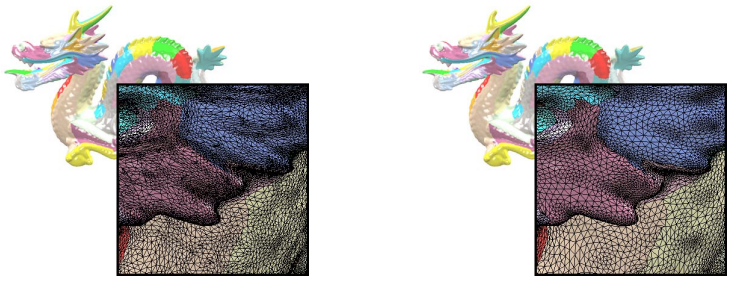
model	# vertices	# charts	initial chartification	iterative local update	approximation mesh generation
dragon	400,000	789	5m 9s	10m 50s	2m 5s
happy Buddha	541,366	1,472	6m 50s	15m 30s	2m 54s
xyz rgb dragon	2,933,046	709	1h 55 m	3h 35 m	16m 31s
lucy	14,027,868	1,615	3h 15 m	31h 30 m	1h 56m

chartification. In this technique, the vertices, edges, and faces of an approximation mesh correspond to the corner vertices, chart boundaries, and chart interiors, respectively. Our out-of-core chartification method can be used to extend the shape approximation technique to process huge meshes. The resulting approximation mesh will nicely reflect the features of the original mesh. Note that the previous out-of-core mesh partitioning based on spatial subdivision [4] is not proper for this purpose because it does not generate feature-sensitive chartification.

In [8], to approximate a chart with a plane, the normals of faces in a chart should be similar to each other. Hence, the cost function used for region growing in [8] measures the variation of a face normal from the representative normal of a chart. In Fig. 5, the second column shows the chartification results with this cost function and the third column shows the shape approximation results. Table 5 shows the timing data of the shape approximation examples. Again, in the experiments, the local update process was continued until the update ratios of all charts were larger than five.

5.4 Remeshing

Our feature sensitive out-of-core chartification technique can be used to preform effective remeshing of very large meshes with limited memory. Once chartification has been obtained for a given mesh, we keep the chart boundaries and just modify the sampling and connectivity of inner vertices of each chart



(a) before (b) after

Fig. 6. Remeshing example

during the remeshing process. With this approach, we can expect to obtain a remeshing result that nicely preserves the features of the original mesh because the chart boundaries are aligned with high-curvature features. For remeshing of chart interiors, we adopt the explicit surface remeshing technique [18], which can separately remesh chart interiors while keeping the chart boundaries.

Fig. 6 shows an example. In Fig. 6(b), the mesh generated by the remeshing has a simple and regular structure. Although we have reduced the size of the given mesh by a half, the new mesh still contains the original shape features. In addition, no artifacts are visible around the boundaries between charts even though each chart has been remeshed independently.

6 Conclusion and Future Work

In this paper, we introduced a feature sensitive out-of-core chartification technique for large polygonal meshes. To process huge meshes, our out-of-core algorithm keeps only a partial mesh in the main memory at a time and locally updates chartification. To verify the validity of our approach, we showed that the results of our local update scheme are as good as the previous global update scheme.

In the current implementation, after each local update, updated charts are written to the hard disk and released from the main memory although some of them can be used for successive chart updates. If we maintain a cache of charts in the main memory, we will be able to reduce the overhead by unnecessary file I/O. The design of a cache structure for the charts which improves the performance of out-of-core chartification is an interesting future work.

Acknowledgements

The authors would like to thank Junho Kim for helpful discussion. The dragon, happy Buddha, xyz rgb dragon, and lucy models are courtesy of Stanford Graphics Lab. The skull model is courtesy of Headus and Phil Dench. The dinosaur and feline models are courtesy of Cyberware and Multi-Res Modeling Group at Caltech, respectively. This research was supported in part by the BK21 program, the ITRC support program, and KOSEF (F01-2005-000-10377-0).

References

1. Hoppe, H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proc. IEEE Visualization 1998. (1998) 35–42
2. Lindstrom, P.: Out-of-core simplification of large polygonal models. In Proc. ACM SIGGRAPH 2000 (2000) 259–262
3. Isenburg, M., Lindstrom, P., Gumhold, S., Snoeyink, J.: Large mesh simplification using processing sequences. In: Proc. IEEE Visualization 2003. (2003) 465–472
4. Ho, J., Lee, K.C., Kriegman, D.: Compressing large polygonal models. In: Proc. IEEE Visualization 2001. (2001) 357–362

5. Isenburg, M., Gumhold, S.: Out-of-core compression for gigantic polygon meshes. ACM Transaction on Graphics **22**(3) (2003) 935–942
6. Lévy, B., Petitjean, S., Ray, N., Maillot, J.: Least squares conformal maps for automatic texture atlas generation. ACM Transaction on Graphics **21**(3) (2002) 362–371
7. Sander, P.V., Wood, Z.J., Gortler, S.J., Snyder, J., Hoppe, H.: Multi-chart geometry images. In: Proc. Eurographics Symposium on Geometry Processing 2003. (2003) 146–155
8. D. Cohen-Steiner, P. Alliez, M.D.: Variational shape approximation. ACM Transaction on Graphics **23**(3) (2004) 905–914
9. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Transaction on Graphics **22**(3) (2003) 954–961
10. Sander, P.V., Snyder, J., Gortler, S.J., Hoppe, H.: Texture mapping progressive meshes. In Proc. ACM SIGGRAPH 2001 (2001) 409–416
11. Garland, M., Willmott, A., Heckbert, P.S.: Hierarchical face clustering on polygonal surfaces. In: Proc. 2001 ACM Symposium on Interactive 3D Graphics. (2001) 49–58
12. Lloyd, S.: Least square quantization in PCM. IEEE Transaction on Information Theory **28** (1982) 129–137
13. Max, J.: Quantizing for minimum distortion. IEEE Transaction on Information Theory **6** (1960) 7–12
14. Touma, C., Gotsman, C.: Triangle mesh compression. In: Proc. Graphics Interface 1998. (1998) 26–34
15. A.Gersho, R.M.Gray: Vector Quantization and Signal Compression. Kluwer Academic Publishers (1991)
16. Surazhsky, V., Alliez, P., Gotsman, C.: Isotropic remeshing of surfaces: A local parametrization approach. In: Proc. 12th International Meshing Roundtable. (2003)
17. Choe, S., Kim, J., Lee, H., Lee, S., Seidel, H.P.: Mesh compression with random accessibility. The 5th Korea-Israel Bi-National Conference on Geometric Modeling and Computer Graphics 2004 (2004) 81–86
18. Surazhsky, V., Gotsman, C.: Explicit surface remeshing. In: Proc. Eurographics Symposium on Geometry Processing 2003. (2003) 20–30