A Stream Algorithm for the Decimation of Massive Meshes

Jianhua Wu Leif Kobbelt

Computer Graphics Group, RWTH–Aachen, Germany

Abstract

We present an out-of-core mesh decimation algorithm that is able to handle input and output meshes of arbitrary size. The algorithm reads the input from a data stream in a single pass and writes the output to another stream while using only a fixed-sized in-core buffer. By applying randomized multiple choice optimization, we are able to use incremental mesh decimation based on edge collapses and the quadric error metric. The quality of our results is comparable to state-of-the-art high-quality mesh decimation schemes (which are slower than our algorithm) and the decimation performance matches the performance of the most efficient out-of-core techniques (which generate meshes of inferior quality).

Key words: mesh decimation, massive data, stream processing, multiple-choice optimization

1 Introduction

Due to the advances in 3D scanning technology and numerical simulation, the complexity of geometric models is increasing much faster than the graphics and computing performance of current PC hardware. This is why mesh decimation has been an active research area over the past ten years.

While the first generation of mesh decimation algorithms are based on the assumption that the complete geometry data fits into a global in-core data structure, the second generation algorithms relax this restriction by reducing the size of the active working set and storing the major part of the data externally on disk. For such algorithms, the actual in-core memory requirements are independent from the size of the input or output mesh but the data in external memory usually has to be accessed several times which slows down the process significantly.

In this paper we present a *stream* algorithm for mesh decimation which does not require to permanently store the data at all (not even on disk). The concept of stream processing is that input data is taken in sequentially without backtracking. The output data is written sequentially as well and no feed-back to the input stream is possible. The amount of in-core memory that is allocated by a stream algorithm does not depend on the amount of data to be processed. Since stream algorithms process the data in one single pass, we can implement a software system where the geometry generating pre-process feeds its output directly into the decimation post-process without storing it to disk.

When decimating a stream of geometry data, the mesh is logically split into three parts: the unread postfix of the input stream representing the unprocessed part of the input mesh, the in-core portion of the mesh that is currently processed by the decimation algorithm, and the written prefix of the output stream representing the part of the output mesh that has already been decimated to the target resolution (cf. Fig. 1).

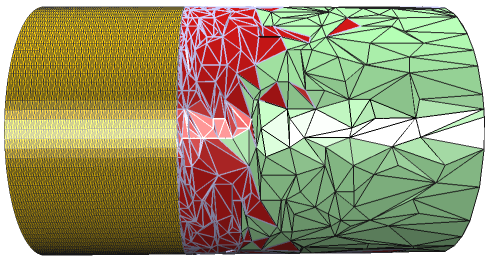

Figure 1: This snapshot of the stream decimation to a cylinder mesh shows the yet unprocessed part of the input data (left), the current in-core portion (middle) and the already decimated output (right). The data in the original file happened to be pre-sorted from right to left.

Since the stream algorithm uses an in-core buffer of limited size, we have to assume that the geometry stream is *approximately* pre-sorted (e.g. by one coordinate). This is a very natural assumption since most massive mesh models are generated by some marching cubes type algorithm which builds up the geometry layer by layer [16, 5]. Obviously the pre-sorting does not have to be perfect. We only have to guarantee that the portion of the input stream that lies between the actual occurrence of a particular triangle and its position in a perfectly sorted stream, fits into the in-core buffer of the stream algorithm. In the rare cases where this mild sorting requirement is not satisfied, we have to apply an out-of-core pre-sorting step like in [15].

Besides the independence from the sizes of the input and output meshes respectively, our stream algorithm

has several important additional features: Since the decimation technique itself is based on incremental edge-collapsing with quadric error metrics (QEM) [8], we obtain a mesh quality which is indistinguishable from state-of-the-art in-core decimation techniques. On the other hand, we obtain decimation rates of $30K$ to $40K$ triangles/sec on commodity PC hardware including I/O times which is competitive to the fastest out-of-core algorithms with similar properties (which, however, produce meshes of far inferior quality).

