Fast Mesh Decimation by Multiple-Choice Techniques

Jianhua Wu Leif Kobbelt

Department of Computer Science VIII, Computer Graphics and Multimedia,
Aachen University of Technology (RWTH-Aachen),
Ahornstraße 55, 52074 Aachen, Germany
Email: {wu,kobbelt}@cs.rwth-aachen.de

Abstract

We present a new mesh decimation framework which is based on the probabilistic optimization technique of Multiple-Choice algorithms. While producing the same expected quality of the output meshes, the Multiple-Choice approach leads to a significant speed-up compared to the well-established standard framework for mesh decimation as a greedy optimization scheme. Moreover, Multiple-Choice decimation does not require a global priority queue data structure which reduces the memory overhead and simplifies the algorithmic structure. We explain why and how the Multiple-Choice optimization works well for the mesh decimation problem and give a detailed CPU profile analysis to explain where the speed-up comes from.

1 Introduction

Since the complexity of polygon mesh datasets (emerging, e.g., from 3D scanning or iso-surface extraction) is increasing much faster than the rendering performance of graphics hardware, mesh decimation techniques have been an active research area over at least the last decade [5, 7]. After many different approaches have been investigated and compared, today two basic concepts can be considered as the standard solutions: One concept is *vertex clustering* and the other is *incremental decimation*.

While vertex clustering [17, 13] is very fast and effective, the quality of the resulting meshes is often not satisfying. The major drawbacks of this approach are that it usually leads to a vertex distribution which does not adapt to the local curvature of the surface and that it cannot guarantee a proper manifold topology of the resulting mesh. On the other hand, incremental decimation [8, 6, 20], where one atomic decimation step is executed after the other, typically leads to superior mesh quality in terms of approximation error for a prescribed triangle count as well as triangle count for a prescribed approximation error. In addition, incremental decimation can guarantee the preservation of the initial topology.

Although there are many different incremental decimation schemes in the literature, they all follow the same basic principle that each possible atomic decimation operation (candidate) is rated according to some quality criterion. Then in every step the "best" candidate is decimated which triggers new evaluations of the quality criterion in its vicinity. As observed in [10], the decimation problem can be understood as an instance of the knapsack-problem with the number of decimation operations being the objective function and the geometric approximation error being the capacity function.

Obviously, finding the *optimal* decimation sequence is a very complex problem [1] and consequently one has to find solutions with *approximate optimality*. The above best-first strategy is, in fact, a greedy strategy to find a decimation sequence that is close to optimal.

In this paper, we are using a different optimization strategy to address the mesh decimation problem. Instead of doing greedy optimization (which requires to find the best choice among *all* candidates) we are using a Multiple-Choice paradigm (which requires to find the best choice only among a *small subset* of the candidates).

The motivation for using this probabilistic optimization strategy is the fact that when decimating high resolution meshes, most of the vertices will be removed anyway – usually 90% to 99% of the original data. Hence it is not necessary to look at all possible candidates in every decimation step.

Let us e.g. assume that we decimate a given 3D model down to 5% of the original complexity. The basic idea of Multiple-Choice techniques is to pick a small random subset of candidates, say 8 possible edge collapses, and then perform the best of them. This strategy leads to a wrong decision only in the rare case when all 8 collapses do not belong to the 95% majority of candidates that are supposed to be removed. The probability for a wrong decision is hence

$$\left(\frac{5}{100}\right)^8 \approx 10^{-11},$$

which implies that a reliable decision if a certain atomic decimation step is part of the optimal decimation sequence or not can be based on a small subset of candidates. Notice that for the above estimate we exploit the fact the most atomic operations are independent from each other and the exact order of the decimation steps matters only for direct neighbors.

