Eurographics Italian Chapter Conference (2010)
E. Puppo, A. Brogni, and L. De Floriani (Editors)

Adaptive quad mesh simplification

A. Bozzo[1] and D. Panozzo[1] and E. Puppo[1] and N. Pietroni[2] and L. Rocca[1]

[1]Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Italy
[2]Visual Computing Group - ISTI-CNR, Pisa, Italy

Abstract

*We present an improved algorithm for the progressive simplification of quad meshes, which adapts the resolution of the mesh to details of the modeled shape. We extend previous work [TPC*10], by simplifying the approach and combining it with the concept of* Fitmaps *introduced in [PPT*10]. The new algorithm has several advantages: it is simpler and more robust; it does not need a parametrization of the input shape; it is adaptive; and it preserves projectability of the output mesh to the input shape, thus supporting displacement mapping. We present experimental results on a variety of datasets, showing relevant improvement over previous results under several aspects.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surface representations

1. Introduction

Quad meshes, i.e. meshes composed entirely of quadrilaterals, are becoming more and more popular in computer graphics and geometric modeling, because of their high impact in a variety of applications. In recent years, several methods have been proposed for the simplification of quad meshes, i.e., the task of producing a low complexity mesh M' out of a high complexity one M (see Section 2). Compared to the case of triangle meshes, simplification of quad meshes poses extra challenges, because connectivity is more constrained, and also quads are less adaptive than triangles. The main goal addressed by most methods is to obtain a mesh with good quality, i.e., having almost flat and square faces, and most vertices with regular valence four. On the other hand, quality of approximation and adaptiveness are usually addressed only indirectly.

In this paper, we extend the method proposed in [TPC*10] to explicitly take into account adaptiveness and quality of approximation, while also improving performance and results in terms of mesh quality. Our algorithm, like the previous one, generates a mesh made of convex, almost right-angled, flat quads; it does this by progressively simplifying an initial quad mesh at high resolution, through the combined effect of local operations that modify mesh connectivity, and smoothing operations that improve the geometry of quads. Depending on needs, our algorithm can be used to obtain either a mesh with equally sided quads and a uniform distribution of vertices, as in [TPC*10], or an adaptive mesh in which resolution of elements is distributed according to details of the modeled shape.

The basic technique is modified in several aspects: we adopt a smaller and better controllable set of local operators; we change the criteria to trigger such operators; we adopt a simpler and more efficient method for mesh smoothing, which does not require a parametrization of the input mesh. These changes alone provide a simpler and more robust algorithm, which achieves much better results in terms of quality of the output mesh, by addressing the same goals as the original method.

Next, we further extend the method, to explicitly address accuracy and adaptivity, by incorporating *Fitmaps,* which were introduced in [PPT*10]. Fitmaps are a pair of scalar fields defined on the input shape and computed during preprocessing, which estimate locally how well a portion of shape can be approximated with a bilinear patch. Such fields work as a guidance to select operations during the simplification process: they allow us to adapt the output tessellation to details of the input, to evenly distribute the approximation error, and to preserve projectability, i.e., the possibility to map the output surface to the input by normal projection, thus supporting displacement mapping.

Our algorithm has been experimented on a variety of datasets. We provide visual as well as numerical compar-

isons that demonstrate significantly better results with respect to the previous algorithm.

2. Related work

Compared to the more consolidated problem of simplifying triangle meshes [LRC*02], quad mesh simplification is harder, and algorithms supporting this task have been developed only recently. In triangle mesh simplification, the main objective is to obtain a mesh with a reduced number of faces that approximates well the input shape. In quad mesh simplification, the main challenge is to obtain a mesh that maximizes regularity in terms of both connectivity (valence of vertices) and geometry (shape of elements), while quality of approximation is usually addressed only as a side issue.

Quest for regularity has often leaded to global methods that are inherently not adaptive and not progressive. Remeshing algorithms, for instance, build a completely new mesh, not necessarily at lower complexity, which represents the input shape well and has a superior quality [BZK09, DBG*06, HZM*08, KNP07, RLL*06]. The main objective of remeshing is a mesh made of square faces of uniform size and with a nearly regular connectivity. An additional objective may be alignment of elements to either feature lines, or curvature, or a cross field defined on the surface.