2 Related work

In general, mesh decimation is a very complex optimization problem. Given some input mesh $\mathcal{M} = [\{p_i\}, \{T_j\}]$, the task is to find another mesh $\mathcal{M}' = [\{p'_i\}, \{T'_j\}]$ which has a prescribed number of triangles and minimizes the approximation error $\|\mathcal{M} - \mathcal{M}'\|$ (cf. [7, 10]). For most applications, the computation of the exact global optimum is far too complex [1]. Hence, one usually tries to find solutions $\mathcal{M}'$ with *approximate optimality* where the computation costs can be traded for geometric suboptimality. Over the last years the *greedy optimization* paradigm has established the de facto standard for mesh decimation algorithms. In the greedy approach, the decimation is performed by a sequence of atomic decimation steps which typically remove a single vertex from the mesh. The greedy paradigm then states that in each step the best decision is made without any look-ahead or backtracking. In order to efficiently identify the best choice in each step, all candidates have to be organized in a global priority queue [12, 8].

Recently, another approximate optimization technique has been applied to the mesh decimation problem [22]. It has been shown that *randomized multiple choice optimization* [17] produces results that have almost the same quality as the results of greedy optimization but with significantly reduced computation costs. Since no global data structure such as a priority queue needs to be maintained, the algorithmic structure of multiple choice decimation is extremely simple. In each step, a random set of candidates is picked and the best among these candidates is chosen. Usually a small number of 10 to 15 random candidates is sufficient to produce meshes which are indistinguishable from the meshes produced by greedy algorithms.

To address the problem of ever increasing input model sizes, out-of-core decimation techniques have been introduced. These techniques are designed to perform the decimation while reading the data such that only the output model has to be stored. In [6] this is achieved by pre-sorting all edges according to their length and using this ordering for the decimation sequence. A more efficient algorithm appeared in [14] where a vertex clustering technique [19] is applied. Incoming geometry data is accumulated on the fly in a voxel grid which guarantees complete independence from the input mesh complexity. Since the (random access) voxel grid has to be stored in-core, the memory requirements of this algorithm are on the order of the output mesh complexity.

To enable application scenarios where neither the input mesh nor the output mesh of a decimation algorithm fit into main memory, the out-of-core vertex clustering has been extended in [15] such that no internal data structure is required anymore. Instead, the decimation is performed in several passes. Although the algorithm requires several out-of-core sorting steps it still achieves a high decimation performance of some 30K triangles/sec on a commodity PC.

The major drawback of vertex clustering techniques for mesh decimation is the relatively poor geometric and topologic quality of the resulting meshes. The output is usually no longer manifold and the vertex density does not adapt to the local curvature. A non-uniform clustering technique has been proposed in [20] but the required space partition data structure requires memory space proportional to the output mesh complexity. The RSIMP approach [2] in its out-of-core implementation [21] is another adaptive vertex clustering technique which can be tuned to exploit caching effects by sorting the data according to the corresponding voxel-bins. However, it still loses performance compared to [15]. The multi-phase algorithm in [9] can produce high quality meshes, since after an initial uniform clustering phase, the second phase applies a standard greedy decimation procedure [8]. However, again, the maximum size of the output mesh is limited by the memory resources.

Another way to decimate massive meshes is to split the data into smaller blocks and then stitch the decimated pieces together. This approach can easily be combined with incremental edge collapsing [8] and thus usually leads to meshes with far superior quality compared to vertex clustering. However, the splitting and stitching can be computationally expensive and special care has to be taken at the seams between the blocks to avoid mesh artifacts.

In [11] this approach is applied to terrain models and in [18, 4] to arbitrary meshes. In both cases the advantage of improved mesh quality is compromised by a significantly reduced decimation rate. In [4] a rate of only 6K-10K triangles/sec is achieved for large datasets on a PC comparable to the one used in [15]. This rate is even halved if the spatial splitting pre-process is counted in as well.

In this paper we present a new out-of-core decimation algorithm that overcomes many drawbacks of ear-

lier techniques. First, the decimation is based on edge collapsing and quadric error metrics which guarantees a high quality of the output meshes. However in contrast to [4] our scheme uses the multiple choice optimization strategy and no spatial splitting of the input mesh. On the other hand, our algorithm is as fast as the out-of-core vertex clustering technique in [15]. This is achieved because our stream algorithm processes the data in one single pass. The memory requirements of our algorithm are independent from the size of the input and the output.

