Enhancing Digital Documents by including 3D-Models

Swen Campagna Leif Kobbelt Hans-Peter Seidel

Computer Graphics Group
University Erlangen-Nuremberg, Germany *

Abstract

Due to their simplicity, triangle meshes are used to represent geometric objects in many applications. Since the number of triangles often goes beyond the capabilities of computer graphics hardware and the transmission time of such data is often inappropriately high, a large variety of mesh simplification algorithms has been proposed in the last years. In this paper we identify major requirements for the practical usability of general purpose mesh reduction algorithms to enable the integration of triangle meshes into digital documents. The driving idea is to understand mesh reduction algorithms as a software extension to make more complex meshes accessible with limited hardware resources (regarding both transmission and display). We show how these requirements can be efficiently satisfied and discuss implementation aspects in detail. We present a mesh decimation scheme that fulfills these design goals and which has already been evaluated by several users from different application areas. We apply this algorithm to typical mesh data sets to demonstrate its performance.

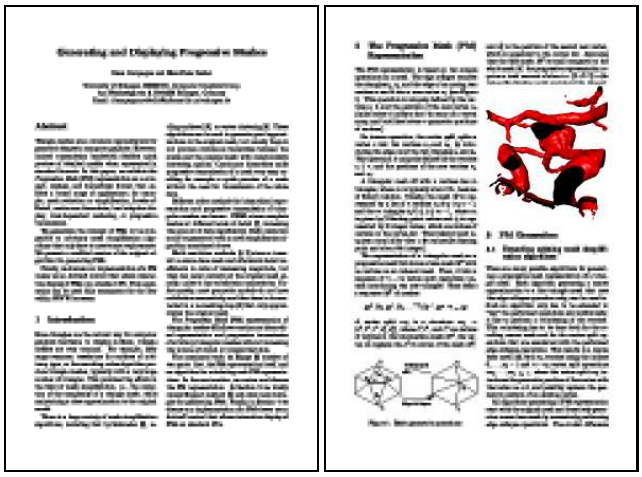


Figure 1: Example of a digital document: a 3D iso-surface reconstructed from a CT-scan (red) is included into the digital document to provide additional information.

1 Introduction

Just like multi-media data, the use of 3D objects enriches traditional digital documents like text documents, web-pages, online books, or encyclopedia on CD-ROM. The possibility to add 3D data enables interaction and provides a maximum amount of information.

Digital 3D documents do not only extend existing applications, but they also provide completely new possibilities. A very large community may visit a virtual electronic museum over the internet, regardless where on the world it is located. In tomorrow's museum catalogue, Michelangelo's fresco-paintings could come together with a whole virtual chapel. A customer may compose, arbitrarily often change, and inspect furniture arrangements in a digital warehouse catalogue and by pressing a button he would get a detailed part list together with the total price. Or an electronic clinical record could contain not only textual descriptions and pictures, but also, e.g, an iso-surface reconstructed from a CT-scan (cf. Fig. 1).

Many applications process or generate 3D surface geometry. In order to make such data persistent, either by storing it to a backup device or by transmitting it via the internet to other computer systems, a certain representation format has to be chosen. Storing or transmitting surface geometry data means to find a mapping from the geometry (e.g. points) and topology (e.g. triangles) to the linear structure of random access memory, harddisks, or ethernet devices.

There exist several surface representation formats. Since a 3D object can be of arbitrary complexity, it is usually assembled by several parts or so-called patches. CAD-software uses NURBS (Non-Uniform Rational B-Splines) to describe fairly complex patches. Using simpler types of patches naturally requires a larger number of them to describe an object of similar complexity.

The simplest patch at all is a triangle. There are several reasons, why triangle meshes have become a common way to represent surfaces: graphics hardware accelerates the display of triangle meshes (in fact, today any other surface representation format is eventually converted to triangles meshes for display). Furthermore, algorithms for triangle meshes are more easy to implement and more robust than for higher order patches. Thus, triangle meshes have emerged as a de-facto standard for representing 3D geometry in computer graphics.

Since the complexity of triangle meshes grows faster than the capabilities of computer graphics hardware and transmission bandwidths, there is much ongoing effort in the area of mesh simplification. Many algorithms have been proposed for the decimation of triangle meshes. [19, 20, 17] give an overview over the large variety of techniques. Recently, an attempt was made to investigate the generic nature of incremental mesh decimation algorithms [13]. Even more important, incremental mesh simplification algorithms can be used to generate hierarchical triangle mesh representations that are essential when using triangle meshes in digital documents (cf. next section).

After considering the use of triangle meshes in digital libraries, in the following sections we discuss in detail, how the practical usability of a simplification algorithm can be achieved. We present the necessary components that have to be considered and combine these partial solutions to a mesh simplification algorithm. We verify the achievement of the design goal by applying our implementation to data sets from different application areas.

*Computer Sciences Department (IMMD9), University of Erlangen-Nürnberg, Am Weichselgarten 9, 91058 Erlangen, Germany, Email: {campagna|kobbelt|seidel}@informatik.uni-erlangen.de

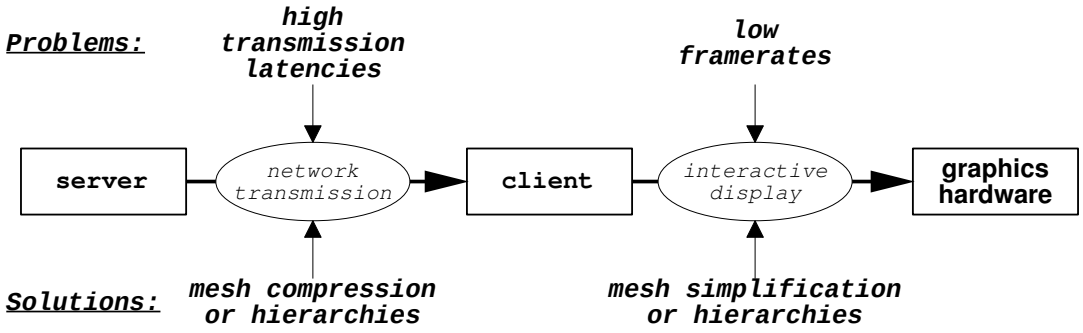


Figure 2: The two bottlenecks and possible solutions when using complex triangle meshes in client/server systems for interactive display.