The progressive and local approach typical of most algorithms for triangle meshes cannot be adapted to quad simplification easily. Collapse of a quad diagonal is recognized as a valid operation to simplify the mesh locally while preserving quad structure, but a simplification algorithms cannot be based just on it, because it tends to severely corrupt the shape of surviving elements and to destroy regular connectivity. Early algorithms for progressive mesh simplification combine diagonal collapse with operations that affect larger areas, such as poly-chord collapse [DSSC08, SDW*09]. However, global operations have several drawbacks: their all-or-nothing nature makes them a clumsy tool to maximize any sought objective; they are not suitable for selective refinement; and they are difficult to apply in an out-of-core context. More recent methods rely just on local operations: in [DSC09] poly-chord collapses are split into smaller independent sub-steps; while in [TPC*10] six strictly local primitive operations are used. Our algorithm uses just four out of those six operations.

For objects with details at different scales, such as natural shapes acquired through range scanning, the contrasting objectives of having a good fit and a coarse control mesh can be achieved only if the mesh is adaptive. Adaptivity and regularity are highly contrasting objectives: transition from coarse to fine patches requires introducing some irregular vertices, or warping the shape of some quads, or both. In [TPC*10] a method is proposed to obtain an adaptive mesh by naturally blending such two objectives, but this requires a scalar field to be user-defined on the input mesh, which describes the importance of different parts of the mesh. In this paper, we automatize such an approach by incorporating *Fitmaps* proposed in [PPT*10] (see Section 3.3).

3. The algorithm

As already mentioned in the introduction, our algorithm modifies the technique originally proposed in [TPC*10], and extends it to an adaptive technique by using Fitmaps from [PPT*10]. In Subsection 3.1 we describe the outline of the original method; in Subsection 3.2 we describe our modifications to the non-adaptive (homeometric) case; and in Subsection 3.3 we describe Fitmaps as well as their use to obtain the adaptive algorithm.

3.1. The original method

The algorithm described in [TPC*10] progressively simplifies an initial quad mesh by applying local operators. Three classes of operators are defined: two coarsening operators - diagonal collapse and edge collapse - that reduce the number of elements in the mesh; two optimizing operators - edge rotate and vertex rotate - that improve mesh quality; and two cleaning operations - doublet removal and singlet removal - that remove degenerate configurations. Such operators are interleaved with tangential smoothing that displaces vertices on the surface while maintaining the overall shape of the object, in order to improve the geometry of faces.

Given a quad mesh M_0 at high resolution (which may be generated from a triangle mesh, with a conversion algorithm also presented in [TPC*10]), the simplification algorithm has the following general outline:

1. Iteratively process mesh M_i to produce mesh M_{i+1} until user-defined criterion is met. At each cycle:
   a. for a fixed number of times:
      i. perform any profitable local optimizing operation, until none is available, then clean degeneracies
      ii. select and perform a local coarsening operation and clean degeneracies
   b. smooth the mesh just in those zones affected by local operations
2. Perform global smoothing of mesh M_n.

The sought objective is to generate a mesh with faces as squared as possible and as uniform as possible in their size: such a condition is called *homeometry*. The variance of lengths of edges and diagonals measures how far a mesh is from being homeometric.

Collapse operations applied during Step 1.a.ii simplify the mesh. The shortest element (either a diagonal or an edge) is selected at each step for collapse. The other operations are aimed, on one hand, at improving mesh quality in terms of shape (Step 1.a.i) and sample distribution (Step 1.b) and, on

the other hand, at driving the selection of best coarsening operations to be performed next.

The stop criterion is user-defined and it is usually related to the size of the output.