3 Ideal stream algorithm for decimation

As we already defined in the introduction: a stream algorithm has to transform a sequential input stream of arbitrary size into an output stream of also arbitrary size by using an in-core buffer of only fixed size. In our mesh decimation setup, these streams consist of geometry data in STL format. In this format (dubbed "triangle soup" in [14]) every triangle is given by its three vertices with three coordinates each. Although this format redundantly stores every mesh vertex several times (six times in average) it is still preferred in many applications such as rapid prototyping since no global indexing is required.

Let N_{max} be the maximum number of triangles that fit into the in-core triangle buffer. The stream algorithm is performing three different operations that affect the filling level $N_{current} \leq N_{max}$ of that buffer.

- **READ**(k) takes the next k triangles from the input stream and inserts them into the current in-core portion (which maintains both geometry and topology) of the mesh. $N_{current} \leftarrow N_{current} + k$
- **DECIMATE**(k) performs k edge collapse operation on the in-core portion of the mesh according to the multiple choice optimization strategy. Each edge collapse removes two triangles from the mesh. $N_{current} \leftarrow N_{current} - 2k$
- **WRITE**(k) removes k triangles from the in-core portion and writes them into the output stream. $N_{current} \leftarrow N_{current} - k$

While the algorithm is running, these three operations are applied in arbitrary order. The only hard restriction is that the buffer must not overflow which implies that if $N_{current} = N_{max}$, we have to apply *DECIMATE* or *WRITE* before reading the next input triangle. Some additional weak restrictions are that the filling level should be kept as high as possible to provide a sufficiently large set of candidates for the multiple choice optimization and that the number of *DECIMATE* and *WRITE* operations should be balanced such that the output mesh has the prescribed target resolution.

Let p be the percentage to which the input mesh should be decimated and x be the unknown number of input triangles. Obviously, the decimation algorithm has to perform $p\,x$ *WRITE* operations and hence $(1-p)\,x/2$ *DECIMATE* operations to process the complete input data. Even though the stream algorithm does not know the number of input triangles, we can still derive that the *ratio* between *DECIMATE* and *WRITE* operations is

$$\frac{DECIMATE}{WRITE} = \frac{1-p}{2\,p}$$

which implies that the ideal randomized multiple choice algorithm is given by the following pseudo-code:

READ (N_{max})

while *input not empty*
  random choice with probabilities $1-p$: $2\,p$
    A: DECIMATE (1) *& READ* (2)
    B: WRITE (1) *& READ* (1)

while *buffer not empty*
  random choice with probabilities $1-p$: $2\,p$
    A: DECIMATE (1)
    B: WRITE (1)

This algorithm consists of three stages: the initial filling of the buffer (line 1), the actual stream processing (lines 2 to 5), and the concluding clearing of the buffer (lines 6 to 9).

4 Real stream algorithm for decimation

For a real implementation of this ideal algorithm, we have to make a few modifications in order to make the algorithm run effectively. In the introduction we explained that while a stream algorithm is running, the data is split into three disjoint parts ($\mathcal{A}$: postfix of the input stream, $\mathcal{B}$: in-core portion, and $\mathcal{C}$: prefix of the output stream). In the case of mesh decimation these three parts are submeshes of the original data at different resolution levels (cf. Fig.1).

In order to maintain the global mesh consistency, the two boundary polygons P_{ab} and P_{bc} which represent the interfaces between the parts $\mathcal{A}$/$\mathcal{B}$ and $\mathcal{B}$/$\mathcal{C}$ respectively, must not be modified by the *DECIMATE* operations. For P_{ab} this is necessary since no vertex should be removed before all its neighbors are read from the input stream. For P_{bc} this restriction guarantees that the output mesh has a consistent triangle neighborhood relation, i.e. the neighborhood of a triangle is not further decimated after it has been written to the output stream (cf. Fig. 2). In our implementation the operators *DECIMATE* and *WRITE* take care of this consistency checking (cf. Section 4.3

and 4.4). Notice that, depending on the geometric shape of the input mesh, the boundary polygons P_{ab} and P_{bc} usually consist of many connected components.

