R-Simp to PR-Simp: Parallelizing A Model Simplification Algorithm

Dmitry Brodsky
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
dima@cs.ubc.ca

Abstract

As modelling and visualization applications proliferate, there arises a need to simplify large polygonal models at interactive rates. Unfortunately existing polygon mesh simplification algorithms are not well suited for this task because they are either too slow (requiring pre-computation) or produce models that are too poor in quality.

Given a multi-processor environment a common approach for obtaining the required performance is to parallelize the algorithm. Many non-trivial issues need to be taken into account when parallelizing a sequential algorithm. We present *PR-Simp* a parallel model simplification algorithm and the issues involved in parallelizing *R-Simp*.

1 Introduction

Many of today's graphics applications require simplification of polygonal models to be performed at interactive rates. Given the proliferation of multi-processor systems a common approach for obtaining the required speedup is to parallelize the sequential algorithm. Unfortunately most simplification algorithms are difficult to parallelize due to several issues. We present these issues as they relate to conventional simplification algorithms and to R-Simp. We then present solutions to these issues in the form of PR-Simp a parallelized version of R-Simp. We chose to parallelize R-Simp because its approach to simplification makes it easier to parallelize.

1.1 Background

The original sequential algorithm, R-Simp [1], was inspired by splitting algorithms from the vector quantization literature [4]. The algorithm simplifies in reverse from coarse to fine, allowing us to guarantee a displayable result within a specified time limit. At every iteration of the algorithm, the number of vertices in the simplified model is known, enabling control of output model size. We use curvature to guide the simplification process, permitting preservation of important model features, and thus a reasonable level of output model quality.

The multi-processor environment consisted of several uni-processor systems interconnected by a highspeed network. The system contained no shared memory and thus all interprocess communication was done using explicit message passing. The parallelization issues presented here are with respect to this configuration.

1.2 Issues

Several issues need to be considered when creating a parallel simplification algorithm. The first issue, and probably the most important, is the ability to partition the data and to subdivide the simplification task into several smaller simplification tasks. Most conventional simplification algorithms simplify by removing primitives (vertices, edges, polygons) from the original model. The primitives are removed sequentially such that each removal causes the least amount of distortion to the original surface[3, 5, 6, 8]. When a primitive is removed it causes the surface to deform in its local area. This deformation has to be taken into account for the next removal. Thus, it is difficult to partition the surface such that each portion can be simplified on its own because a simplification in one part of the surface can influence the surface in a different part. This is less of an issue for R-Simp due to its divide and conquer approach; once a surface is partitioned a modification in one partition will not spill into other partitions.

The second issue is the minimization of communication. The amount of communication is always an issue since it contributes significantly to the overhead of a parallel program. It is even more of an issue for model simplification algorithms because the datasets are usually extremely large whereas the computations themselves are relatively simple. A solution where a single administrator process distributes work to set of worker processes will not perform well because the communication overhead can easily exceed the amount of work performed by a worker process.

Simplification steps are implicitly ordered in conventional simplification algorithms due to their sequential nature. The ordering will always result in the final simplification being optimal. In a parallel algorithm the implicit ordering is lost since tasks that happened in sequence are now performed in parallel. Thus, because the ordering is lost the result is not guaranteed to be optimal. Most algorithms also need to have a global view of the entire dataset. This requirement forces the worker processes, in a parallel algorithm, to communicate their changes to the other worker processes. Both of these two sub-issues could be dealt with, to some degree, by having the worker processes perform an all-to-all broadcast after each iteration, but that would add considerable overhead. R-Simp operates on clusters and a change in one cluster does not affect any other clusters thus latter sub-issue is handled implicitly. R-Simp operates on the principle of refining portions of the model that need refining. The order of these refinements is not important. Thus the former sub-issue is partially handled implicitly, but the results are still sub-optimal. More details will be given in Sections 2.2 and 2.3.

