A Case Study of Improving Memory Locality In Polygonal Model Simplification: Metrics and Performance

Victor Salamon[1], Paul Lu[1], Ben Watson[2], Dima Brodsky[3], Dave Gomboc[1]

[1] Dept. of Computing Science, University of Alberta,
Edmonton, AB, Canada, T6G 2E8, {salamon,paullu,dave}@cs.ualberta.ca
[2] Dept. of Computer Science, Northwestern University,
Evanston, IL, USA, watsonb@cs.nwu.edu
[3] Dept. of Computer Science, University of British Columbia,
Vancouver, BC, Canada, dima@cs.ubc.ca

Abstract. Polygonal model simplification algorithms take a full-sized polygonal model as input and output a less-detailed version of the model with fewer polygons. When the internal data structures for the input model are larger than main memory, many simplification algorithms suffer from poor performance due to paging.
We present a case study of the recently-introduced *R-Simp* algorithm and how its data locality and performance can be substantially improved through an off-line spatial sort and an on-line reorganization of its internal data structures. When both techniques are used, *R-Simp*'s performance improves by up to 7-fold. We empirically characterize the data-access pattern of *R-Simp* and present an application-specific metric, called *cluster pagespan*, of *R-Simp*'s locality of memory reference.

1 Introduction

Trade-offs between quality and performance are important issues in real-time computer graphics and visualization. In general, the more polygons in a three-dimensional (3D) computer graphics model, the more detailed and realistic is the rendered image. However, the same level of detail and realism may not be required in all computer graphics applications. For real-time display, fewer polygons result in a faster rendering; rendering speed may be the most important criteria. For other applications, the speed of rendering may be traded off for improved image quality. The flexibility to select a version of the same model with a different level of detail (i.e., a different number of polygons) can be important when designing a graphics system.

Polygonal model simplification algorithms take a full-sized polygonal model as input and output a version of the model with fewer polygons. Although the simplified models are often of high quality, it is also clear that, upon close examination, some details have been sacrificed to reduce the size of the model (Figure 1). A number of model simplification algorithms have been proposed [3],

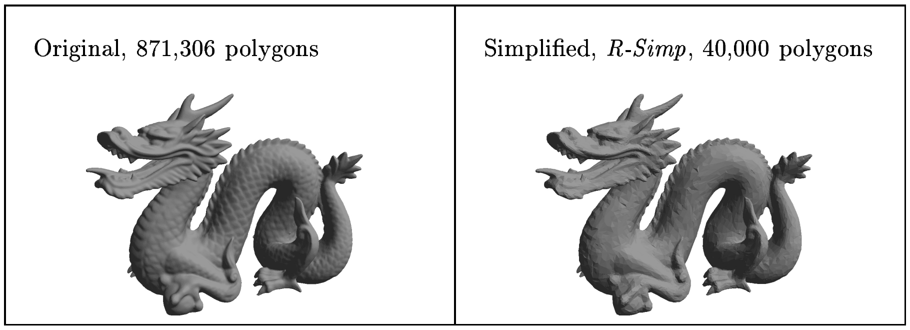

Original, 871,306 polygons
Simplified, *R-Simp*, 40,000 polygons

Fig. 1. Original and Simplified Versions of Polygonal Models: *dragon*

each with their different strengths and weaknesses in terms of execution time and the resulting image quality. In general, the algorithms are used to pre-compute simplified versions of the models that can be selected for use at run-time.

However, models that have over a million polygons require tens of megabytes of disk storage, and require hundreds of megabytes of storage for their in-memory data structures (Table 1). For example, the *blade* model has almost 1.8 million polygons and requires 331.3 megabytes (MB) of virtual memory after being read in from an 80.1 MB disk file.[1]

Consequently, we examine the problem of data-access locality and its effect on performance for a model simplification algorithm. In particular, we make a case study of the recently-introduced *R-Simp* simplification algorithm. We focus on systems-oriented run-time performance and related metrics of *R-Simp*, as opposed to measures of model quality.

1.1 Motivation and Related Work: Large Models

As new model acquisition technologies have developed, the complexity and size of 3D polygonal models have increased. First, model producers have begun to use 3D scanners (for example, [5]). Second, as the speed and resolution of scientific instrumentation increases, so has the size of the data sets that must be visualized. Both of these technology changes have resulted in models with hundreds of millions to billions of polygons. Current hardware cannot come close to displaying these models in real time. Consequently, there is a large body of "model simplification" research addressing this problem [3].