The major benefit of the Multiple-Choice optimization compared to the greedy optimization is that the algorithmic structure is much simpler. An implementation of the greedy strategy usually requires a priority queue data structure for the candidates that has to be initialized and updated during the decimation (whenever the priorities/qualities of the candidates are re-evaluated). For the Multiple-Choice optimization we do not need a priority queue and consequently we save memory space and computation time. Our experiments show that the Multiple-Choice decimation is more than a factor of 2.5 times faster than a highly optimized greedy implementation (both using the QEM quality criterion [6]). While producing the same quality of the output meshes as the QEM-based greedy optimization, our multiple choice approach can run at decimation rates of more than 70K triangles per second on a standard PC.

2 Multiple-Choice Techniques

Multiple-Choice algorithms (MCA) are a probabilistic optimization technique that has been investigated thoroughly in the fields of telecommunication, distributed systems, and theoretical computer science. The fundamental idea behind MCA is quite simple and intuitive and can be explained best using the well-established bins-and-balls model [2, 18] (cf. Fig. 1).

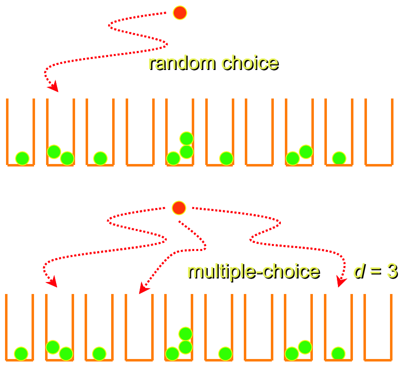

Figure 1: Bins-and-balls model. Top: random ball insertion; Bottom: multiple-choice ball insertion.

Consider we have n balls to be uniformly distributed over n bins, i.e., each ball is put into a random bin and for each ball, the destination bin is selected independently from earlier choices (cf. Fig. 1 Top). Obviously, the expected number of balls in each bin would be one. Moreover, a more detailed probability analysis of this random allocation procedure shows that the *maximum load*, i.e., the expected number of balls in the fullest bin, will be

$$(1 + o(1))\frac{\ln(n)}{\ln(\ln(n))},$$

with high probability [12].

Instead of putting each ball into an independently selected bin, the idea of MCA is to choose a small random subset of d bins and then put the ball into that bin with the least number of balls already in it [18] (cf. Fig. 1 Bottom). By this MCA strategy, we can guarantee that the maximum load is

$$\frac{\ln(\ln(n))}{\ln(d)} + O(1),$$

(see [2] for detailed proof) which is an exponential improvement compared to the pure random approach.

After their discovery, MCA have been used as an optimization tool in many different applications. For example, Karp et al [9] used them for efficient hashing in the context of shared memory computer simulation. In this application, the balls are just the hash-keys and the bins stand for table entries. The maximum load is the maximum length of the collision chains in the hash table.

Another application example is data allocation and data management [15]. Here the balls represent data objects and the bins are memory blocks or disk sectors. The maximum number of requests to the same storage location corresponds to the maximum load.

In parallel and distributed systems, MCA have been adopted to online load balancing [2]: the balls represent jobs, the bins are online computers, and the maximum load is the maximum number of jobs per machine. Other applications include routing in networks [4], queueing processes [19], etc. A thorough survey of Multiple-Choice optimization techniques can be found in [16].

So far we are not aware of any application of MCA to optimization problems in the field of computer graphics. In this paper we are using MCA to find approximately optimal solutions in the context of mesh decimation.

3 Multiple-Choice Decimation

In order to apply MCA to the mesh decimation problem we have to map balls, bins, and maximum load to the corresponding mesh entities. Since the balls are enumerated in the outer loop ("*for each ball make a MC decision*") they correspond to the decimation steps. The bins represent the possible choices in each step and hence they correspond to the possible candidates. The maximum load finally is the value that is to be optimized and consequently we associate it with the quality criterion that is used to rate the candidates (cf. Table 1).