The last issue is memory efficiency. One of the many reasons to parallelize a simplification algorithm is to be able to simplify extremely large models. Ideally if there are p processors then the amount of memory p_i should use should be $\frac{M}{p}$, where M is the memory footprint of the sequential algorithm. In practice this is very hard to achieve because there are bookkeeping structures that are needed and thus there is always some duplication. Conventional algorithms need to keep track of all the modifications done to the dataset, thus they must store the entire model in memory. Each worker process in the parallel algorithm has to store a copy of the entire model. Hence the parallel algorithm would use p times as much memory. This is not the case for the parallel version of R-Simp.

These are the basic issues that must be addressed when designing a parallel simplification algorithm. In the next section, section 2, we describe in detail how the above issues are solved. In section 3 we show the speedup obtained by parallelizing R-Simp. And finally in section 4 we summarize the work and draw conclusions about the experiences obtained parallelizing R-Simp.

2 Description

The three issues presented above were used as the guiding forces for the design and implementation of PR-Simp. We will discuss the solutions to these three issues. Specific details about the actual simplification algorithm will not be presented because they are not relevant to the parallelization of R-Simp. You can find the details in [1].

2.1 Overview

PR-Simp was designed to run on 2, 4, or 8 processors. These numbers were chosen to fit with the partitioning scheme of R-Simp. With a few minor modifications PR-Simp could run on any number of processors. Figure 1 shows the general flow of PR-Simp in a four processor configuration. Processor N_0 is the administrator

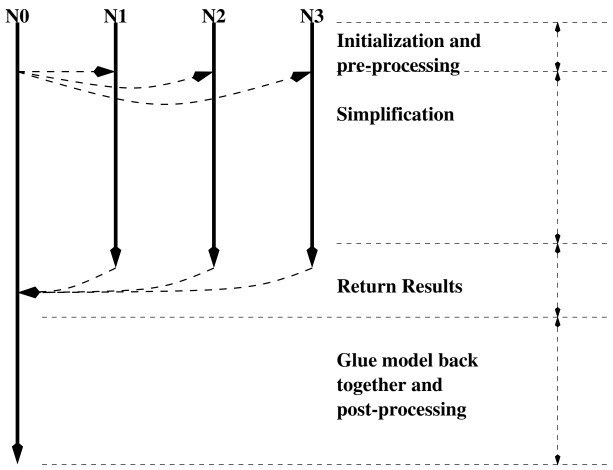


Figure 1: The flow of PR-Simp on four processors.

process. It has special duties during the initialization and post-processing stages. Processes $N_1 ... N_{n-1}$ are all identical and can be viewed as worker processes. There are four main stages in PR-Simp. The first stage loads the original model and initializes the required data structures. The details of this stage are described in the Data Partitioning section (Section 2.2). At the end of the first stage the administrator process broadcasts the total surface area of the model to the worker processes because they require it for the simplification stage. Simplification occurs in the second stage. The standard R-Simp simplification algorithm is run at each node, including that of the administrator. Most of the communication occurs in the third stage. In the third stage all the worker processes package up their data and send it to the administrator. They do not send the new polygonalized surface, rather they send the necessary information for the administrator process to reconstruct the surface. Once the administrator has received all the data from all the workers it reconstructs the surface in the same way R-Simp reconstructs the surface. This is performed in the last stage. The overall algorithm has remained unchanged in terms of the simplification process. The modifications made to R-Simp to make it parallel are discussed in the next three sections.

2.2 Data Partitioning

The first step in parallelizing R-Simp is to partition the data and the work to the worker processes. Since the initial data sets are usually immense the administrator process cannot simply partition the data and send it to the workers. Instead we opted for having each worker process read the portion of the model it was responsible for. This simplified the algorithm considerably and removed the tremendous overhead that would have been incurred if the data was sent to the worker processes. An unfortunate side effect of this decision is that there is a requirement for the existence of a network file system that provides access to the model file to all clients. Given that the version of MPI used, LAM, used the same service the design decision seemed appropriate. We determine the portion of the model each worker is responsible for by using the worker's rank.