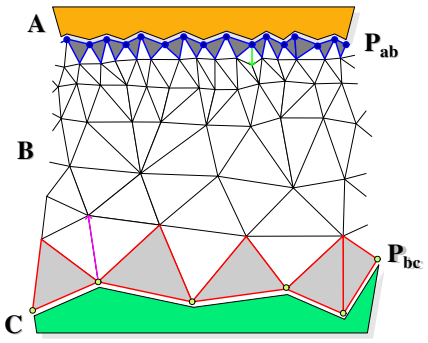

Figure 2: During stream processing the mesh data is split into three parts. Part $\mathcal{A}$ (top) is the unread part of the input stream and part $\mathcal{C}$ (bottom) has already been written to the output stream. Part $\mathcal{B}$ (middle) is currently in-core. The two boundary polygons P_{ab} and P_{bc} must not be modified. This restriction disqualifies the shaded in-core mesh parts from the set of candidates for the decimation.

Another issue is that with a pure random decision whether a *DECIMATE* or a *WRITE* operation is performed next, it can happen that a very small (high resolution) triangle is accidentally written to the output stream. This typically leads to strong variations in the vertex densities of the output mesh. In a real implementation we therefore have to make sure that the *WRITE* operation is only executed when a sufficient number of coarse triangles is present in the in-core buffer, i.e. when a sufficient number of decimation steps has been performed before.

Finally in a real operating system the file and stream access is usually much faster if we read and write the data in larger blocks. Hence we do not want to apply the *READ* and *WRITE* operation one by one but rather in larger chunks.

4.1 Overall procedure

The major difference between the ideal and the real stream decimation algorithm is that the decision whether to *DECIMATE* or to *WRITE* is no longer random. Only the decision *which* candidate to *DECIMATE* or to *WRITE* still remains a random process. The blockwise *READ* and *WRITE* is achieved by introducing a "hysteresis" which means that we are not reading any data before the filling level $N_{current}$ is down to 50%. However, whenever we actually perform a *READ* operation we read in the maximum number of triangles $N_{max} - N_{current}$ to raise the filling level to 100%.

In order to rate the *resolution level* R of the current in-core portion of the mesh, we count the total number of triangles N_{in} that have been read from the input stream until now, the number $N_{current}$ of triangles currently in the triangle buffer, and the number N_{out} of triangles that have been written to the output stream until now. Since the target resolution is given by the percentage p, we know that each triangle in the output stream statistically (not geometrically) represents $1/p$ triangles from the input stream. Consequently, the $N_{current}$ triangles in the buffer represent $N_{in} - N_{out}/p$ many input triangles and hence we find the current in-core resolution level to be

$$R = \frac{N_{current}}{N_{in} - N_{out}/p}.$$

The stream algorithm has to make sure that *WRITE* operations are only executed when the resolution level of the in-core portion equals p. Notice that the above resolution estimate is based on a statistical argument. Since the resolution level of the in-core triangles usually varies between the two boundaries P_{ab} and P_{bc}, we cannot make any statement about individual triangles. Nevertheless, since our candidate selection is based on a random (multiple) choice, we have to rely on the *expected* (average) resolution level.

In the initialization phase of the algorithm, we have to fill the triangle buffer and then apply *DECIMATE* operations. Before the first triangle is written to the output stream, the boundary polygon P_{bc} is empty but as long as only a small portion of the mesh has been read, the boundary polygon P_{ab} covers a relatively large part of the in-core mesh. Hence we have to be careful about the right filling level to avoid extreme over-decimation of the non-boundary parts of the in-core mesh. To achieve this, we use a special initialization loop that alternates reading and decimation until the in-core resolution level R has reached the target value p.

set n such that $\frac{1}{n} \geq p > \frac{1}{n+1}$
READ $(N_{max}/2)$
repeat $(n-1)$ *times*
    READ $(N_{max}/2)$
    DECIMATE $(N_{max}/4)$

At the end of the loop we invariantly have a filling level of $N_{current} = N_{max}/2$. After the loop has been repeated $(n-1)$ times we have read $\frac{n}{2} N_{max}$ triangles in total and no triangle yet written to the output stream. Hence the current in-core resolution level is $R = \frac{1}{n} \approx p$.