Table 1: Multiple-Choice decimation correspondence to the balls-and-bins model.

balls-and-bins model	**Multiple-Choice decimation**
balls	atomic decimation steps
bins	candidate operators
maximum load	maximum *approxiamtion* error

In this setup, the MCA approach to mesh decimation consists of testing a small set of *d* randomly selected candidates (e.g. edge collapses) in each step and performing that decimation operation among this small set that has the best quality value.

The parameter *d* in the above description controls how good the MCA approximates the standard greedy approach. In fact, for $d = n$ both algorithms are identical and the priority queue mechanism used in the greedy approach has only the purpose of speeding up the algorithm since it allows to reuse the ordering of the unmodified quality values from the previous decimation step.

The statistical analysis of MCA (Sect. 2) implies that even for relatively small values d the approximation of MCA is very good. As will be demonstrated in Section 5, the mesh quality that we obtain with MCA is already indistinguishable from the greedy approach if we choose $d = 8$. With such a small number of candidates to be tested in every step, we do not need a global data structure like a priority queue to speed up the computation because the re-computation of the ordering is actually less expensive than updating the global queue. This reduces the memory consumption and makes the algorithm much easier to implement.

4 Decimation Algorithm

Using the above idea of Multiple-Choice decimation, the overall framework of our incremental decimation algorithm is as follows:

1. Read (part of) the mesh model into the main memory.
2. Randomly choose *d* atomic decimation operators from all candidates, compute their respective decimation costs using a specific error/quality metric.
3. Select that operator with smallest cost/error and perform this operator
4. Repeat step 2 and 3, until the user-defined terminating condition is met.

This generic framework is somehow similar with those previous ones (except the Multiple-Choice optimization strategy), but it is much easier to implement and it does not put any limitations on the choice of the atomic decimation operator or on the error or quality metric.

In our implementation we use the error quadrics (QEM) introduced in [6] to rate the quality of each candidate. The atomic decimation operator is the half-edge collapse that does not introduce new vertex positions but rather sub-samples the original mesh [10] (see Fig. 2). We prefer half-edge collapses since they make progressive transmission more efficient (no intermediate vertex coordinates) and enable the construction of nested hierarchies on unstructured meshes [11] that can facilitate further applications.

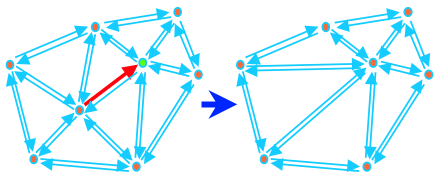

Figure 2: Half-edge collapse operator.

Our implementation is based on the OpenMesh data structure [3] which is a generic edge-based polygon mesh data structure. We could even increase the performance of our implementation by switching to a more specialized data structure but we decided to use OpenMesh for software design reasons.

In Table 2, we summarize and compare the greedy and the MCA implementation of the QEM-based decimation. It is obvious to see that the MCA decimation does much less work than the greedy version and it can be expected to be much faster. Moreover, since both algorithms are using the same quality criterion, they are producing very similar decimation results.

Table 2: Multiple-Choice decimation algorithm compared with greedy simplification.

	greedy	**MCA**
Initialize	initialize quadrics, evaluate quality for all candidates, perform global queue sorting	initialize quadrics
Select candidate	top of the queue	best out of *d* random choices
Decimate	perform operator, locally recompute qualities, update global queue	perform operator

5 Comparisons and Results

We compare our MCA decimation scheme to a highly optimized implementation of the greedy decimation scheme. Both implementations are using the same underlying mesh data structure, the same routines for the QEM evaluation and the same half-edge collapse procedure. Hence the differences in running time and memory allocation are only due to the different optimization principles.

All experiments were done on a commodity PC with AMD 1700+ CPU and 1 GB RAM. Since the MCA is a randomized algorithm (and since the system performance usually varies slightly due to background processes competing for CPU time) we let each experiment run five times and then took the *median* of the respective timings to eliminate outliers.