Initially and for simplicity we assumed that PR-Simp will be run on a 2, 4, or 8 processor systems. This assumption was made because initially R-Simp subdivides the original model into eight clusters using three axis aligned planes located in the centre of the model's bounding box and these eight clusters could easily and evenly be distributed to 2, 4, or 8 processors. With a few minor modifications PR-Simp could use any number of processors. To determine which clusters belong to which processors we used the following two formulas:

$$start_i = \frac{8}{Size} * Rank_i \qquad (1)$$

$$end_i = start_i + \frac{8}{Size} \qquad (2)$$

where *Size* is the number of nodes in the processor set and $Rank_i$ is the rank of a processor P_i in the set of processors. If we assume that the initial eight clusters are numbered from zero to seven then the values *start* and *end* provide the range of clusters a worker process is responsible for.

Next we partition the work. Naively one can partition the work by dividing by the number of processors. That is, if we want a model of size M then we have each process simplify its portion to a size of $\frac{M}{Size}$. This approach does not work because the clusters do not contain equal number of vertices. Thus the amount of work done has to be scaled by the number of vertices in each cluster, that is:

$$\frac{S}{F} = \frac{S_{P_i}}{F_{P_i}} \qquad (3)$$

where S is the size of the full simplified model, F is the size of the original model, F_{P_i} is the number of vertices given to processor P_i to process, and S_{P_i} is the target number of vertices for processor P_i. Thus, the target number of vertices for processor P_i is:

$$S_{P_i} = \frac{S}{F} * F_{P_i} \qquad (4)$$

All the necessary information is available locally for this computation since it can be obtained from reading in the model.

Most of this work occurs in the pre-processing stage (see Figure 1). There is a small broadcast done by the administrator. This broadcast transmits the total face area of the model. Since the workers only process a portion of the original model, they are not able to compute the total area, that they need for the simplification stage. The administrator processes all the faces because they are needed for the post-processing stage and thus it is able to compute the required value. Once the pre-processing stage is complete each process simplifies its portion of the model using the sequential algorithm.

2.3 Reducing Communication

The next step in parallelizing R-Simp is to minimize the amount of communication. Initially, the administrator/worker model used, had the administrator send individual clusters to worker processes to be partitioned and sent back. This resulted in a slowdown and preliminary timings showed that the communication was the bottleneck. Thus it was necessary to minimize the amount of communication to achieve the desired speedup.

To reduce the amount of communication the model in Figure 1 was chosen. There are only two synchronization/communication points. There is a broadcast in stage one, and in the third stage all the worker processes send their results to the administrator. The broadcast in stage one involves the administrator broadcasting a single double to all the worker processes. We also performed our own marshalling rather then using the support provided by `MPI_Pack` and `MPI_Unpack` because the marshalling procedures in MPI are slow due to the amount of copying involved.

PR-Simp's main data-structure is shown in Figure 2. The result of the simplification stage is an array of these structures. Each worker process sends this array to

```
typedef struct node {
  double        vertex[3];
  double        normal[3];
  double        centroid;
  unsigned int  *vertlist;
  unsigned int  vertnum;
  double        patcharea;
  unsigned char topo;
} Node;
```

Figure 2: The Node data-structure.

the administrator. To avoid packing the data into one buffer the data is sent as multiple messages with special tags. The data is sent in the following sequence:

1. A message containing n, the number of elements in the array, an unsigned long.
2. Then $2S_{P_i}$ messages are sent:
   (a) A message containing an array of 11 doubles to hold the information contained in the `Node` structure.
   (b) A message containing the `vertlist`, an array unsigned ints.

This translates to approximately $2S$ messages being sent. If the data was packed then only $Size - 1$ messages would be sent. Packing adds a tremendous amount of overhead as is evident by the fact that it is faster to send $2S$ simple messages rather than $Size - 1$ packed messages. The total number of messages being sent is $2(Size - 1 + S)$. By reducing the communication the desired speedup was achieved.

2.4 Reducing the Memory Footprint

Finally, PR-Simp's memory footprint needed to be minimized. By minimizing the memory footprint we were able to achieve additional speedup due to fewer memory allocation calls and we were also able to simplify larger models, which was our secondary goal.