After the initialization, the main loop is processing the stream according to the following pseudo-code:

while *input not empty*
  READ $(N_{max}/2)$
  DECIMATE $((1-p)\,N_{max}/4)$
  WRITE $(\frac{p}{2}\,N_{max})$

Again, at the end of the loop we have $N_{current} = N_{max}/2$ and the in-core resolution level is always kept close to $R \approx p$. Notice that in line 3 of the above pseudocode, we are performing $(1-p)\,N_{max}/4$ edge collapses to remove $(1-p)\,N_{max}/2$ triangles from the mesh such that the *READ* and *DECIMATE* operations together increase the filling level by $\frac{p}{2}\,N_{max}$ which is exactly the number of triangles that are then written to the output stream in line 4.

After all triangles have been read from the input stream, we have to clean the in-core buffer. If the in-core resolution level would be exactly $R = \frac{1}{n} = p$ after the main loop, we could simply write out all remaining triangles. However, in order to account for the slight inaccuracy $\frac{1}{n} - p = \varepsilon \geq 0$ that has been made in the initialization loop, we have to decimate a few more triangles to get the exact number of output triangles $N_{out} = p\,N_{in}$.

DECIMATE $((1-n\,p)\,N_{max}/4)$
WRITE $(n\,p\,N_{max}/2)$

The number of $n\,p\,N_{max}/2$ output triangles corresponds to the number of $n\,N_{max}/2$ input triangles that have been read in the initialization loop.

Notice that if the input mesh is not large enough or if the decimation percentage p is too small then the complete output mesh might fit into the in-core buffer. In this case the stream decimation reaches the end of the input stream while still in the initialization loop. Hence, the main loop is skipped and the buffer contents is written to the output stream. Also, the length of the input stream is most likely not an exact multiple of $N_{max}/2$ such that the stopping criteria for the initialization and the main loop have to check the actual number of read triangles.

4.2 Reading input triangles

The *READ*(k) operation reads a sequence of k triangles from the input stream and inserts them into a half-edge mesh data structure [3] representing the in-core part of the mesh. The computationally most expensive part of this procedure is to find the edge(s) in the boundary polygon P_{ab} where this new triangle has to be connected. If such an edge does not exist then the new triangle is inserted as an isolated triangle with all its three edges forming a new connected component of P_{ab}. Otherwise the new triangle is connected to its neighbors and the boundary P_{ab} is updated. To speed up the neighbor search, the algorithm maintains a lookup table for all mesh vertices currently belonging to P_{ab}. The error quadrics associated with each vertex of the new triangle are initialized or updated by adding the squared plane equation.

4.3 Multiple choice decimation

As explained in Section 2, the concept of multiple choice optimization consists of sampling a random set of candidates in every step and choosing the best among them. No global sorting of candidates is needed. In the case of mesh decimation the overall set of candidates is the set of half-edges in the mesh. The quality of a candidate collapse operation is rated by using the quadric error metric introduced in [8]. Hence, in every decimation step we test for a small number (usually 5 to 15) of half-edges the potential quadric error and then execute that collapse with the *smallest* error. If Q_p and Q_q are the error quadrics of two vertices $\mathbf{p}$ and $\mathbf{q}$ connected by a half-edge pointing from $\mathbf{p}$ to $\mathbf{q}$ then after the collapse only the vertex $\mathbf{q}$ survives and its associated error quadric is set to $Q_p + Q_q$.

As mentioned earlier, the decimation procedure must not modify the two boundary polygons P_{ab} and P_{bc}. Hence all half-edges emanating from a boundary vertex have to be excluded from the candidate set. Also the half-edges pointing to vertices on the polygon P_{ab} cannot be collapsed because the error quadric of its boundary end-vertex is not completely known yet. Half-edges pointing to boundary vertices on P_{bc} however, can be collapsed because their error quadrics are known and they do not modify P_{bc}.

4.4 Writing output triangles

The selection of the triangles to be written to the output stream is also implemented as a randomized multiple choice selection. For a random set of candidate triangles we evaluate their error quadrics and then choose the one with the *largest* error. In analogy to the last section we define the error quadric of a candidate triangle $T = [\mathbf{p}, \mathbf{q}, \mathbf{r}]$ by simply adding the three quadrics Q_p, Q_q, and Q_r associated with its three vertices. The actual quadric error value is then obtained by evaluating this quadric at the center of the triangle.