Most of these model simplification algorithms (e.g., [9, 4, 2]) use a greedy search approach with a time complexity of O(n log n), where n is the number of polygons in the original model. Also, the greedy algorithms have poor locality of memory access, jumping around the surface of the input model, from puzzle piece to puzzle piece. One exception to this trend is the simplification algorithm

[1] Initial virtual memory size is read from the `/proc` file system's `stat` device under Linux 2.2.12 before any simplification computation is performed.

Model	Faces (Polygons)	Vertices	Initial Virtual Memory Size (MB)	File Size for PLY Model (MB) [7]
hand	654,666	327,323	123.6	31.1
dragon	871,306	435,545	164.1	32.2
blade	1,765,388	882,954	331.3	80.1

Table 1. Summary of Full-Sized Polygonal Models

described by Lindstrom [6], based on the algorithm by Rossignac and Borrel [8]. Lindstrom's algorithm is fast, but it produces simplified models of poor quality and, by its nature, it is difficult to control the exact number of polygons in the simplified model.

In contrast, the recently-introduced *R-Simp* algorithm [1] produces approximated models of substantially higher quality than those produced by Rossignac and Borrel's approach, and *R-Simp* allows for exact control of output size, all without a severe cost in execution speed. *R-Simp* is unique in that it iteratively refines an initial and poor approximation, rather than simplifying the full input model. Consequently, *R-Simp* has a time complexity of $O(n \log m)$, rather than $O(n \log n)$, where m is the number of polygons outputted in the simplified model. Since $m \ll n$ in practice, *R-Simp*'s advantage can be substantial.

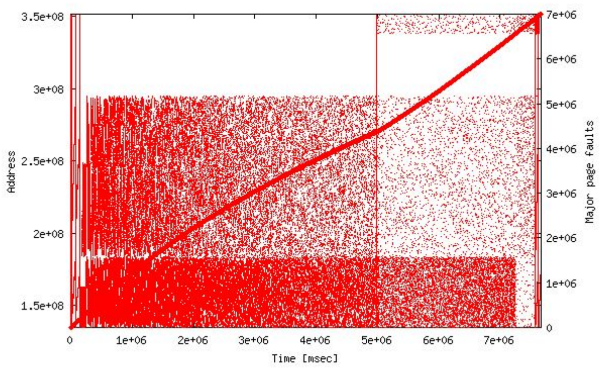
2 The Problem: Locality of Memory Accesses

To understand the data-access patterns of *R-Simp*, we hand-instrumented a version of the code such that an on-line trace is produced of all the model-related memory accesses. Each memory access is timestamped using the value returned by the Unix system call `gettimeofday()`, where the first access is fixed at time 0. The ability to map and label a memory access to a specific *R-Simp* data structure is beyond the ability of most standard tools. The resulting trace file is processed off-line.

A scatter plot of the virtual address of each memory access versus the timestamp shows a "white noise" pattern (Figure 2), which indicates relatively poor locality of reference due to a large working set. The graph intuitively explains why paging is a performance problem for models that do *not* fit in main memory. We only show the scatter plot for *dragon* as the plots for the other two models are similar. The hardware platform is a 500 MHz Pentium III system, running Linux 2.2.12, with 128 MB of RAM, and a IDE swap disk. Although there are dual processors, *R-Simp* is a single-threaded application. All reported real times are the average of 5 runs.

The instrumentation and tracing adds significant overhead to *R-Simp*'s execution time. Consequently, we have scaled the X-axis. For example, the non-instrumented *R-Simp* requires 7,736 seconds to compute the simplified model for *dragon*. The instrumented *R-Simp* requires much more time to execute, but we have post-processed the trace information so that the X-axis appears to be 7,736

X-axis is scaled real time in milliseconds,
LHS Y-axis is virtual address (scatter plot),
RHS Y-axis is cumulative major page faults (solid curve)
in the instrumented version
NOTE: X and Y-axes have different ranges for each graph



Dragon: 7,736 seconds run-time, 6,410,393 major page faults
in non-instrumented version

Fig. 2. Memory-Access Pattern for Original *R-Simp* on *dragon*: Computing 40,000 polygon simplified model

seconds. Although the scaling may introduce some distortions to the figure, it still captures the basic nature of the memory-access patterns.