The memory footprint was reduced by only retaining data that is needed. Initially all the vertices are read as in R-Simp. As the vertices are read the mean vertex for the model is computed by:

$$mv = \frac{1}{N}\sum_{i}^{N} \vec{v}_i \qquad (5)$$

Next, the faces that are needed by worker processes P_i are read and the vertices that are no longer needed are released. Although this procedure increases the number of `free()` calls the memory footprint is reduced to approximately 30% of the original. The administrator does not delete any vertices and reads all the faces because it uses them during the post-processing stage. The memory footprint is still reduced by initializing only those vertices and faces used in the simplification stage. The resulting footprint is 50% smaller than the original.

3 Results

Simplification algorithms are usually judged by two criteria. The first criterion is speed, the time required to simplify a model. The second and more difficult to measure criterion is quality. Intuitively speaking, quality of a simplification is its appearance or its geometric accuracy. We also look at the speedup attained from parallelizing R-Simp and the efficiency of the parallel algorithm.

3.1 Performance

We first consider the speedup obtained by parallelizing R-Simp and the efficiency of PR-Simp, the parallel algorithm. Speedup is the ratio of the execution time for the best known sequential algorithm and the parallel algorithm. The simplification time was measured over a wide range of inputs (see Figure 3) and so the average speedup was computed. The average speedup was computed by finding the mean of the speedup at each input point:

$$S(n,p) = \frac{T^*(n)}{T(n,p)} = \frac{1}{N}\sum_{i}^{N}\frac{T^*(n_i)}{T(n_i,p)} = 1.82 \qquad (6)$$

Where N is the total number of trials and n_i is a trial with a specific output size. The speedup obtained is approximately a factor of 2.0; that is significant given the performance of R-Simp. Figure 3 shows the performance of several simplification algorithms. Graphs 1, 2, and 3 show the performance of R-Simp [1], QSlim [3], and a simplified vertex clustering algorithm developed by [7]. The clustering algorithm (graph 3) is the fastest of all the algorithms, but it produces the worst results. These three algorithms were run on an SGI *Onyx*^{2} Infinite Reality Engine with 195 megahertz CPUs and 512 megabytes of memory. Algorithms 4 and 5 were run on a Pentium III running at 450 megahertz and 128 megabytes of memory. The forth algorithm is the regular version of R-Simp. This performance run was done to show the relationship of R-Simp running on the Onyx and on the Pentium III. R-Simp on the Pentium III is about 1.05 times slower than the Onyx version. Graph 5 is that of PR-Simp the parallel algorithm running on