In principle, all triangles not touching the boundary polygon P_{ab} could be candidates for output. However, in practice it turns out that with this large set of candidates, the randomized multiple choice selection will generate many small holes scattered all over the in-core mesh. As a consequence the components of the boundary P_{bc} will cover large portions of the mesh and thus reduce the candidate set for decimation significantly (cf. Fig. 3). We therefore restrict the candidate set to those triangles adjacent to P_{bc} (and not adjacent to P_{ab}) since this guarantees that the number of connected components of P_{bc} does not

increase. Only in exceptional cases where no suitable triangle adjacent to P_{bc} can be found, we allow all interior triangles to be candidates for output.

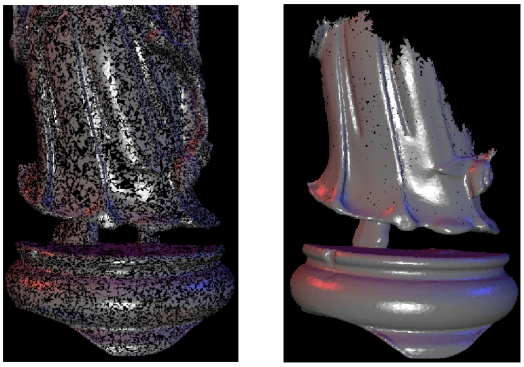

Figure 3: Multiple choice selection with candidates taken randomly from the whole in-core mesh (left) or just from the triangles adjacent to P_{bc} (right).

When we select a triangle and write it to the output stream, we change the positions of its three vertices to the optimal position indicated by the associated error quadric (minimum of the quadric). This improves the feature alignment of the vertices in the output mesh. Notice that by this final re-positioning we obtain precisely the output quality of those decimation schemes that apply full edge collapses in every step (while we are only using half edge collapses). The reason for this is that the intermediate vertex positions computed for the full edge collapses have no influence on the error quadric accumulation.

Since we have to preserve the mesh consistency along the boundary polygon P_{bc}, we can re-position each output vertex only once and then freeze its position for the neighboring triangles.

5 Results

To make our results comparable to others we use a setup for our experiments in which the stream algorithm reads its input data from a file and writes it back to another file by using the UNIX pipe mechanism. All CPU-timings we give include these I/O times. Notice that in an integrated software system, the decimation could directly process the output of the mesh generation process and hence would run much faster since no disk access would be necessary at all.

We use a PC with 866 MHz P3 processor since a similar type of computer has been used by OOCSx [15] and OEMM [4] as well. For the timings in Table 1 we set the in-core buffer capacity to $400K$ triangles for the Buddha and David models ($1200K$ triangles for the St. Matthew model) and the multiple choice decisions are always based on 8 random candidates. The resulting in-core memory consumption of the corresponding UNIX-process is $100MB$ ($315MB$ for St. Matthew), which includes memory space for a half-edge based mesh representation of the in-core portion and various hash tables to accelerate random access and searching. This memory consumption is also comparable to OEMM.

models	T_{input}	T_{output}	p (%)	time (hh:mm:ss)	rate (t/sec)
Buddha	1,087,470	21,748	2	0:32	33.3K
		217,038	20	0:29	30.0K
David head	4,000,885	80,016	2	2:04	31.7K
		409,048	10	2:09	27.8K
David 2mm	8,254,150	82,541	1	4:19	31.6K
		829,048	10	4:30	27.5K
David 1mm	56,230,343	562,219	1	32:40	28.4K
		2,815,413	5	33:21	26.7K
St. Matthew	372,767,445	559,087	0.15	3:54:06	26.5K
		1,863,837	0.5	4:01:27	25.6K

Table 1: Run-time performance of the stream algorithm including I/O times on a PC with 866 MHz P3 processor.

The observed decimation rate is about $28K$ triangles/sec which matches the *average* rates obtained by OOCSx on PCs for massive datasets. The rate is slightly sensitive to the input model size since a larger boundary polygon P_{ab} increases the searching time when inserting new triangles into the in-core mesh. However, the sensitivity to the output model size is not as strong as it is for OOCSx where large intermediate file sizes slow down the algorithm significantly. Compared to OEMM, our timings are about a factor 3 to 5 times faster and this factor even increases if we count the OEMM construction as part of the decimation.