Figure 2 has some notable patterns. The darker horizontal band, between addresses 1.3×10^8 and 1.8×10^8 on the LHS Y-axis, is the region of virtual memory where the data structure for the vertex list is stored. The horizontal region above the vertices is where the face list is stored. The data structures and details of *R-Simp* are discussed in Section 3. There is a vertical line at time 5.0×10^6 milliseconds for *dragon*, which is when *R-Simp* finishes the simplification process and begins to create the new simplified model. The band of memory accesses at the top of the graph, and to the immediate right of the vertical line, is where the new model is stored.

Superimposed on Figure 2 is a solid curve representing the cumulative major page faults of *R-Simp* over time, as reported by the Unix function `getrusage()`. A major page fault may require a disk access to swap out the victim page and does require a disk access to swap in the needed page of data. Note that there is some discrepancy between the major page faults incurred by the instrumented *R-Simp* (Y-axis on the RHS) and the major page faults incurred by the non-instrumented *R-Simp* (noted in the captions of Figure 2 and Table 3). We have purposely not scaled the RHS Y-axis since that is more problematic than scaling real time. Again, the figures are only meant to give an intuitive picture of the data-access patterns inherent to *R-Simp*.

3 The *R-Simp* Algorithm

We briefly summarize the main data structures in *R-Simp* and the main phases of the computation [1].

3.1 Data Structures

R-Simp uses three main data structures to perform model simplification. The first two data structures are static and are used to store the original model. The last data structure, the *cluster*, is more dynamic.

First, the *vertex list* is a global array that stores the vertices from the input model. Each vertex contains (x, y, z) coordinates, an adjacency list of vertices, and an adjacency list of faces. Second, the *face list* is also a global array that stores the faces from the input model. Each face contains its normal, its area, and pointers to the vertices that make up the face. The global vertex and face lists are accessed throughout *R-Simp's* execution. Third, the *cluster* structure represents a portion of the original model and is a node in an n-ary tree. A cluster contains a list of vertices from the original model and other supplementary data. Iteratively, a leaf cluster is selected from a priority queue and subdivided into 2, 4, or 8 sub-clusters. At the end of the simplification phase, each cluster represents one vertex in the simplified model.

3.2 Phases of *R-Simp*

The first phase is the input phase. The original model is read in from the file and the initial vertex list and face list are created. These data structures require a lot of virtual memory (Table 1). The blocks of memory that are allocated in this phase are used throughout the entire simplification process (Section 2, Figure 2).

In the second phase, the in-memory data structures are initialized. The initialization phase creates the vertex and face adjacency lists.

The third phase is the heart of *R-Simp*: the simplification phase. In this phase the original set of vertices is reduced to the desired size. The simplification phase starts with all the vertices in a single cluster. This cluster is then subdivided into a maximum of eight sub-clusters. These clusters are inserted into a priority queue and the main loop of *R-Simp* begins.

The priority queue holds references to the clusters and orders the clusters based on the surface variation (i.e., amount of curvature). The greater the surface variation, the higher the priority of the cluster. Therefore, the simplified model will have more vertices in regions of high surface variation. This process iterates until the priority queue contains the required number of clusters, which is the number of vertices in the simplified model. Each split causes more clusters to

Model	Output (polygons)	Original *R-Simp*	w/Spatial Sort	w/Reorg.	w/Both
hand	10,000	1,401	854	305	270
hand	20,000	1,905	1,140	342	293
hand	40,000	2,597	1,582	410	353
dragon	10,000	4,375	1,524	1,409	892
dragon	20,000	5,829	2,058	1,778	1,183
dragon	40,000	7,736	2,781	2,484	1,563
blade	10,000	8,306	4,052	4,539	3,545
blade	20,000	10,877	5,289	5,700	4,792
blade	40,000	14,312	7,084	7,720	6,713

Table 2. Performance of Polygonal Model Simplification: Various Strategies, times in seconds

be created, thus the amount of memory used by this phase depends on the level of detail required. In general, the coarser the level of detail required, the less memory is needed.

The fourth phase is the post-simplification phase: the final set of vertices for the simplified model are computed. Each cluster represents a single vertex, v_{so}, in the simplified model; for each cluster, *R-Simp* computes the optimal position of v_{so}. Finally, the algorithm changes all of the pointers from the original vertices in a cluster to v_{so}.