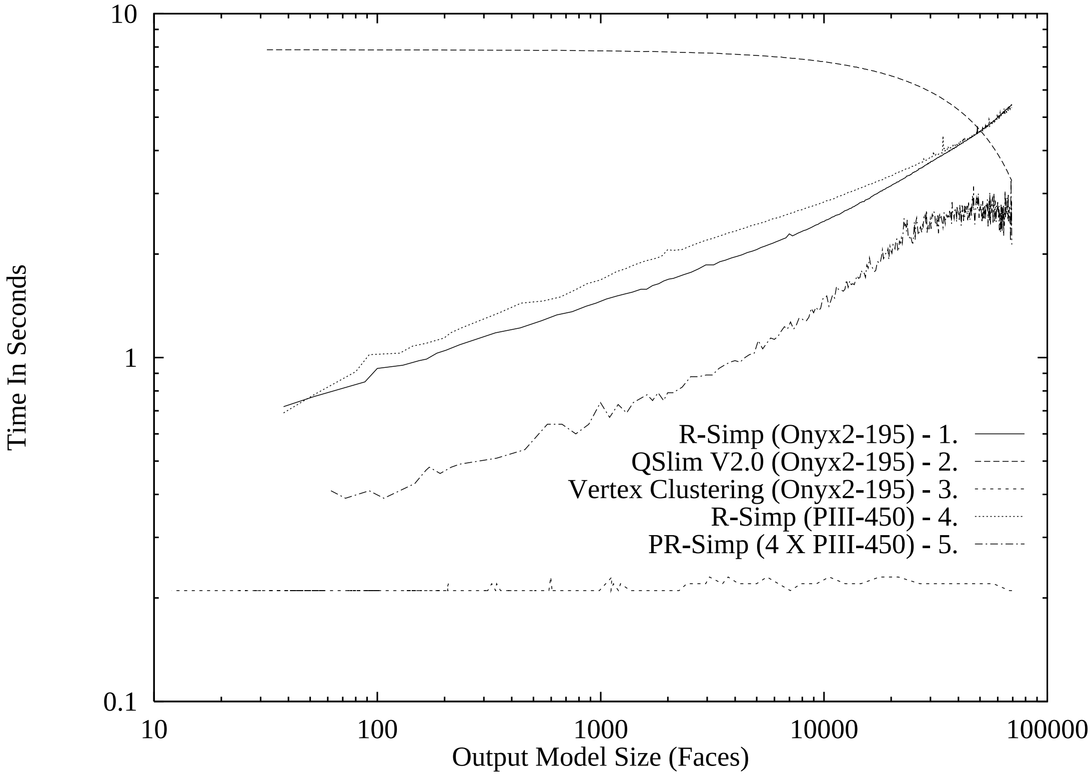


Figure 3: Performance of PR-Simp as compared to R-Simp. The effect of output model size on simplification time for the Stanford Bunny.

four Pentium IIIs. Figure 3 shows that R-Simp is significantly faster than QSlim; one of the fastest known vertex merge algorithms. PR-Simp is twice as fast as R-Simp for models of 10000 polygons or less. Given the amount of data that is being processed a larger speedup would be difficult to obtain.

The efficiency of this algorithm is about 50%, which is average for the majority of parallel algorithms. The efficiency is computed by taking the speedup and dividing it by the number of processors used:

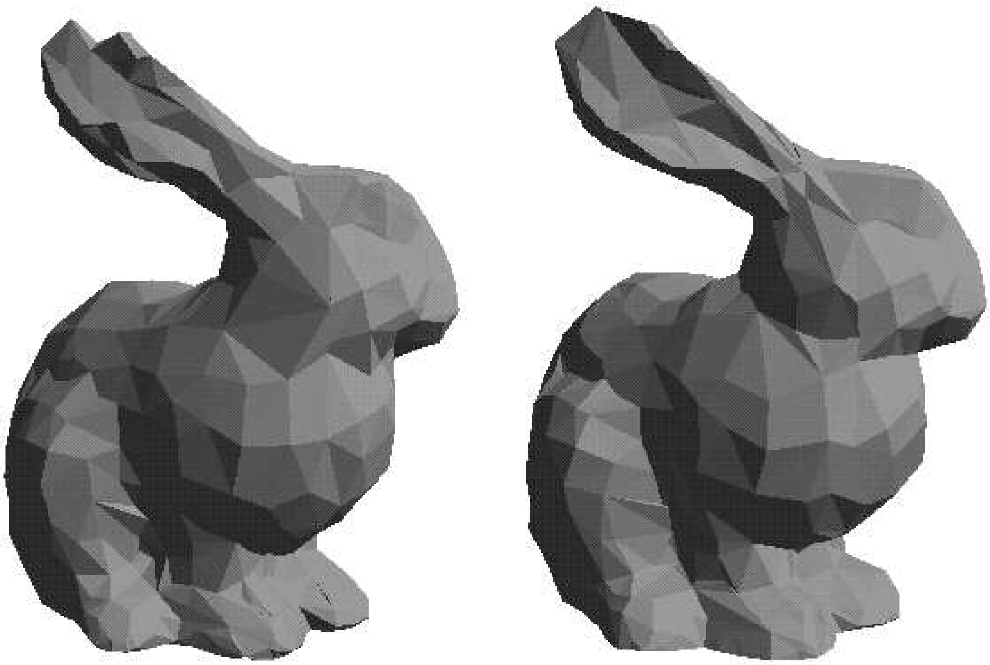
$$E(n,p) = \frac{S(n,p)}{p} = \frac{1.82}{4} = 45.5\% \tag{7}$$

Ideally the efficiency should be 1.0, our efficiency was approximately 50% with an attained speedup of 2.0 on a system with four processors.