2 Client/Server Systems

Although interactive display of 3D surfaces is possible on stand-alone graphics workstations, major gains in functionality in the context of digital libraries are acquired if 3D documents are made immediately available to a large number of users in a world-wide distributed system like the internet. Thus, the data should be located on server systems and accessed by client computers. This leads to two major bottlenecks (cf. Fig. 2):

- *Transmission bandwidths* are limited. The user has to wait until his client computer received the whole data set and then finally displays the object.
- Different client systems typically have a very broad range of *graphics performance*. A standard PC may only display objects with a few hundred triangles interactively, while a graphics workstation can process several thousand triangles at interactive frame rates. This effect is independent from the surface representation formats.

These two bottlenecks clearly show the main drawback of triangle meshes: a large number of triangles is required to describe a complex 3D object. Mesh compression techniques [23, 8] only help to reduce the transmission latency which is still depending on the size of the whole triangle mesh. Fortunately, hierarchical reorganization of the triangle mesh representation helps to avoid both bottlenecks at the same time. Furthermore, the advantages of a hierarchical representation are of practical use even for stand-alone computer systems.

3 Mesh Hierarchies

The upper row of Figure 3 shows the traditional sequential transmission and display of a 2d picture. In contrast, the lower row of Figure 3 exploits the advantages of a hierarchical image format like progressive GIF or JPEG: already after receiving only a few percent of the total data, one can imagine how the final image would look like.

Hierarchical representation of triangles meshes has the same motivation and advantages as it is well known for the mentioned hierarchical image formats. The upper row of Figure 4 shows the sequential, and the lower row the hierarchical transmission and display of a 3D object. The benefits of the hierarchical representation are clearly visible and obviously help to avoid the bottleneck of limited transmission bandwidths.

The performance of computer systems is increasing steadily, considering CPU speed, the size of main memory, the triangle throughput of computer graphics hardware, and transmission bandwidths. Looking at contemporary systems, PCs or graphics workstations, we can observe that all of them are able to store triangle meshes into their main memory which they are not able to display with acceptable frame rates. It makes sense to extrapolate this observation into the future, i.e., the ratio of main memory size to graphics performance will not change significantly.

Since computers can easily handle larger meshes than they can actually display, mesh reduction algorithms or pre-computed mesh hierarchies (from which a reduced mesh can be extracted) are required to bridge the gap. This can be considered as a pipelining process:

Hence, *the maximum size of a triangle mesh that may be displayed on a specific computer is obviously limited by the number of triangles that may be processed by a mesh simplification algorithm on the same system*. Thus, hierarchical triangle meshes also help to avoid the second bottleneck: interactive display.

Several hierarchical triangle mesh representation formats are already available. The Virtual Reality Modeling Language (VRML) [4] has been defined for the transmission of 3D geometries over the internet. An object may be represented hierarchically by using a level-of-detail (LOD) node in the scene description. This LOD node uses several distinct versions of the object at various resolutions. Thus, the total amount of memory requirement increases with the number of levels.

Hierarchical triangle meshes based on wavelets methods [5, 15] require a certain structure of the mesh — the so-called subdivision connectivity. Since only few models have this special connectivity a priori, a conversion has to be performed in a preprocessing step which re-samples the original object [5, 14].

Another method for the hierarchical representation of a triangle mesh is to first transmit a coarse approximation and then detail information is added successively to the current mesh. This defines a very smooth sequence of finer and finer triangle meshes. The most popular technique that works in this way is the *progressive mesh* (PM) representation [10]. The benefits of PMs are that they do not require more memory than the mesh at its highest resolution [3] and that they provide a smooth and lossless hierarchy of triangle meshes.

We have implemented an ActiveX-control for Microsoft Windows that allows to progressively receive such PM-records (cf. Figure 5 on the left) [2]. Another implementation is a viewer for Motif and OpenGL that may be used as a helper application for web browsers (cf. Figure 5 on the right) [2]. This PM-viewer also allows the load-adaptive display of a progressive mesh, i.e., it automatically adapts the complexity of the shown mesh to the capabilities of the used graphics hardware and the current processor load.

To convert an arbitrary triangle mesh to a PM, an incremental mesh decimation algorithm is required that iteratively performs

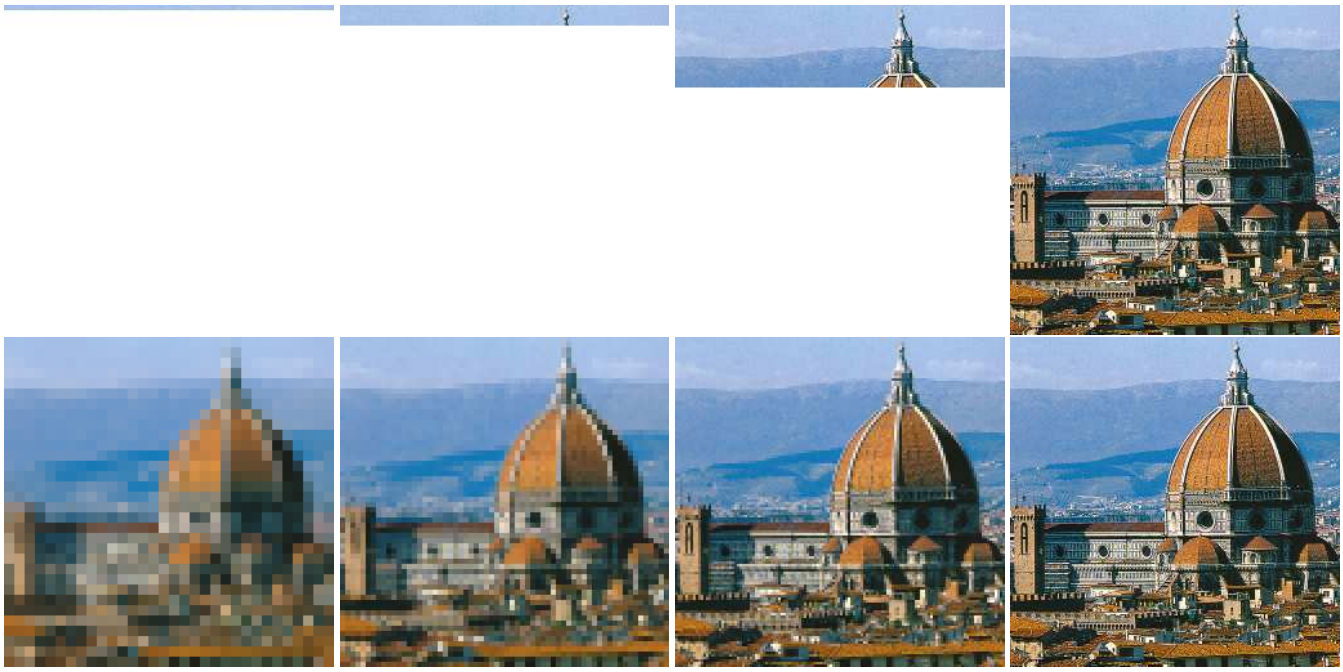


Figure 3: Upper row: sequential transmission of an image. Lower row: progressive transmission using a hierarchical image file format. The two images in each column have been generated using the same amount of transmitted data.