The fifth phase is the triangulation phase, where the algorithm creates the faces (i.e., polygons) of the new simplified model. The algorithm iterates through all the faces in the original model and examines where the vertices of these faces lie in the output cluster. If two or more vertices point to the same new vertex (i.e., two or more original vertices are within the same cluster) then the face has degenerated and only if all three original vertices point to different new vertices do we keep the face and add it to the new face list. The vertices of this new face point to the vertices in the new vertex list.

In the last phase, the output phase, the algorithm writes the new vertex list and the new face list to the output file.

4 Improving Memory Locality and Performance in *R-Simp*

In addition to experiments with an instrumented version of *R-Simp*, we have also benchmarked versions of *R-Simp* without the instrumentation and with compiler optimizations (`-O`) turned on. *R-Simp* is written in C++ and we use the `egcs` compiler, version 2.91.66, on our Linux platform. The hardware environment is the same as in Section 2.

When the data structures of a model fit within main memory, *R-Simp* is known to have low run times [1]. However, the *hand*, *dragon*, and *blade* models

Model	Output (polygons)	Original *R-Simp*	w/Spatial Sort	w/Reorg.	w/Both
hand	10,000	1,042,444	701,789	265,328	249,849
hand	20,000	1,361,492	880,498	281,338	254,083
hand	40,000	1,812,649	1,167,593	297,028	268,952
dragon	10,000	3,670,516	1,409,489	1,070,466	702,050
dragon	20,000	4,892,932	1,808,060	1,329,220	875,622
dragon	40,000	6,410,393	2,386,238	1,767,490	1,084,346
blade	10,000	7,678,929	3,878,566	4,127,079	3,125,348
blade	20,000	9,988,417	5,002,567	5,195,597	4,149,098
blade	40,000	13,063,493	6,599,763	6,997,380	5,717,086

Table 3. Major Page Fault Count of Polygonal Model Simplification: Various Strategies

are large enough that they require more memory than is physically available (Table 1).[2] Consequently, the baseline *R-Simp* (called "Original *R-Simp*") experiences long run times (Table 2) due to the high number of page faults that it incurs (Table 3). As expected, the larger the input model, the longer the run time and the higher the number of page faults. As the output model's size increases, the run time and page faults also increase.

As previously discussed (Section 3), the global vertex and face lists are large data structures that are accessed throughout the computation. Therefore, they are natural targets of our attempts to improve the locality of memory access. In particular, we have developed two different techniques that independently improve the memory locality of *R-Simp*; real-time performance improves by a factor of 2 to 6-fold, depending on the model and the size of the simplified model. When combined, the two techniques can improve performance by up to 7-fold.

4.1 Metrics: Cluster Pagespan and Resident Working Set

We introduce *cluster pagespan* as an application-specific metric of the expected locality of reference to a model's in-memory data structure. Cluster pagespan is defined as the number of *unique* virtual memory pages that have to be accessed in order to touch all of the vertices and faces, in the global vertex and face lists, in the cluster. For each iteration of the simplification phase, *R-Simp*'s computation is focussed on a single cluster from the front of the priority queue. Therefore, if the pagespan of the cluster is large, there is a greater chance that a page fault will be incurred. The smaller the cluster pagespan, the lower the chance that one of its pages has been paged out by the operating system.

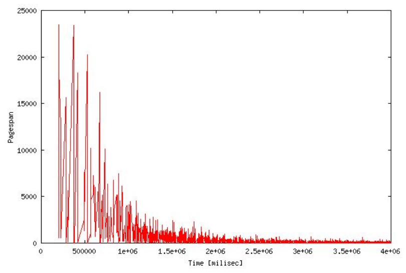
Figure 3(a) shows the cluster pagespan of the cluster at the front of the priority queue during an execution of *R-Simp* with the *blade* model. Since the

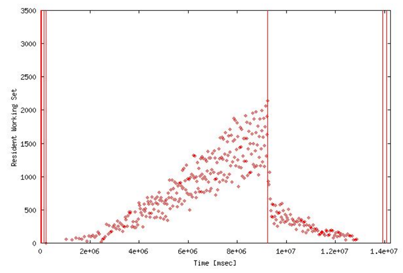
[2] The *hand* model by itself would initially fit within 128 MB, but the operating system also needs memory. Consequently, paging occurs.