Comparing PR-Simp to QSlim it is evident that PR-Simp is considerably faster. QSlim is faster than R-Simp when fine output models are required (see Figure 3). PR-Simp on the other hand is faster than QSlim for all output model sizes. Finally, visually comparing the results of PR-Simp to R-Simp and the other simplification algorithms (see Figures 4 and 5) it evident that there is some loss to quality. This is especially noticeable on the bunnies ears in Figure 4.

3.2 Discussion

A noticeable side effects of parallelization is the degradation in output model quality. The decline in quality is due to the loss of ordering to the simplification operations provided implicitly by the sequential algorithm. The R-Simp algorithm needs to expand the S clusters that contain the most curvature, the order of expansions does not matter. Parallelization destroys the implicit ordering and so the S cluster that need expanding are not expanded. A similar example, given a set A it is straight forward to remove the n largest elements, call this set A_l. Now if set A is subdivided into m sets, $A = \{A_0, A_1, \ldots A_{m-1}\}$, and we remove from each set A_i, for all i, the largest $\frac{n}{m}$ elements and stick them into the set A_{sl}. The sets A_i and A_{sl} will have the same number of elements but not necessarily the same elements. This exact situation occurs here. On average, each process expands $\frac{S}{Size}$ clusters. The resulting set of clusters, $Size \times \frac{S}{Size}$, is not necessarily the S clusters that needed to be expanded. This effect can be seen in Figure 4. In Figure 4a we see that the ear and the crease in the leg are not as well defined as in Figure 4b. This is due to the fact that the process that contained the ear and/or the leg was not given a sufficient vertex



(a) Parallel Algorithm (b) Sequential Algorithm

Figure 4: The effects of parallelization on quality.

bound to expand the necessary clusters while the process(s) that contained flatter areas probably expanded them too much.

PR-Simp was implemented on a system where shared memory was not available. Without shared memory all the communication had to be done through proper communication channels that contributed considerable overhead. To reduce this overhead the communication was minimized but in return output model quality was lost. If shared memory was available a single priority queue could be used to store the expansion results. Returning to a single priority queue would return a large portion of the ordering and therefore output model quality. This modification would not bring back full ordering because clusters would still be expanded in parallel. Shared memory could be instantiated using message passing by performing all-to-all broadcasts after every iteration of the algorithm. Unfortunately, this approach counters the effort to minimize communication and thus is not feasible. It would be more efficient to go with the pure administrator/worker design. Thus, shared memory would be advantageous for the implementation of PR-Simp if the processors and the shared memory were local to one another, that is a multi-processor machine, or the implementation of global shared memory (GSM) was considerably faster then the explicit message passing.

The necessity for the ordering only occurs when the vertex bound is used as the stopping criterion. There are other stopping criteria that that do not require the cluster expansions to be ordered. One such stopping criterion is the error bound where the error is the distance between the original and the simplified surface. No ordering to cluster expansion is needed because clusters are split until the simplified surface is within a specified distance from the original surface. The present implementation of PR-Simp is well suited for this stopping criterion and would produce optimal simplifications.

From the problem of optimality arises the problem of scalability. With one processor the simplifications are optimal because there is an implicit order imposed on the cluster expansion operations. As the number of processors increase the ordering decreases because the ordering is only imposed within groups of $\frac{S}{P_n}$, where P_n is the number of processors. Hence the more processors the smaller the groups and therefore there is less overall ordering. Thus as the number of processors increase the optimality of the simplified model will decrease. A benefit of increasing the number of processors is the ability to simplify extremely large models. The bottleneck for most simplification algorithms is the amount of memory. Since each processor has its own memory then the more processors there are the more memory there is and the size of the input model that can be simplified increases.

The computed speedup is 1.82, examining graphs 4 and 5 in Figure 3 we notice that for output model sizes of 0 to 20000 polygons the speedup is larger than 1.82. We also notice that the curve in graph 5 is considerably steeper than that of graph 4. The reason is that as the output models become finer more data needs to be sent from the worker process to the administrator process which adds communication overhead. Thus the steeper curve is due to communication overhead. The speedup is computed for the entire range of output model sizes, from 6 polygons to the full model, if we only looked at the range from 0 to 20000 polygons we would obtain a greater speedup and the efficiency would also improve.