The parameter *d* in the MCA is set to *8* in all experiments. With this value we obtain a very similar output quality for both approaches and the variance for the MCA is sufficiently low.

The approximation error is measured by the Hausdorff-distance between the original mesh and the decimation result. We chose the maximum error instead of some average since this value is more relevant in most technical applications (maximum tolerance).

The timings (excluding reading from and writing to disk) and errors for various models are summarized in Table 4. The MCA algorithm is about a factor of **2.5 times** faster than the greedy version. It reaches a maximum performance of up to **70K** decimated triangles per second. We also depict the absolute maximum geometric errors for the Bunny model (Figure 3) and the Max model (Figure 4) when decimating them to various levels of details by the two different *algorihtms*. Whether the models are drastically decimated or not, the accuracy of the *MCA algorihtm* is almost identical to that of the greedy version.

To obtain a better understanding of which subtasks in the decimation algorithms are using the most CPU time, we run a detailed profiling on the greedy and the MCA version when processing the same model (the bunny). The results are given in Table 3. It turns out that in the MCA version the evaluation procedure for the QEM as well as the procedure that checks if a given candidate collapse is legal (topological consistency, normal flipping)

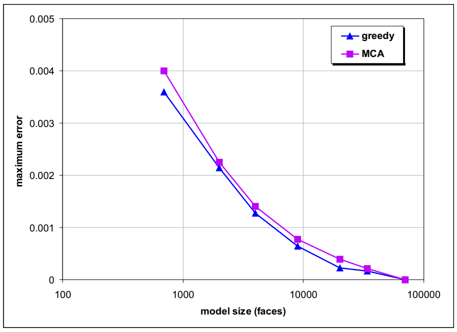

Figure 3: Absolute maximum geometric error for Bunny model. The diagonal length of bounding box is *0.25*.

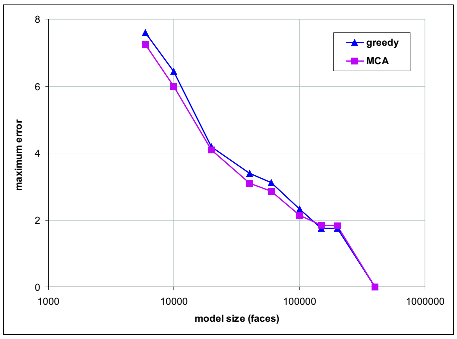

Figure 4: Absolute maximum geometric error for Max model. The diagonal length of bounding box is *450.35*.

are executed less frequently than in the greedy version. This indicates that a certain portion of these evaluations and tests during the update of the priority queue are redundant, i.e., the quality rating of a candidate is updated several times (triggered by collapses in the vicinity) before it is actually considered for decimation. Of course, this redundancy cannot be avoided completely in the MCA version but it is obviously reduced.

Another observation is that the best-of-*d* selection (*d=8*) takes much less time than the updating of the priority queue which justifies the claim that recomputing the ordering in the MCA version is less expensive than reusing the ordering of the unmodified part from the previous step as it is done in the greedy version.

Table 3: Running time profiles of Multiple-Choice decimation and greedy decimation. Notice that the total time is higher than in Table 4 because of the frequent time measurements during runtime.

Stages	**MCA**		**greedy**	
	t(s)	%	t(s)	%
init. quadrics	0.143	12.0	0.144	5.9
eval. quadrics	0.625	52.6	0.874	35.6
chk. collapse	0.246	20.7	0.408	16.6
update mesh	0.048	4.0	0.050	2.0
best out of *d*	0.127	10.7	–	–
update queue	–	–	0.980	39.9
total	1.189	100	2.456	100

Analysing the memory usage, we find that for QEM-based decimation algorithms and a mesh model with *n* vertices, we need *10*4*n* Bytes for the quadrics (double precision) and *3*8*n* Bytes for the edge-based priority queue (*3n* edges, each has *4* Bytes priority value and *4* Bytes edge pointer) without considering the storage for the mesh itself. In total, the greedy implementation requires at least *64n* Bytes memory overhead. Our MCA version instead only uses *40n* Bytes for the quadrics which reduces the memory overhead by *37.5%*. Even compared with the well-known memoryless simplification method [14], which only needs *24n* Bytes for a priority queue, our approach is still acceptable with a much faster speed. In addition, since the current memory overhead of MCA is just used for candidate ordering and the MCA strategy is independent of that, we can remove this memory overhead completely by adopting the cost functions in [14] if needed.