On the 866 MHz PC the time for I/O access is responsible for approximately 50% of the total running time. If we run the same algorithm on a faster PC with 2.8 GHz P4 processor, the average decimation rates go up to $40K$ – $59K$ triangles/sec in the above experiments.

As we show in Fig. 4, 5 and 6, the quality of our output meshes is the same as obtained by OEMM and superior to the quality obtained by OOCSx with respects to visual appearances, error measurements and topology consistencies. Fig. 4 also reveals that our out-of-core stream decimation can generate comparable results with in-core QEM [8] simplification. To produce these figures we used the example data provided by P. Cignoni for the OEMM and QEM results and re-implemented OOCS [14] which produces the same results as OOCSx.

6 Conclusion and limitations

In this paper we presented a new out-of-core mesh decimation algorithm that works on arbitrarily large streams of triangle data. It combines high quality decimation results with high decimation performance by using incremental decimation based on the quadric error metric and

Figure 4: Decimation results for the Happy Buddha model (from left to right): stream decimation (18,486 tri.), OEMM (18,338 tri.), QEM (18,338 tri.) and OOCS (20,950 tri.). Their corresponding relative Hausdorff maximum errors (over the bounding box diagonal) are 0.89%, 0.89%, 0.87% and 1.12% respectively.

by processing the input data in one single pass. The small memory requirements make it possible to run the algorithm even on low-end PCs. On the other hand, since the requirements can be adapted to the available system resources, we can tune the algorithm to increase the performance on high-end PCs by using more RAM.

We used the quadric error metric in our implementation for efficiency reasons. However, the algorithmic structure is flexible enough to include any other more sophisticated distance metric as well [13]. Another convenient feature is that our algorithm can prescribe exactly the number of output triangles while vertex clustering techniques can only prescribe approximate values. In our experiments, all massive datasets could be processed properly without any pre-sorting.

One difficulty is that triangles adjacent to the boundary polygon P_{ab} have to be excluded from the decimation since the neighborhoods of the respective vertices are not yet complete. This however implies that boundaries in the original mesh cannot be decimated at all since the stream algorithm cannot distinguish true mesh boundaries and temporary boundaries in P_{ab} before the input stream is read completely. Nevertheless since triangles adjacent to P_{ab} are also no candidates for being written to the output, they stay in the in-core buffer until the input stream is empty and can then be decimated before the buffer is cleared.

The more serious limitation is that the in-core buffer has to store all the boundary triangles of the input mesh. In the case of the St. Matthew model this was the reason why we had to increase the buffer size: Due to imperfect 3D reconstruction this model has a large number of small holes and hence many boundary triangles. One possible way to address this issue is to introduce a preprocessing step to tag such boundaries and this step might even be embedded in the model generation procedure.

Acknowledgements

We would like to thank the Stanford Graphics Group and the Digital Michelangelo Project for providing the datasets. Many thanks to Mario Botsch for insightful discussions, to Loïc Barthe for proofreading the paper and to anonymous reviewers for their constructive critiques. The first author was funded by the DFG Graduate College “Software for Communication Systems” at RWTH-Aachen, Germany.

References

[1] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 24–33, 1994.

[2] Dmitry Brodsky and Benjamin Watson. Model simplification through refinement. In *Graphics Interface 2000 Proceedings*, pages 221–228, May 2000.

[3] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges - a scalable representation for triangle meshes. *ACM Journal of Graphics Tools*, 3(4):1–12, 1998.

[4] Paolo Cignoni, Claudio Rocchini, Claudio Montani, and Roberto Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, to appear, 2002.

Figure 5: Detail view of the decimation results for the David $1mm$ model with $56M$ triangles in the original data: stream decimation (562,219 faces, top), OEMM (500,000 faces, middle), and OOCS (578,503 faces, bottom). Left and right column show the same models with different shading.