4 Conclusions

There are three major issues that needed to be addressed for the parallelization of R-Simp. The first issue dealt with the partitioning of the data and the work between the nodes. This was accomplished during the startup of each node using the rank of the node and the size of the processor set. No communication was necessary to perform this task. The second issue dealt with reducing the overall communication. This is important because with large datasets and simple computations the communication overhead can easily surpass the cost of the overall algorithm. A modified administrator/worker architecture was chosen where communication only occurred at the very end of the algorithm when it was necessary to glue the simplified model together; the administrator performed that job. We also performed our own marshalling to avoid message packing and the associated expense of copying. Adopting the modified administrator/worker architecture and doing our own marshalling provided a speedup of approximately 2.0. The final issue was memory efficiency. This was important because the goal was not only to simplify quickly but to be able to simplify extremely large models. The result is PR-Simp a parallel version of R-Simp that is approximately two times faster and produces similar quality output models.

References

[1] Dmitry Brodsky. R-simp: Model simplification in reverse, a vector quantization approach. Master's thesis, University of Alberta, 1999. http://www.cs.ubc.ca/ dima/research.html.

[2] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. Technical report, Istituto per l'Elaborazione dell'Infomazione - Consiglio Nazionale delle Ricerche, 1997.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997.

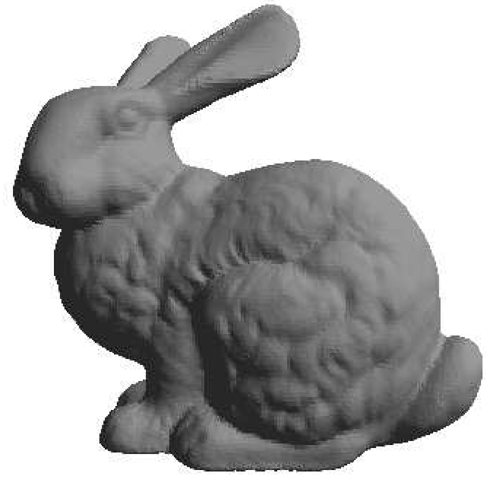
[4] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[5] Hugues Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

[6] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *Proceedings IEEE Visualization'98*, pages 279–286. ACM, IEEE Computer Society Press, 1998.

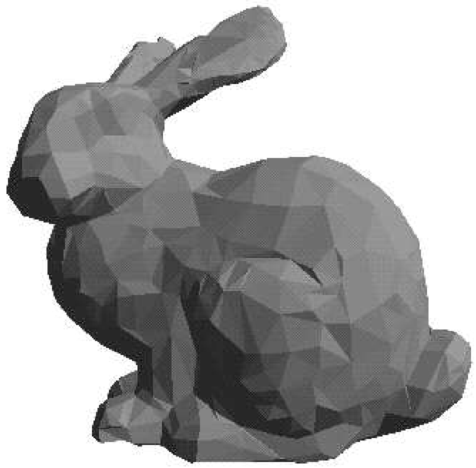
[7] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, Berlin, 1993. Springer-Verlag.

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



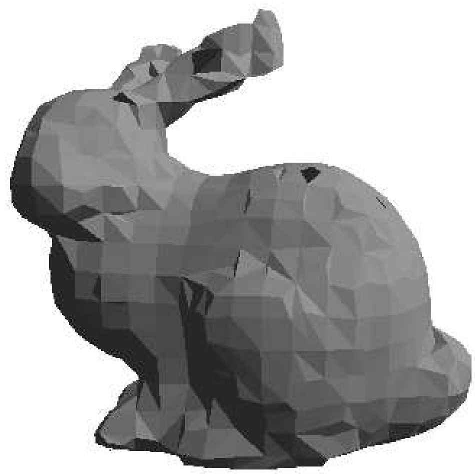
(a) Original

(b) R-Simp
Error 0.155%



(c) PR-Simp
Error 0.284%

(d) QSlim
Error 0.071%



(e) Vertex Clustering
Error 0.302%

Figure 5: Visual results of five simplification algorithms and Metro's [2] mean total error measure.