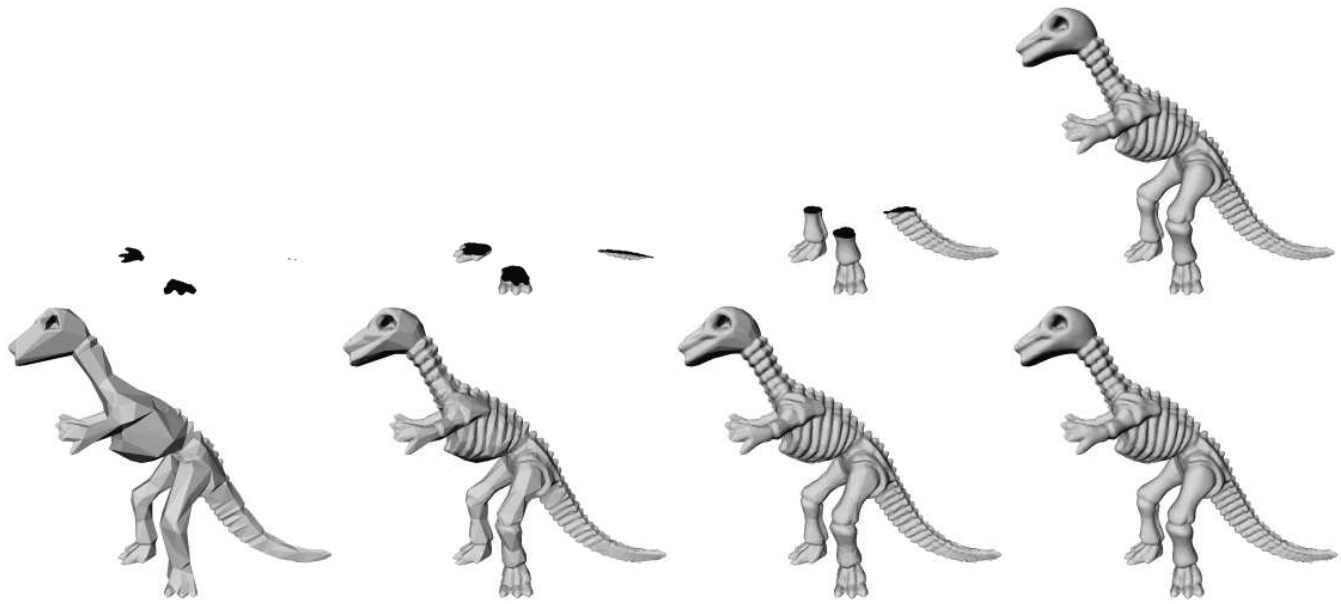


Figure 4: Upper row: sequential transmission of a 3D object. Lower row: progressive transmission using progressive meshes. The two models in each column use the same number of triangles (from left to right: 1.759, 7.038, 28.155, 112.623 triangles).

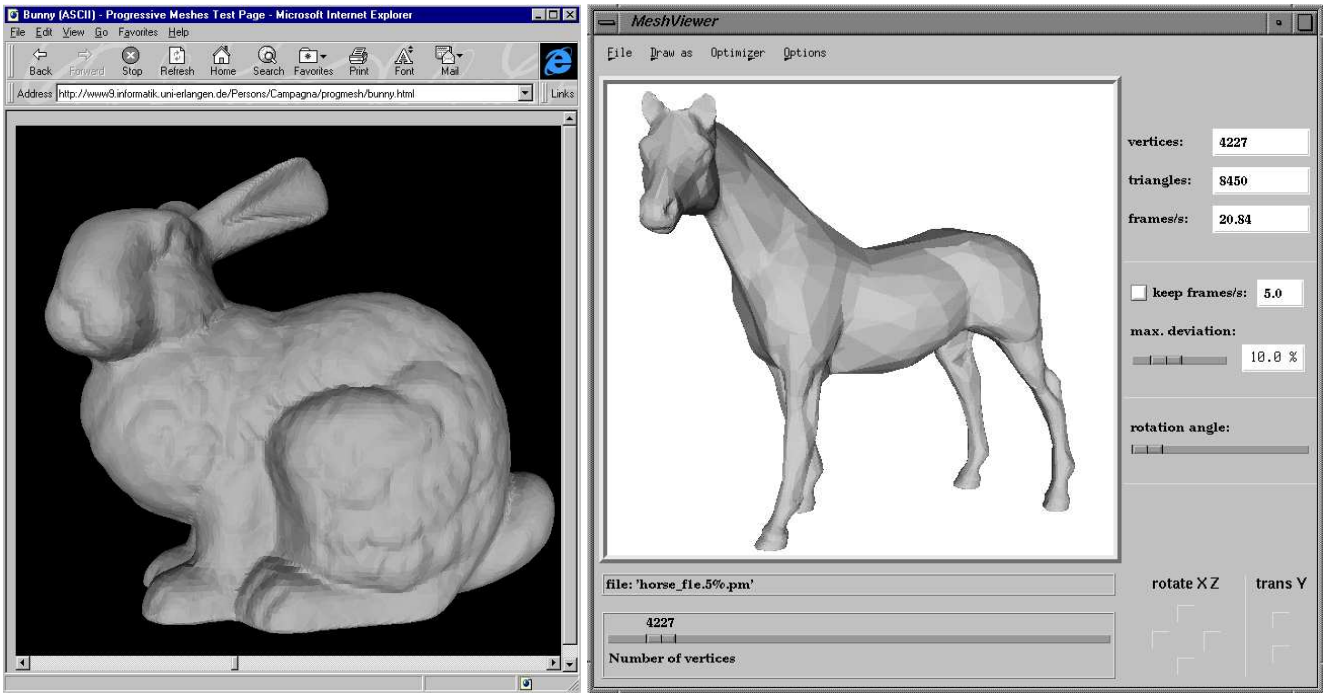


Figure 5: Interactive display of progressively received triangle meshes. Left: ActiveX control for Microsoft Windows. Right: web-browser helper application for Motif and OpenGL that also allows the load-adaptive display of the hierarchy.

edge collapse operations. The mesh simplification algorithm tries to find a base mesh of a PM that represents the global shape of the object and that is encoded in a traditional way (e.g. by shared vertices). The remainder of a PM file encodes the detail information, i.e., a large sequence of vertex splits that reverse the operations performed during mesh decimation. Transmission of a PM file may be interrupted when desired (resulting in an approximation of the object).