3.2. Basic simplification

In its basic version, our algorithm pursues the same objective of the original algorithm, i.e., homeometry, by recombining similar ingredients. The outline of the algorithm differs from the original one, as follows:

- Iteratively process mesh M_i to produce mesh M_{i+1} until user-defined criterion is met. At each cycle:
  1. select and perform a diagonal collapse (and related cleaning operations);
  2. perform any profitable edge rotation (and related cleaning operations) in the portion of mesh affected by collapse;
  3. smooth the mesh just in the 1-ring of the zone affected by previous operations;

This outline essentially introduces a finer level of granularity in the simplification loop: during each cycle just one local coarsening operation is executed, then the mesh is adjusted only locally with optimization and cleaning operations and with smoothing. Additionally, the global smoothing at the end of the simplification process is not required anymore.

Local operations. Just one type of coarsening operation, i.e., *diagonal collapse* is used in Step 1 (see Figure 1 left). This operation eliminates a quad q, two edges and a vertex, by collapsing one diagonal of q. As in the original algorithm, we maintain a heap of potential collapses, prioritized according to least cost, which is kept up-to-date throughout the simplification process. In the basic case, the shortest diagonal is collapsed at each step. The position of vertex v resulting from collapsing a diagonal d is initialized to the midpoint of d, then v is displaced by re-projecting it onto the surface of M_0. Vertex projection is supported from a spatial index, which is built on M_0 during initialization, and supports efficient ray casting. Given the position of v, together with the surface normal n_v at v estimated on mesh M_i at the initial position of v, we find the closest point p on M_0 that is hit by a ray cast from v along direction n_v, and we displace v to p.

Just one type of optimization operation, i.e., *edge rotation* is used in Step 2 (see Figure 1 right). This operation substitutes an existing edge e with one of the other two diagonals of the hexagon formed by the two quads incident at e. Edge rotations have also the side effect of modifying lengths of diagonals, effectively driving the selection of local operations to be performed next.

After performing a diagonal collapse, we consider all faces in the 1-ring of the collapsed element and we test their edges for potential rotations. The criterion for triggering a swap operation is completely different from the one used in the original algorithm: it aims at improving the regularity of mesh connectivity, rather than squareness of faces. Given an edge e, let $v_1,\ldots,v_6$ be the vertices bounding the pair of faces incident at e. We measure the valence $D(v_i)$ of each such vertex and we set an energy $\sum_{i=1}^{6}|D(v_i)-4|$. We rotate e if and only if such an operation decreases this energy. In this way, we tend to increase the number of regular vertices of M_{i+1}.

Cleaning operations are the same ones as in the original algorithm, i.e., *doublet removal* and *singlet removal* (see Figure 1 center). Collapse and swap operations may generate *doublets*, i.e., configurations where two adjacent quads share two consecutive edges, which join at a vertex with valence two. Doublet-removal is applied to eliminate a doublet as soon as it appears, by simply merging the two quads. A doublet removal may generate other doublets, which are eliminated recursively, and, in rare cases, a *singlet,* i.e., a configuration in which a face has two consecutive edges coincident, resulting in a vertex of valence one. A singlet is also eliminated as soon as it appears, by removing the degenerate quad and joining its two adjacent quads at a common edge.

Tangent space smoothing. This operation consists in moving vertices so that they never leave the surface of the mesh, increasing the overall homeometry at the same time. For a better match between the simplified model and the original mesh M_0, vertices are kept on M_0, rather than on current mesh M_i.

Smoothing is performed through a relaxation process and it has two main purposes: it directly improves mesh quality and it helps selecting the best candidate operation to perform next. Relaxation is always applied locally on the zone affected by previous operations (Step 3). Each vertex involved in relaxation is subject to forces of a system of springs. The rest position of each spring coincides with the ideal length of its associated edge, or diagonal, which corresponds to the average length μ of edges of M_i, or to $\sqrt{2}\mu$, respectively, for a fully homeometric mesh.

In the original algorithm, mesh parametrization was used and relaxation was performed in parametric space. Such an approach is rather accurate, but it involves computing a parametrization of mesh M_0, i.e., resolving a problem possibly harder than mesh simplification. We resort to a much simpler approach, which exploits the spatial index already used for vertex projection during diagonal collapse, and turns out to be equally effective. Each vertex is relaxed independently, by moving it first to the rest position of its system of springs in 3D. Relaxation in 3D is straightforward to compute, but, in general, it moves a vertex v out from the surface of M_0, so we re-project v onto the original surface M_0.