Cluster Pagespan: X-axis is scaled wallclock time, Y-axis is pagespan of cluster at front of priority queue

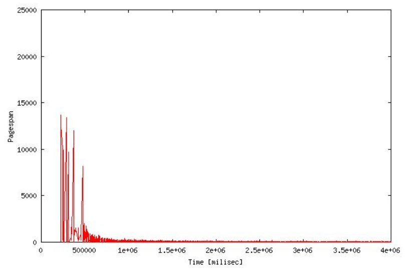
Resident Working Set of Original Model: X-axis is scaled wallclock time, Y-axis is number of clusters in the model with $\geq$ 95% of pages in main memory

(a) Original *R-Simp*, 40,000 polygons in output





(b) *R-Simp* with both Spatial Sorting and Reorganization, 40,000 polygons in output



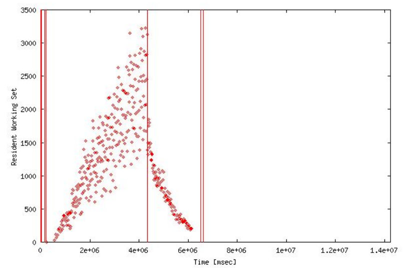


Fig. 3. Cluster Pagespan and Resident Working Set for *blade*

initial clusters are very large, the cluster pagespan is also large at the beginning of the execution. As clusters are split, the cluster pagespan decreases over time. The cluster pagespan data points are joined by lines in order to more clearly visualize the pattern. An instrumented version of *R-Simp* is used to gather the data. Furthermore, for each iteration of the simplification loop, *all* of the clusters in the priority queue are examined. If $\geq$ 95% of the pages containing the vertices and faces of a cluster are in physical memory (as opposed to being swapped out onto the swap disk), that cluster is considered to be resident in main memory. Therefore, Figure 3(a) also shows the *count* of how many of the clusters in the priority queue are considered to be in memory and part of the resident working set, over time.

As the clusters decrease in size, more clusters can simultaneously fit into main memory. Note that the vertices and faces are from the original (not simplified) model. The graph of the resident working set is not monotonically increasing because the operating system periodically (not continuously) reclaims pages. Also, the vertical lines represent important phase transitions. The resident working set count begins to decrease after the simplification phase because the post-simplification and triangulation phases dynamically create new vertex and face lists, which displace from memory the lists from the original model.

The cluster pagespan graph in Figure 3(a) indicates that there is poor memory locality in how the model is accessed for the initial portion of *R-Simp*'s execution. Consequently, the total number of clusters that reside in main memory remains under 2,000 for most of the simplification phase of the algorithm.

We now describe two techniques that measurably improve memory locality, according to cluster pagespan and the resident working set count, and also improve real time performance.

4.2 Off-Line Spatial Sorting

The models used in this case study are stored on disk in the PLY file format [7]. The file consists of a header, a list of the vertices, and a list of the faces. Each polygonal face is a triangle and is defined by a per-face list of three integers, which are index locations of vertices in the vertex list. There is no requirement that the order in which vertices appear in the list corresponds to their spatial locality. Two vertices that are spatial neighbours in the 3D geometry-space can be in contiguous indices in the vertex list, or they can be separated by an arbitrary number of other vertices. There is also no spatial locality constraint on the order of faces in the file. In *R-Simp*, the vertices and faces are stored in main memory in the same order in which they appear in the file, therefore the order of the vertices and faces in the file have a direct impact on the layout of the data structures in memory.

The large cluster pagespan values seen in the early portion of *R-Simp*'s execution (Figure 3(a)) suggests that perhaps the PLY models have not been optimized for spatial locality. Therefore, we decided to spatially sort the PLY file. The model itself is unchanged; it has the same number of vertices and faces at the same locations in geometry-space, but we change the order in which the vertices and faces appear in the file. Our spatial sort reads in the model from the file, sorts the vertices and faces, and then writes the same model back to disk in the PLY format. Therefore, the spatial sort is a preprocessing step that occurs before model simplification. The spatially-sorted version of the PLY file can then be re-used for different runs of the simplification program.