4 Mesh Decimation

Up to now, research in the field of mesh simplification has focused on the development of new decimation algorithms and it seems to have gained a status of maturity. For every sort of application a suitable simplification algorithm can be found. All of them are limited by the tradeoff between speed and the quality of the resulting mesh.

We claim that besides specific technical properties the following aspects are essential for practical use:

- *Robustness of the algorithm:* most real world data contain topological inconsistencies, e.g. complex vertices. Thus, algorithms assuming the input data to come as (bounded) 2-manifolds would fail. This aspect can be handled by topology modifying algorithms [6, 21, 16, 9].

- *Intuitive parameters to control the reduction:* most people using mesh reduction software are not experts in the field of computer graphics algorithms. Thus, intuitive parameters to control the simplification process are strongly required.

- *Thoughtful use of computer hardware resources:* mesh reduction techniques have to be fast, e.g. with respect to Schroeder's recent definition [21] of 10^8 reduced triangles per day. Yet, the reduction process has to fit into the memory of current computer systems.

As mentioned above, any mesh decimation algorithm using edge collapse operations may be used to convert an arbitrary triangle mesh into the PM representation. According the above claims we now present the ingredients that fulfill these requirements in the next sections. Afterwards, we put these ingredients together to form an efficient mesh simplification algorithm that may be used to generate PMs.

5 Robustness

Besides wavelet-methods [7, 5] and re-tiling of surfaces [24] most known mesh decimation algorithms iteratively reduce a given mesh. A sequence of simple topological operations is applied to the current mesh removing certain geometric entities in each step. Possible basic operations include vertex removal and re-triangulation [22], general edge collapse [10], half edge collapse [13], and vertex collapse [6, 21].

Most current iterative mesh decimation algorithms use a priority queue to obtain optimal approximations to the original mesh. Their generic structure is as follows:

```
For all geometric entities {
  Measure cost for applying operator
  Put result into priority queue
}
Loop until queue empty {
  Perform operator with least cost
  Update costs in queue
}
```

The computationally most expensive part is measuring and updating the cost function for the potential execution of a topological operation. It is intrinsic to each algorithm, how the cost function for every possible operation is calculated. An algorithm may consider the geometric deviation as the cost for performing an operation or

the change of the "fairness" of the modified surface. The fairness is typically related to first or second order surface derivatives and we call the cost function that performs the appropriate calculation "fairing oracle" of order one or two.

Possible topological inconsistencies in the mesh, e.g., complex vertices [22] or edges, complicate the computation of the cost function. Such inconsistencies may occur for different reasons. They could be used to represent a feature or erroneous preprocessing algorithms generate inconsistent meshes. We encountered many cases where the data could have been represented by an ideal 2-manifold. But for algorithmic reasons the generating programs, e.g. data acquisition tools or surface meshers, produced complex entities.

Mesh modifying algorithms, e.g. mesh decimation, can use different strategies to cope with possible topological inconsistencies:

- The mesh can be repaired in a preprocessing step [1]. Since this is usually done by a separate algorithm, valuable information may be lost. But this allows the simplification algorithm to make certain simplifying assumptions.

- The handling of all possible problems may be incorporated into the algorithm. Unfortunately, this complicates implementation and the algorithm itself becomes much more involved.

- Object-oriented techniques can be used to separate the data-structures representing the mesh from the algorithms working on it.

Since the last item combines the advantages of the others, we prefer this strategy.

Fig. 6 shows how to separate the raw mesh from algorithms working on that data by object-oriented software design. The raw mesh data is encapsulated into a data-structure that may exploit meta-knowledge about that data. This encapsulating object has to provide the following interface:

- Access to geometric entities, e.g. lists of vertices, edges, or triangles.

- Methods to inform the data structure about requirements of the algorithm, e.g. 2-manifolds.

- Methods to ask the data structure, if a topological operation is possible with respect to the given requirements.

- Methods to commit basic operations on the data.

The encapsulating data structure can treat topological problems in several ways. It can try to repair them on-the-fly, e.g. by splitting complex vertices [21], while repairing in a preprocessing step could incorporate the methods presented in [1]. Finally, it can simply disallow the modification of the affected sub-meshes.

The separation into a representation and operation layer does not change the structure of an algorithm, but simply decouples independent aspects of processing triangles meshes. Most iterative mesh decimation algorithms can be formulated using such an encapsulating data structure, if the interface is carefully implemented. Thus, it makes sense to encapsulate everything to treat complicated and complex meshes into a separate class. The algorithms can then abstract from these basic tasks and concentrate on their main issue. This eases implementation aspects of the mesh modifying algorithms very much. Further, it helps to make programs more robust, since an algorithm may process a mesh even if that mesh contains unexpected topological problems that would cause the algorithm to fail.

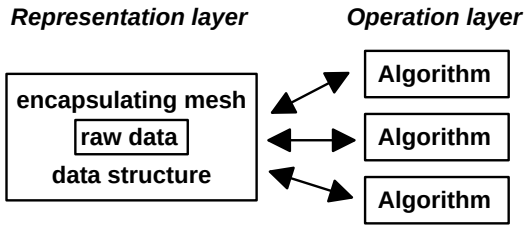


Figure 6: Separating the triangle mesh from the algorithms working on that data.

6 Intuitive parameters

To control the iterative mesh reduction and to terminate the simplification process, it is necessary that the user defines some parameters. People familiar with computer graphics techniques are able to use such algorithms after a short time. But in practice, people have to use these programs who may not understand the underlying algorithm and the parameters for geometric or topologic values. Thus, we want to draw the attention on this aspect of mesh decimation algorithms in this section.

As described previously, most current mesh reduction algorithms use a priority queue to decide which topological operation to perform next. This means that all currently possible operations have to be considered and a cost has to be computed for performing this operation. This cost is a scalar value that is used to sort the set of candidates. Since most algorithms impose several conditions on potential operations, the problem arises, how to combine these different measures into a single scalar value. Several strategies are possible:

- The easiest case occurs if only a single measure is used that directly provides a scalar value, e.g. the geometric deviation of the current from the original mesh [12].

- If the cost function considers more than one aspect, each of them providing scalar values, these can be combined by a weighted sum. E.g., [18] combines the geometric deviation and a measure of the tilt of the surface normals, or [10] uses a scalar energy function that incorporates a large number of different aspects of the generated mesh.