The Figures 5 to 7 demonstrate that the visual quality of the decimated meshes generated by the MCA version are not distinguishable from the result of the greedy version. In Figure 6, the dragon model is drastically simplified (99%) but the difference between both versions is still hardly perceivable. Since both versions are based on the same quality criterion, they preserve the same amount of geometric detail (cf. Fig. 7).

Table 4: Time and error performance of Multiple-Choice decimation algorithm and greedy decimation method both based on the quadric error metric and half-edge collapses.

models	**triangles**		**maximum error**			**runing time(s)**			**speed(tri/s)**
	input	output	greedy	MCA	gr/M	greedy	MCA	gr/M	MCA
bunny	69666	696	0.0036	0.0040	0.90	2.18	0.97	2.25	71821
sucker	230141	10138	2.5	2.62	0.96	7.27	3.28	2.21	70165
max	398043	5918	7.60	7.25	1.05	15.19	5.97	2.54	66674
dragon	871414	10520	0.0012	0.0011	1.09	37.85	15.03	2.52	57978
buddha	1087469	16412	0.0015	0.0014	1.08	49.37	18.58	2.66	58529

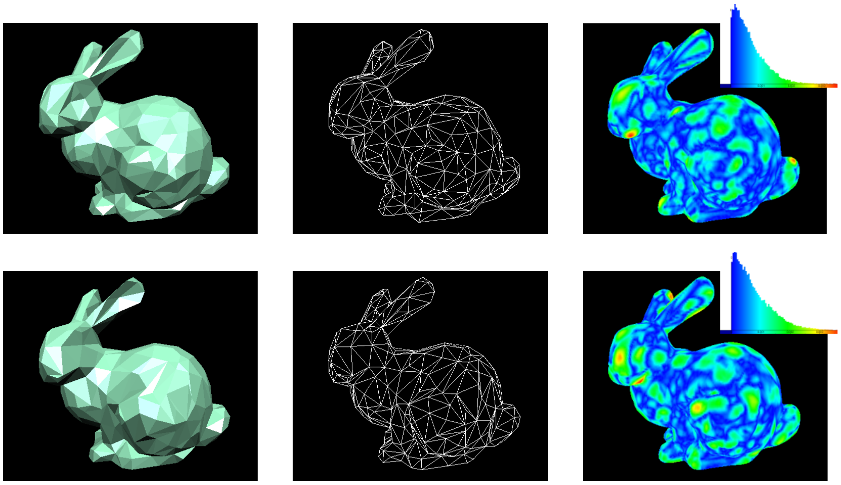

Figure 5: The bunny model decimated to 696 triangles shown with flat shading, hidden-lines and error distribution. The upper row is generated by the greedy version, the lower row by the MCA version.

6 Conclusion

In this paper we applied the generic probabilistic optimization principle of Multiple-Choice algorithms to the problem of incremental mesh decimation. We compared the resulting algorithm to the standard mesh decimation algorithm which is based on the greedy optimization principle. Our results and their discussion show that the MCA approach leads to a simpler algorithmic structure (no priority queue data structure), it runs significantly faster (fewer redundant computations, no global update) and it produces much less memory overhead.