The spatial sort is a recursive Quicksort-like algorithm. After reading the model into memory, a 3D bounding box for the model is computed, as are three orthogonal dividing planes that partition the bounding box into eight isomorphic sub-boxes. Each sub-box is recursively partitioned; the stopping condition for the recursion is when a sub-box contains less than two vertices. As the sort recurses back up, the vertices in the vertex list are reordered so that vertices in the same sub-box at each level of recursion, which are spatial neighbours, have indices that are contiguous.

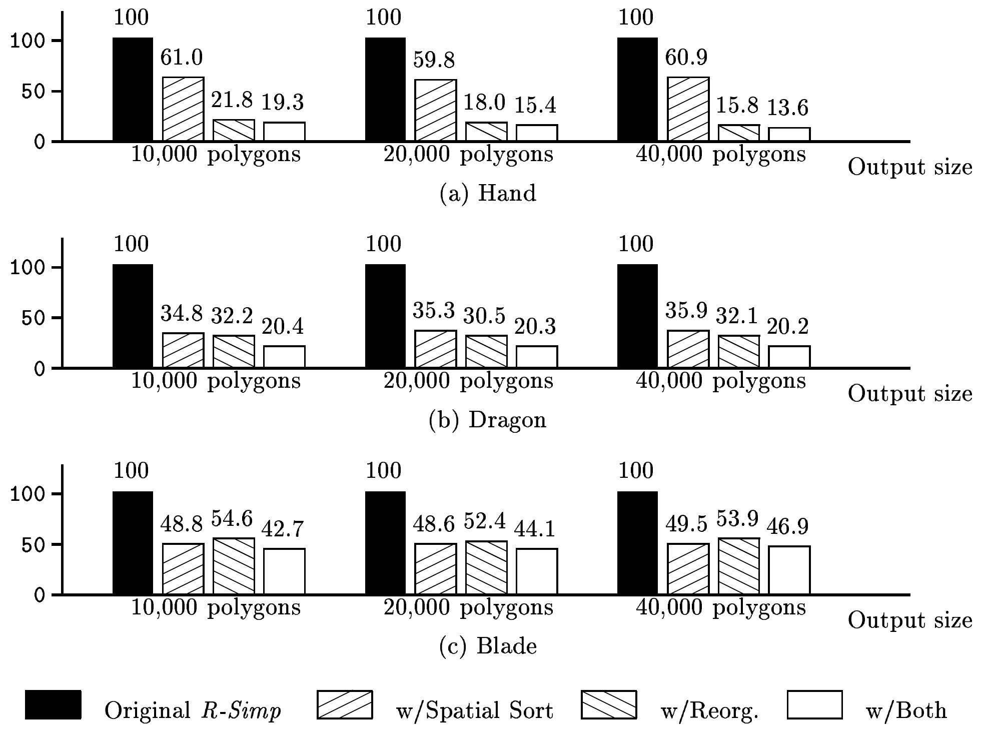


Fig. 4. Normalized Execution Times

We spatially sort the faces according to their vertices to ensure that faces which are neighbours in geometry-space are also neighbours in the face list. Finally, the model is written back to disk in the PLY format with the vertex and face lists in the new, spatially-sorted order.

Our implementation of the spatial sort is written in C++. Spatially sorting the *hand*, *dragon*, and *blade* models require 28, 28, and 89 seconds, respectively, on our 500 MHz Pentium III-based Linux platform. Again, the models only have to be sorted once since the new PLY files are retained on disk.

When *R-Simp* is given a spatially-sorted model for the input, cluster pagespan is reduced throughout the process's execution with a resulting improvement in the resident working set of the original model. For *blade*, there are significant improvements in both cluster pagespan and resident working set, which results in more than a 50% reduction in *R-Simp*'s execution time (Table 2). Spatial sorting also benefits the *hand* and *dragon* models.

Spatial sorting is a simple and fast procedure with substantial performance benefits for *R-Simp*. Although the spatial sort is currently an off-line preprocessing phase from *R-Simp*, we may integrate it into our implementation of *R-Simp* in the future. In the meantime, our experiments and measurements suggest that other researchers should consider spatially sorting the input models for their graphics systems.

4.3 On-Line Reorganization of Data Structures

The performance improvements due to a static spatial sort are substantial. However, as a model is iteratively simplified, there may be an opportunity to dynamically improve memory locality.