- One adequate aspect is chosen to provide the scalar value, while the others are used for a binary decision that define the *set of candidates* which are actually considered for elimination.

The first strategy considers only a small portion of the information given in a mesh. The second strategy requires the user to choose the weighting factors that may not be intuitive. Further, these factors need not to be invariant under geometric transformation of the mesh. Scaling of objects to the unit cube does not help since the bounding box of an object in general varies under rotation. We find the third strategy most promising for several reasons. First, it reduces the set of candidates that have to be further considered by using binary decisions (this also reduces computation time). Further, it makes the need for counter-intuitive weighting factors unnecessary. We have applied this strategy in our implementation of the algorithm by Ronfard and Rossignac [18] and found the resulting program much more easy to handle.

Usually, authors of mesh reduction algorithms can produce best results when using their own implementations since they know exactly how the parameters affect the reduction process. But non-experts often have difficulties to find the optimal setting. Keeping this in mind, we now want to describe the ingredients for a mesh

simplification algorithm with only very few *intuitive* parameters by following the third of the above strategies.

First, we use the half-edge collapse as topological operator, i.e., vertices are pulled into one of their neighbors, since it eliminates degrees of freedom, i.e., where to put the new vertex, that would have to be optimized involving further user-adjustable parameters.

Next, we use the one-sided Hausdorff distance to measure the deviation of the current mesh from the original vertices [12, 13] as a binary decision. Thus, the user only has to choose an intuitive global error bound to identify the current set of candidate operations for the reduction algorithm. If a specific reduction rate is desired, the program can increase an initial (e.g. automatically chosen) error bound until the desired reduction rate is reached.

Finally, a fairing oracle is used to measure the quality of the generated surface [13] and to steer the greedy reduction algorithm, i.e., to compute the priority of a potential half-edge collapse in the candidate set. We let the user choose between the use of an order 1 or order 2 fairing oracle, since this can be translated into "his language". Order 1 should be used for technical applications or for meshes to be passed to further processing programs, since it is related to local distortion of the mesh (first order derivatives). Order 2 should be chosen for the visualization of meshes, because it considers the local curvature of the modified mesh (second order derivatives). Note that the fairing oracle measures the quality of the modified surface. Hence, "better" configurations are chosen automatically because of the use of the priority queue without the need for any further parameters. Since our implementation of the fairing oracles is closely related to performance issues, we postpone the details to the next section.

As a result of the above recommendations, the user only has to choose an intuitive global error bound and one of two fairing methods (which are related to different application areas). Thus, an unexperienced user will immediately be able to use such an algorithm while still generating approximating meshes of high quality. Of course, other fairness criteria or error metrics based on texture and color are possible according to special requirements of specific applications.

7 Thoughtful use of hardware resources

In the previous section, we have presented the ingredients for an easy to use mesh reduction algorithm. In this section, we focus on implementation issues beyond the separation into an encapsulating data structure and the abstract reduction algorithm as discussed in Section 5. We show how the memory resources can be efficiently used by reducing edge-based algorithms to more compact vertex-based structures. Further, we show how high performance can be achieved at the same time by efficiently implementing the above ingredients.

Euler's formula indicates that a triangle mesh with n vertices has about $3n$ undirected and about $6n$ directed (half-)edges. Thus, each iterative mesh simplification algorithm using a priority queue for the potential vertex removals or edge collapses as topological operations, has to consider that number of geometric entities. A straight forward implementation would have to store a maximum of the same number of candidate topological operations in the queue.

Collecting all edges emanating from one vertex reduces the edge-based oracles and operations to vertex-based ones by local pre-selection of the best candidate. If one edge collapse is performed, all edges starting from the removed vertex have to be removed from the priority queue anyway. Thus, it is not necessary to store all valid potential edge collapses in the priority queue, but only the best one for each vertex. This enables the efficient use of the half-edge collapse as topological operator for the iterative mesh reduction, since only a priority queue for n potential entries is required instead of $6n$.

Further efficiency gains result from the exploitation of geometric coherence which avoids recalculation of intermediate results. Since the priorities of all edges emanating from one vertex are calculated at the same time, intermediate information can be stored and reused for neighboring edges.

With this knowledge we can efficiently implement functions `calcVertexPrio()` and `updateVertexPrio()` for calculating and updating the priorities of a vertex, i.e., for all edges emanating from the vertex, and for identifying the "best" edge. `calcVertexPrio()` first checks for each starting edge of a vertex if the half-edge collapse satisfies the specified error tolerance, and, if yes, calculates the priority of these operations. Finally, it returns that half-edge collapse with the highest priority. `updateVertexPrio()` may be used, if it is known that the geometric deviation of the modified geometry has not changed since the last priority calculation and thus only re-calculation of the fairing oracle (i.e., the priority) is necessary. Using these functions, our mesh simplification algorithm has the following structure:

```
ALGORITHM: simplify mesh

INPUT:
  M: original triangle mesh
  d: max. geometric deviation
  o: order 1 or 2

OUTPUT:
  R: reduced triangle mesh

  For all vertices v of M {
    p = calcVertexPrio( v, d, o );
    add (p,v) to queue;
  }
  Loop until queue empty {
    get next vertex v from queue;
    if ( removal of v possible ) {
      performCollapse( v->e );
      For all vertices v' that require
      recalculation of the priority {
         p = calcVertexPrio( v', d, o );
         update (p,v') in queue;
      }
      For all vertices v' that require
      update of the priority {
         p = updateVertexPrio( v', d, o );
         update (p,v') in queue;
      }
    }
  }
```

As mentioned previously, calculation of the priority of a half-edge collapse involves the use of a "fairing oracle". We provide an oracle of order 1 and of order 2 and let the user choose one of those two that will be used by `calcVertexPrio()` and `updateVertexPrio()` to calculate the priority of a half-edge collapse.

Our order 1 fairing oracle is related to the local distortion of the mesh and is implemented as follows. It uses the function `round(t)` to determine the "roundness" of a triangle t, i.e., the ratio of the longest edge to the radius of the inner circle. First, we calculate for all triangles t neighboring the vertex v the maximum value r_o of their roundness. This can be done in a preprocessing-step, i.e., at the beginning of `calcVertexPrio(v)`, since this value is the same for all potential edge collapse operations starting at v. Next, we calculate the maximum value r_n of the roundness for the modified triangles. If $r_n < r_o$, we assign this decrease to the priority for performing that edge collapse. Otherwise, we still

allow that collapse, if the value of r_n is below a prescribed maximum value, but assign those collapses a priority less than that of the "enhancing" ones. This gives an expert user the freedom to adjust one further intuitive parameter, if he really desires, but frees non-expert users from the need to handle that parameter. The support of this oracle is shown in Fig. 7 (dark gray region).