Our detailed CPU time profile analysis with different models and on different computers reveals some interesting observations and directions for future research. We let the MCA and the greedy version run on two different computers with identical bus performance (133 MHz SDRAM) but different CPUs. One CPU is an 866 MHz Intel PIII and the other one is an ADM 1700+ which is claimed to be twice as fast as the other one. When decimating the bunny model down to 700 triangles we obtained the following average timings:

	MCA	**greedy**
Intel 866	1.12s	2.93s
AMD 1700+	0.97s	2.18s

If we believe that the pure computation is really twice as fast on the AMD 1700+ and if we take the

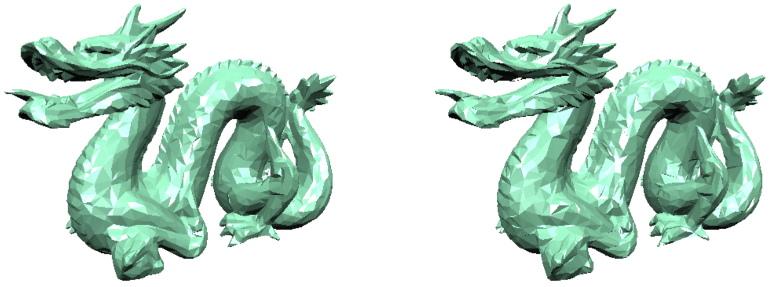
Figure 6: Decimated dragons with 10520 triangles by greedy optimization (left) and by Multiple-Choice optimization (right).

fact into account that both computers have the same bus performance then we conclude that the MCA version spends 0.82 sec with memory accesses and the greedy version spends 1.53 sec with memory accesses. Both numbers are computed by linear interpolation of the two timings at 866 MHz and 1700 MHz, e.g. if a is the computation time on the AMD processor and b is the common memory access time then we have to solve the system,

Intel 866: $2a + b = 1.12$

AMD 1700+: $a + b = 0.97$

which implies that 0.97 - 0.15 = 0.82 sec is the portion of the running time that does not depend on the processor speed. Hence, from the CPU point of view, the MCA version is more than four times faster than the greedy version (0.15 sec vs. 0.65 sec) but from the memory access point of view it is only faster by a factor of two (0.82 sec vs. 1.53 sec).

Since the AMD 1700+ computation time (0.15 sec) is more that five times lower than the memory access time (0.82 sec) in the MCA version, future improvements of the decimation performance should rather aim at reducing the memory accesses instead of trying to speed up the computation.

Acknowledgements

The first author is supported by the DFG Graduate College “Software for Communication Systems” at the RWTH-Aachen in Germany.

References

[1] P. Agarwal, S. Suri, “Surface Approximation and Geometric Partitions”, *Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms*, pp.24-33, 1994.

[2] Y. Azar, A. Broder, A. Karlin, and E. Upfal, “Balanced Allocations”, *SIAM Journal on Computing*, 29(1):180-200, 1999.

[3] M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, “OpenMesh – a Generic and Efficient Polygon Mesh Data Structure”, *OpenSG Symposium 2002*, 2002.

[4] R. Cole, B. Maggs, F. Meyer auf der Heide, etc, “Randomized Protocols for Low-congestion Circuit Routing in Multistage interconnection Networks”, *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pp.378-388, 1998.

[5] M. Garland, “Multiresolution Modeling: Survey & Future Opportunities”, *State of the Art*, Eurographics, pp.111-131, 1999.

[6] M. Garland, P. Heckbert, “Surface Simplification Using Quadric Error Metrics”, *SIGGRPAH97 Conference Proceedings*, pp.209-216, 1997.

[7] C. Gotsman, S. Gumhold, L. Kobbelt, “Simplification and Compression of 3D Models”, *Tutorials on Multiresolution in Geometric Modeling*, Springer, to appear, 2002.

[8] H. Hoppe, “Progressive Meshes”, *SIGGRPAH96 Conference Proceedings*, pp.99-108, 1996.