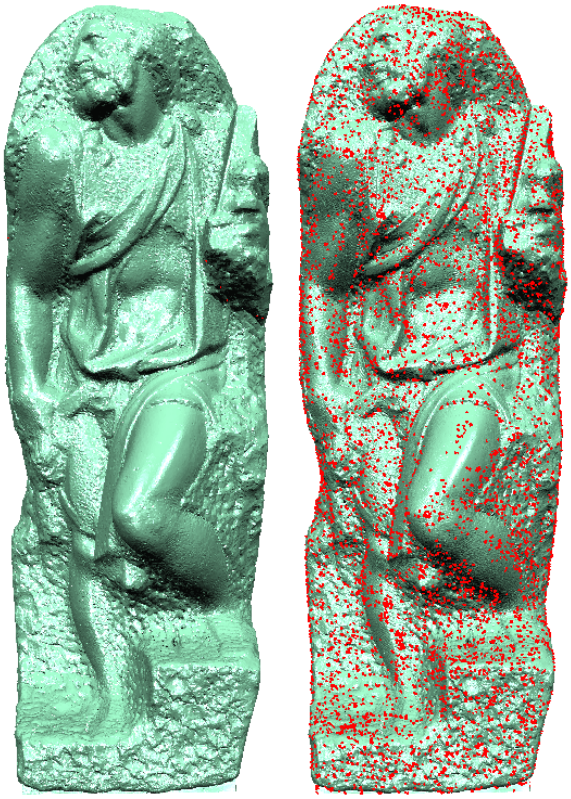

Figure 6: Results shown with topology errors (red/dark dots) for St. Matthew model by stream decimation (left, 1,863,837 tri.) and OOCS (right, 1,910,970 tri.). Note OOCS will generate much more *topology inconsistencies than our method.*

[5] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of ACM SIGGRAPH 1996*, Computer Graphics Proceedings, Annual Conference Series, pages 303–312. ACM, ACM Press / ACM SIGGRAPH, 1996.

[6] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3):139–150, 2000.

[7] Michael Garland. Multiresolution modeling: Survey & future opportunities. In *EUROGRAPHICS 99, State of the Art Report(STAR)*, pages 111–131. Eurographics Association, Aire-la-Ville (CH), 1999.

[8] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH 1997*, Computer Graphics Proceedings, Annual Conference Series, pages 209–216. ACM, ACM Press / ACM SIGGRAPH, 1997.

[9] Michael Garland and Eric Shaffer. A multiphase approach to efficient surface simplification. In *IEEE Visualization 2002 Conference Proceedings*, pages 117–124. IEEE, 2002.

[10] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. Simplification and compression of 3d-meshes. In Armin Iske, Ewald Quak, and Michal Floater, editors, *Tutorials on Multiresolution in Geometric Modeling*. Springer, 2002.

[11] Huges Hoppe. Smooth view-dependant level-of-detail control and its application to terrain rendering. In *IEEE Visualization 98 Conference Proceedings*, pages 35–52. IEEE, October 1998.

[12] Hugues Hoppe. Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108. ACM, ACM Press / ACM SIGGRAPH, 1996.

[13] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics Interface '98 Proceedings*, pages 43–50. Canadian Human-Computer Communications Society, A K Peters., June 1998.

[14] Peter Lindstrom. Out-of-core simplification of large polygonal models. In Kurt Akeley, editor, *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 259–262, New York, 2000. ACM, ACM Press / ACM SIGGRAPH.

[15] Peter Lindstrom and Claudio T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001 Conference Proceedings*, pages 121–126. IEEE, October 2001.

[16] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):163–169, July 1987.

[17] Michael Mitzenmacher, Andrea W. Richa, and Remesh Sitaraman. The power of two random choices: A survey of the techniques and results. In *Handbook of Randomized Computing*. Kluwer Press, 2002.

[18] Chris Prince. Progressive meshes for large models of arbitrary topology. Master's thesis, University of Washington, 2000.

[19] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, New York, 1993. Springer-Verlag.

[20] Eric Shaffer and Michael Garland. Efficient adaptive simplification of massive meshes. In *IEEE Visualization 2001 Conference Proceedings*, pages 127–134. IEEE, October 2001.

[21] Benjamin Watson. Out of core simplification. In *Advanced Issues in Level of Detail*, ACM SIGGRAPH 2002 course notes # 14, 2002.

[22] Jianhua Wu and Leif Kobbelt. Fast mesh decimation by multiple-choice techniques. In Guenther Greiner, Heinrich Niemann, Thomas Ertl, Bernd Girod, and Hans-Peter Seidel, editors, *Vision, Modeling, Visualization 2002 Proceedings*, pages 241–248, November 2002.