Our order 2 fairing oracle is is related to the local curvature of a mesh and is efficiently implemented in the following way. We sum over the dihedral angles within the modified sub-mesh and those of the modified triangles and their neighbors that are not changed. The larger this sum is, the more cost is assigned to a potential edge collapse. For expert users, we provide the additional parameter α that disallows edge collapses that would create geometry with a dihedral angle larger than α. As in the order 1 case, we allow worse angles as long as they improve the situation locally. For non-expert users, we default the parameter α to $\pi/2$ to avoid degeneration of the geometry. The support of this order 2 fairing oracle is again shown in Fig. 7 (dark and medium gray regions). It is obviously larger than the support of the order 1 oracle, since a larger sub-mesh is considered. This requires more priorities to be updated if a half-edge collapse is performed.

The performance-bottleneck for the order 2 fairing oracle is the fact that a large number of priorities have to be updated after each collapse operation. As discussed in the caption of Fig. 7, we reduced the number of vertices that need to be updated by the above implementation of our order 2 fairing oracle (by reducing the support of a general order 2 oracle). Computations can be further reduced by the heuristic assumption that the edge with the least cost starting from such a vertex remains the one with the least cost (cf. Fig 7) and no local search to find the "best" edge is necessary. This assumption is exact in most cases. Thus, only the priorities of some edges have to be updated. In the next section we verify that this strategy both speeds up the performance and still provides good results.

8 Results

We verify the usability of our simplification algorithm by applying it to meshes of different application areas in this section. All times are benchmarked on a SGI, R10000, 195MHz.

First, we used an iso-surface extracted from volume data. The original mesh consists of 81,132 triangles (cf. Fig. 8a). Table 1 shows statistics for applying our reduction algorithm with various error bounds and using either order 1, order 2, or fast order 2 fairing oracles. Our fast update strategy for the order 2 oracle generates results that are of the same visual quality, but it is clearly faster. Fig. 8b-c show reduced meshes for a global error bound of $\varepsilon = .01$ using order 1 or order 2 fairing oracles, respectively. The order 1 fairing oracle does not eliminate as many triangles as the order 2 method, since the triangles do not have the freedom to elongate and thus adapt to the local geometry. Because there is no need for updating as many vertices as for the order 2 method, the order 1 method is clearly the fastest way to reduce a mesh.

We applied our algorithm to a large variety of further models and got equally satisfying results. E.g., Fig. 10 shows the Stanford-Buddha that has been generated by merging different scans (1,087,716 triangles), and Table 2 provides the statistics.

Note that our algorithm clearly meets the Schroeder bound of 10^8 reduced triangles per day [21] (about 1157 reduced triangles per second) even for high reduction rates. For extreme reduction rates, the performance drops below, because of the expensive geometric deviation test (Hausdorff). Notice that the algorithm calculates the exact one-sided Hausdorff distance of the original vertices to the reduced mesh in each iteration to attain a very high reduction while exactly following the prescribed error tolerance. Estimating the error by different error metrics like error quadrics [6] or error accumulation [21] would obviously speed up the algorithm significantly and the reduction rate would be constant during the whole decimation. But the calculated error would either be only an estimation or an upper bound of the actual error. In the first case, it cannot be guaranteed that a specified global error is satisfied, while in the second case the mesh decimation algorithm typically is not able to reduce as many triangles as when calculating the actual error for the same specified error tolerance.

oracle	ε	#Δ coarse	Δ/sec
order 1	.01	28,270 (34.8%)	2634
	.1	3,784 (4.66%)	1992
	1	416 (0.51%)	1445
order 2	.01	23,068 (28.4%)	1659
	.1	3,222 (3.97%)	1099
	1	344 (0.42%)	784
order 2 fast update	.01	23,110 (28.5%)	2181
	.1	3,214 (3.96%)	1602
	1	340 (0.42%)	1190

Table 1: Reduction statistics for the iso-surface shown in Fig. 8a consisting of 81,132 triangles (bounding box size: 46 x 46 x 65). The user supplies the choice of the order and the global error bound. Given is the number of triangles the algorithm generates and performance timings for removed triangles per second. The reduction process requires about 10MB memory.

oracle	ε	#Δ coarse	Δ/sec.
order 2	10^{-3}	286,646 (26.4%)	2345
fast	10^{-2}	30,404 (2.80%)	1542
update	10^{-1}	3,774 (0.35%)	1004

Table 2: Reduction statistics for the scanned Stanford-Buddha statue consisting of 1,087,716 triangles (bounding box size is 8.13 x 19.8 x 8.14). The reduction process requires about 117MB memory.

The Buddha mesh example demonstrates that even very large meshes can be processed within a PC's memory. Hence, the proposed mesh reduction scheme opens the PC platform to applications that have to visualize this kind of data sets without requiring expensive graphics hardware. Yet the scheme provides sophisticated features like global error tolerances and fairness control.

Our current implementation uses a maximum of about 110 bytes per input triangle to store geometric and topological information as well as redundant information to speed up the evaluation of the global distance measure and the fairness oracle. If we trade redundancy for computing time by not caching any intermediate results, we can reduce the memory requirements down to 65 bytes per triangle (assuming 4 bytes pointers and integers), or even further.

Surface attributes like colors, normals, or texture coordinates can also be considered by our decimation algorithm. This can easily be achieved by using appropriate error-metrics in the same way as the geometric deviation test. We simply add further binary decisions for each type of attribute which reduces the current set of candidate operations. Fig. 9 shows a colored sphere with 407.040 triangles on the top left. The other two meshes in the top row show the results when decimating the meshes down to 19.102 (middle) and 5.760 (right) triangles. Both reduced meshes satisfy the same global geometric deviation but different errors have been allowed for the color. The images in the lower row show decimated meshes of the same geometric complexity, respectively, but without considering deviation in color. The bottom left image shows a close-up of the upper middle mesh from a different viewpoint. Note that more triangles are used especially at the coast and the mountains to maintain color attributes.