We have implemented a version of *R-Simp* that dynamically reorganizes its internal data structures in order to reduce cluster pagespan. Specifically, *before* a sub-cluster or cluster is inserted into the priority queue, the cluster may be selected for *cluster data structure reorganization* (or simply, *reorganization*). If selected for reorganization, the vertices and faces associated with the cluster are copied from the global vertex and face lists into new lists on new pages of virtual memory. The basic idea is similar to compacting memory to reduce fragmentation in memory management. Internal to the cluster data structure, the lists of vertices and faces now refer to the new vertex and face lists, thereby guaranteeing the minimal possible cluster pagespan.

Since there are copying and memory allocation overheads associated with reorganization, it is not done indiscriminately. Two criteria must be met before reorganization is performed:

1. The cluster is about to be inserted into the priority queue within the front 50% of clusters in the queue (i.e., the cluster is in the front half of the queue).
2. The cluster pagespan **after** reorganization must be less than n pages.

The first criteria tries to maximize the chances that a reorganized cluster will be accessed again (i.e., it will reach the front of the priority queue again and be re-used). Reducing the pagespan of a cluster that is not accessed again until the post-simplification phase produces fewer benefits. The value of "50%" was empirically determined.

The second criteria controls at what point reorganization is performed, in terms of cluster size. Reorganizing when clusters are large is expensive and inherently preserves large cluster pagespans. Reorganizing when clusters are small may delay reorganization until most of the simplification phase is completed, thus reducing the chances of benefitting from the reorganization. The specific value of n chosen as the threshold has, so far, been determined experimentally. The optimal value for n is found to be 1,024 pages (i.e., 4 MB given the 4 K pages on our platform) for *blade* (Figure 5). For the three models, the optimal value of n was empirically determined to be between 1,024 and 4,096 pages.

The benefits of reorganization are reflected in the reduced cluster pagespan and increased resident working set (for example, Figure 3(b)), reduced number of page faults (Table 3), and most importantly, in the lower run times (Table 2). When both spatial sorting and reorganization are applied to *R-Simp*, there is an additional benefit (Table 2 and Figure 4). Unfortunately, the two techniques have some overlap in how they improve data-access locality, so the benefits are not completely additive.

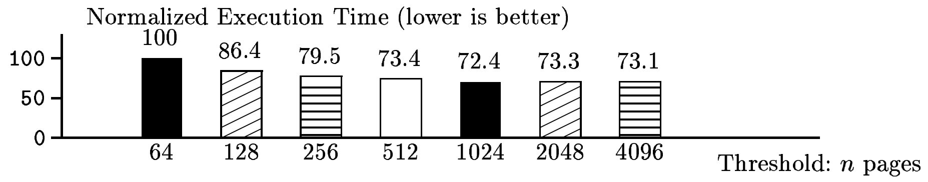


Fig. 5. Varying Reorganization Threshold, *blade*, Normalized Execution Time, 40,000 polygon output

5 Concluding Remarks

In computer graphics and visualization, the complexity of the models and the size of the data sets have been increasing. Modern 3D scanners and rising standards for image quality have fueled a trend towards larger 3D polygonal models and also into model simplification algorithms. *R-Simp* is a new model simplification algorithm with low run times, easy control of the output model size, and good model quality. However, no matter the amount of RAM that one can afford, there may be a model that is too large to fit in memory.

Therefore, we have developed spatial sorting and data structure reorganization techniques to improve the memory locality of *R-Simp* and experimentally shown it to improve performance by up to 7-fold. We have also introduced the cluster pagespan metric as one measure of memory locality in model simplification. For future work, we plan to study if spatial sorting and reorganization can also improve the performance of other simplification algorithms.

References

[1] D. Brodsky and B. Watson. Model simplification through refinement. In *Graphics Interface '00*, pages 221–228. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 2000.

[2] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Proc. ACM SIGGRAPH 1997*, pages 209–216, August 1997.

[3] P.S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University, 1997. Draft Version.

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proc. ACM SIGGRAPH 1993*, volume 27, pages 19–26, August 1993.

[5] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michaelangelo project: 3D scanning of large statues. In *Proc. ACM SIGGRAPH 2000*, pages 131–144, 2000.

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

[7] PLY File Format. `http://www.cc.gatech.edu/projects/large_models/ply.html`.

[8] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, Berlin, 1993. Springer-Verlag.

[9] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, July 1992.