[9] R. Karp, M. Luby, F. Meyer auf der Heide, “Efficient PRAM Simulation on a Distributed

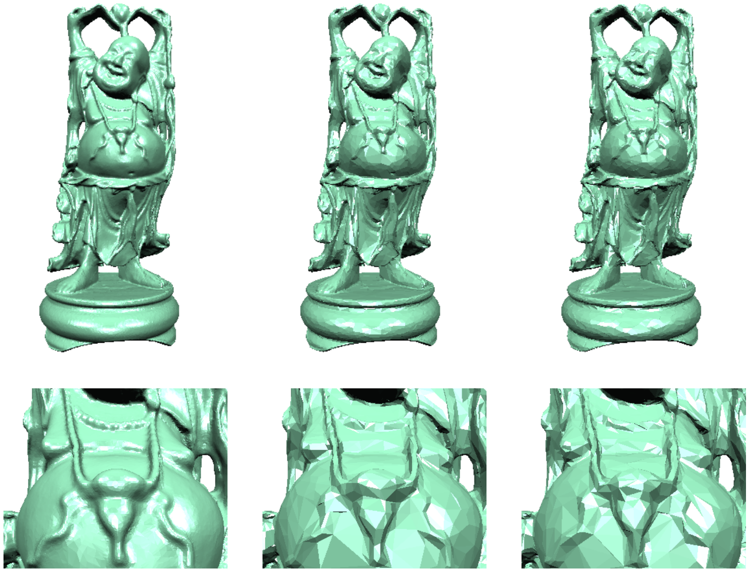

Figure 7: The original buddha model with 1087649 triangles (left) and simplified ones with 32402 triangles by greedy (middle) and MCA (right). The lower row shows zoom-in views of the above models.

Memory Machine", *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, pp.318-326, 1992.

[10] L. Kobbelt, S. Campagna, H. P. Seidel, "A General Framework for Mesh Decimation", *Graphics Interface'98 Proceedings*, pp.43-50, 1998.

[11] L. Kobbelt, S. Campagna, J. Vorsatz, H. P. Seidel, "Interactive Multi-Resolution Modeling on Arbitrary Meshes", *SIGGRPAH98 Conference Proceedings*, pp.105-114, 1998.

[12] V. Kolchin, B. Sevastyanov, V. Chistyakov, *Random Allocations*, John Willey & Sons, 1978.

[13] P. Lindstrom, "Out-of-core Simplification of Large Polygonal Models", *SIGGRAPH2000 Conference Proceedings*, pp.259-262, 2000.

[14] P. Lindstrom, G. Turk, "Fast and Memory Efficient Polygonal Simplification", *IEEE Visualization'98 Proceedings*, pp.279-286, 1998.

[15] F. Meyer auf der Heide, C. Scheideler, V. Stemann, "Exploiting Storage Redundancy to Speed Up Randomized Share Memory Simulations", *Theoretical Computer Science*, 162: 245-281, 1996.

[16] M. Mitzenmacher, A. Richa, R. Sitaraman, *Handbook of Randomized Computing*, chapter The Power of Two Random Choices: A Survey of the Techniques and Results, Kluwer Press, to appear, 2002.

[17] J. Rossignac, P. Borrel, "Multiresolution 3D Approximations for Rendering Complex Scenes", *Modeling in Computer Graphics: Methods and Applications*, pp.445-465, 1993.

[18] B. Vöcking, "Symmetric vs Asymmetric Multiple-Choice Algorithms", invited paper, *Proceedings of 2nd ARACNE workshop (Aarhus, 2001)*, pp.7-15, 2001.

[19] N. Vvedenskaya, R. Dobrushin, F. Karpelevich, "Queueing Systems with Selection of the Shortest of Tow Queues: An Assymptotic Approach", *Problems of Information Transmission*, 32(1):15-27, 1996.

[20] J. Wu, C. Tai, S. Hu, etc, "An Effective Feature-preserving Mesh Simplification Scheme Based on Face Constriction", *Pacific Graphics 2001 Proceedings*, IEEE press, pp.12-21, 2001.