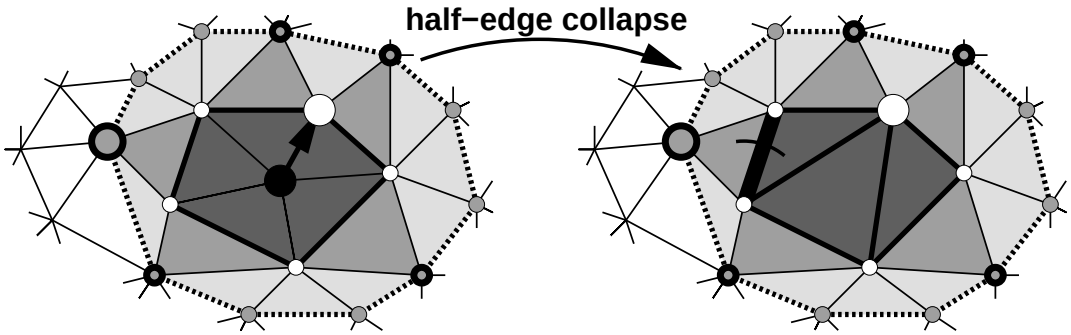


Figure 7: After performing an half-edge collapse, the vertices (white) on the border of the modified geometry (dark gray, the support of the order 1 fairing oracle) have to be recalculated. If using a general order 2 fairing oracle, the priorities of further vertices (gray) have to be updated as well, because of the larger support (dark, medium, and light gray). Since our implementation of an order 2 fairing oracle is designed to have a smaller support (not including the light gray regions), fewer priorities have to be updated (only the bold outlined gray vertices), e.g., the priority of the fat gray vertex, since only a single dihedral angle changes (arc in the right figure). For those "outer vertices", this fact can be exploited to speed up the update of priorities.

9 Conclusions

We have identified relevant requirements for the practical usability of mesh decimation algorithms: robustness, intuitive parameters, and scalable performance with respect to both CPU and memory requirements.

We proposed to encapsulate the raw mesh data into a data structure to simplify algorithms processing triangular meshes by abstracting from special cases. We identified major requirements for the practical usability of general purpose mesh decimation algorithms. We discussed implementation aspects to efficiently attain our recommendations. This includes how to reduce edge-based algorithms to a vertex-based structure to minimize memory requirements, and fast higher-order surface fairing methods. We combined these ingredients to a new mesh simplification algorithm and verified the achievement of the design goals. Moreover, our decimation algorithm may consider any surface attribute like color while still allowing easy usability.

Our implementation provides high CPU-performance, up to more than two times the Schroeder bound of 10^8 removed triangles per day, and may be run even on computer systems with limited memory, while not requiring a single parameter to be adjusted. Decimation time can be significantly reduced if using less restrictive error metrics, e.g. error quadrics or error accumulation instead of the computationally expensive one-sided Hausdorff distance.

Recording the performed half-edge collapses during the iterative mesh decimation process allows to generate a hierarchical triangle mesh in the progressive mesh format. We have implemented an ActiveX-control for Microsoft Windows and a general helper application using Motif and OpenGL that allow to progressively receive such records.

Since progressive meshes also allow the display of a view-dependent triangle mesh [11] we are planning to extend our mesh viewers by this feature to decrease the number triangles.

10 Acknowledgments

We would like to thank our students Peter Eberlein and Thilo Bareuther who have implemented the ActiveX-control and the helper application to receive and display progressive meshes.

References

[1] Gill Barequet and Subodh Kumar. Repairing CAD Models. In *IEEE Visualization '97, Conference Proceedings*, pages 363–370, 1997.

[2] Swen Campagna. `URL: http://www9.informatik.uni-erlangen.de/Persons/Campagna/progmesh`.

[3] Swen Campagna and Hans-Peter Seidel. Generating and Displaying Progressive Meshes. In *3D Image Analysis and Synthesis '97, Conference Proceedings*, pages 35–42, 1997.

[4] Rikk Carey and Gavin Bell. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley, 1997.

[5] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6–11, 1995)*, Computer Graphics Proceedings, Annual Conference Series, pages 173–182, August 1995.

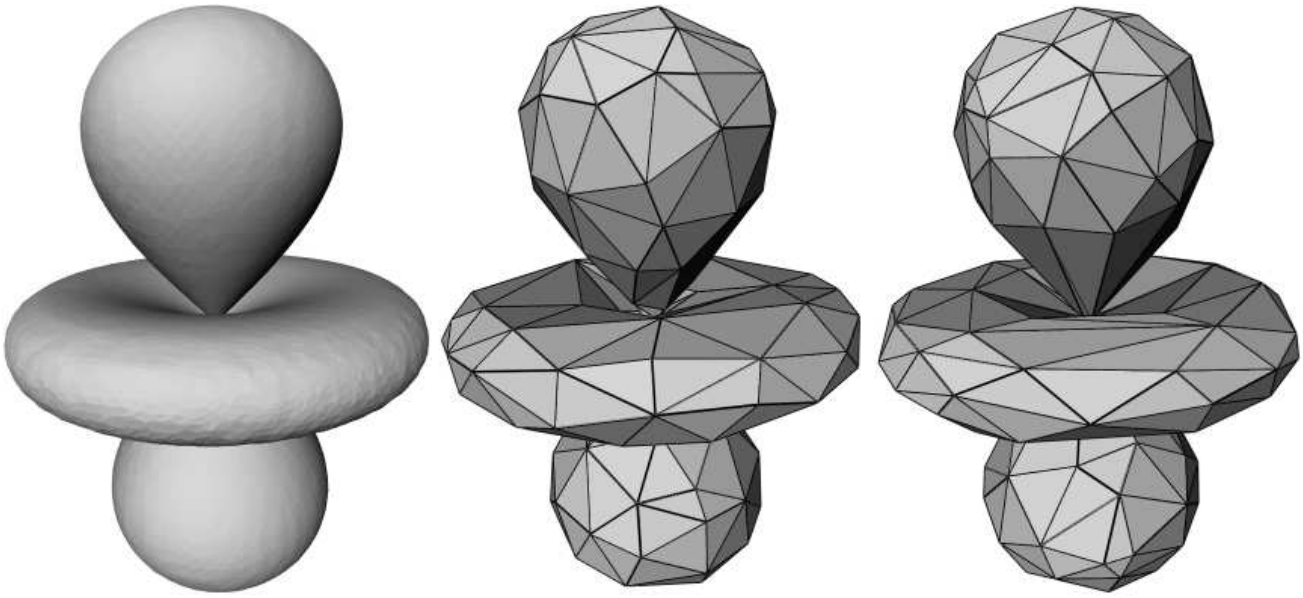
[6] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 209–218, August 1997.

[7] M. H. Gross, R. Gatti, and O. Staadt. Fast Multiresolution Surface Meshing. In *IEEE Visualization '95, Conference Proceedings*, pages 135–142, 1995.

[8] Stefan Gumhold and Wolfgang Straßer. Real Time Compression of Ttriangle Mesh Connectivity, July 1998.

[9] Taosong He, Lichan Hong, Amitabh Varshney, and Sidney W. Wang. Controlled Topology Simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–183, June 1996.

[10] Hugues Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH '96 (New Orleans, Louisiana, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108, August 1996.



(a) (b) (c)

Figure 8: An iso-surface extracted from volume data from a numerical simulation (courtesy Roberto Grosso). (a) shows the original mesh with 81,132 triangles, (b) and (c) simplified surfaces using order 1 or order 2 fairing oracles, respectively. Notice in (c), how the triangles near the center elongate and thus adapt to the local geometry.

[11] Hugues Hoppe. View-Dependent Refinement of Progressive Meshes. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 189–198, August 1997.

[12] Reinhard Klein, Gunther Liebich, and W. Straßer. Mesh Reduction with Error Control. In *IEEE Visualization '96, Conference Proceedings*, pages 311–318, 1996.

[13] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A General Framework for Mesh Decimation. In *Proceedings of Graphics Interface '98*, pages 43–50, 1998.

[14] Aaron W.F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces, July 1998.

[15] Michael Lounsbery, Tony DeRose, and Joe Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[16] Jovan Popović and Hugues Hoppe. Progressive Simplicial Complexes. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 217–224, August 1997.

[17] Enrico Puppo and Roberto Scopigno. Simplification, LOD and Multiresolution Principles and Applications, 1997. Tutorial Eurographics '97.

[18] Rémi Ronfard and Jarek Rossignac. Full-Range Approximation of Triangulated Polyhedra. In *Computer Graphics Forum, Proceedings of Eurographics '96*, pages C67–C76, 1996.

[19] Jarek Rossignac. Simplification and Compression of 3D Scenes, 1997. Tutorial Eurographics '97.

[20] Will Schroeder. Polygon Reduction Techniques, 1995. IEEE Visualization '95. Advanced Techniques for Scientific Visualization, Section 1.

[21] William J. Schroeder. A Topology Modifying Progressive Decimation Algorithm. In *IEEE Visualization '97, Conference Proceedings*, pages 205–212, 1997.

[22] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[23] Gabriel Taubin and Jarek Rossignac. Geometric Compression through Topological Surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[24] Greg Turk. Re-Tiling Polygonal Surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

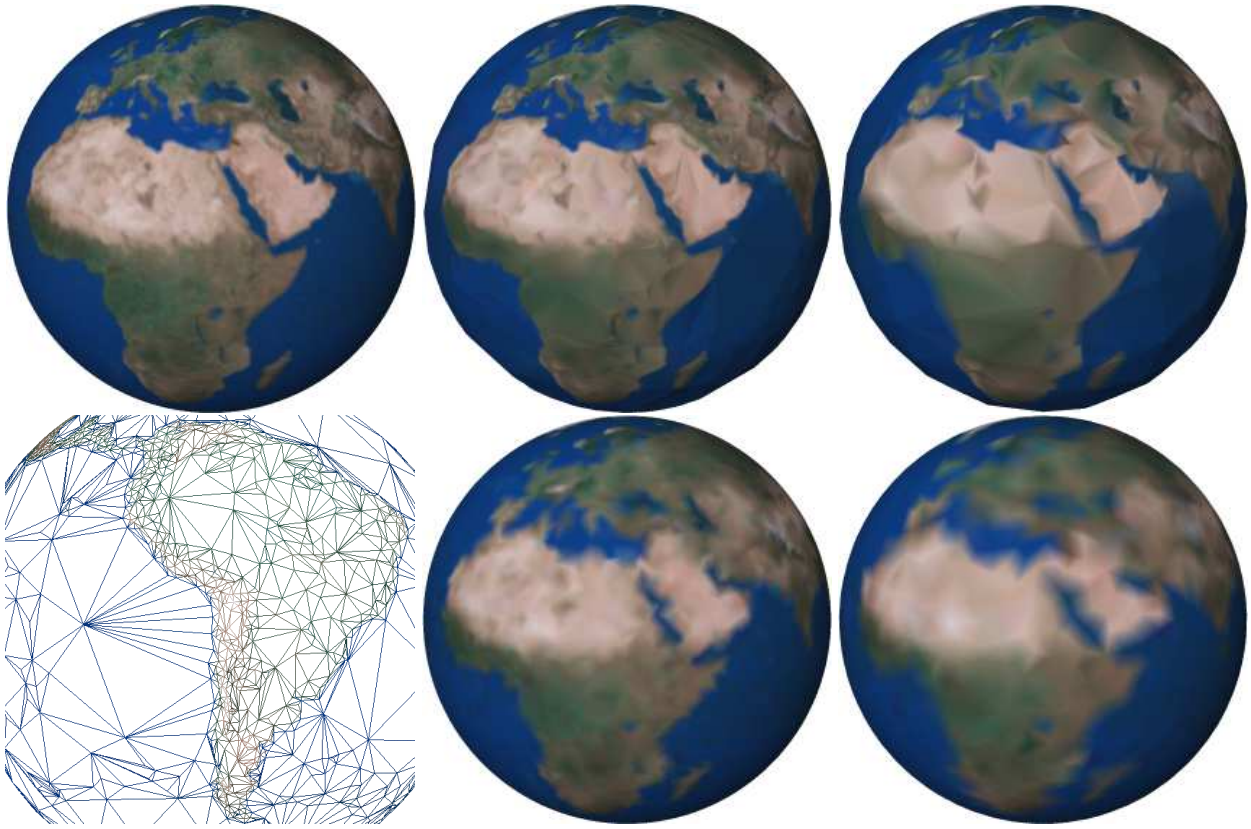


Figure 9: A colored mesh with 407.040 triangles on the top left. Decimation down to 19.102 (upper middle), 19.410 (lower middle), 5.760 (upper right), and 5.770 (lower right) triangles. The decimation algorithm considered vertex attributes for the upper meshes (satisfying the same geometric deviation), while only geometric deviation has been considered for the lower meshes. The bottom left image shows a close-up of the upper middle mesh from a different viewpoint.

Figure 10: The complex Stanford-Buddha statue, generated by merging multiple scans. From left to right: original mesh with 1,087,716 triangles, simplified meshes using the order 2 oracle with fast update, 286,578, 30,392, and 3,774 triangles, respectively. Note how high-frequency detail is removed first due to the fairing-oracle.