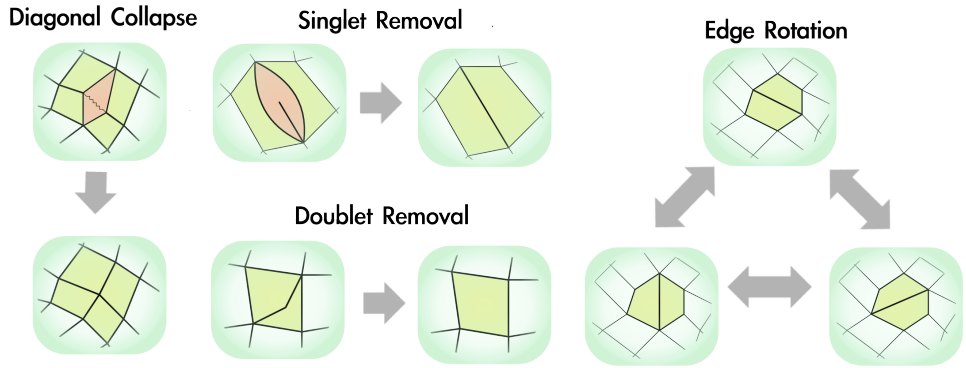


Figure 1: *The set of local operations used during the simplification.*

3.3. Adaptive simplification

The adaptive version of the algorithm is obtained by relaxing homeometry, in favor of a criterion that tends to uniformly distribute approximation error. Classical progressive methods for triangle mesh simplification schedule local operations by applying, at each cycle, the operation that causes the least increase of error. This approach, however, is computationally expensive, since it involves simulating the effect of all possible operations before performing them, and it cannot be extended easily to quad mesh simplification, while also preserving requirements on mesh regularity.

Following [PPT*10], we adopt a rather different approach. We use a guidance field, which is computed on the input mesh during pre-processing, to drive the selection of operations. This field provides an estimate on the density of vertices, for each portion of surface, which is required to distribute error evenly. Another guidance field, which is also computed during pre-processing, helps to avoid warping the mesh too much, thus preserving *projectability,* an important property that is crucial to preserve the possibility of applying *displacement mapping* to reconstruct the surface at an arbitrarily good level of detail during rendering.

Fitmaps. Generic *Fitmaps* have been introduced in [PPT*10], and they have been used in the specific case of surfaces made of bicubic patches. Here, we describe Fitmpas for the simpler case of bilinear patches.

A Fitmap consists of a pair of values for each point p of a surface: the *S-fitmap* F_S and the *M-fitmap* F_M. The *S-fitmap* ("Scale" fitmap) estimates, at each point p, how the error of fitting a bilinear patch to a neighborhood of p increases with radius of the neighborhood. The *M-fitmap* ("Maximal radius" fitmap) estimates how much a face can extend around each point p before correct projection of the face to the input shape through normal displacement becomes impossible.

The Fitmap of mesh M_0 can be interpreted as a prescription on the patches of an ideal approximation $\tilde{M}$:

- The local radius of each face of $\tilde{M}$ should be inversely proportional to the value of the *S-fitmap* computed at its central point;
- No face of $\tilde{M}$ should have a radius larger than the value of the *M-fitmap* computed at its central point.

The first condition aims at distributing error evenly over $\tilde{M}$, thus improving accuracy for a given number of patches. The second condition aims at preserving projectability, but it also prevents surface inconsistencies that might be caused by excessive warping (e.g., squeezing thin or elongated parts). The two channels of the fitmap for the Rampart model, together with a simplified mesh built based on it by our method are depicted in Figure 2. Note that a fitmap is a property of the input only, which does not depend on the sought mesh $\tilde{M}$; on the other hand, a fitmap does not provide guarantees on the true approximation error and projectability of $\tilde{M}$, but rather a heuristic estimate of such values.

We compute the fitmap at vertices of M_0 and we extend it by linear interpolation to all points of M_0. For each vertex p of M_0, we consider neighborhoods of p of increasing radii $r_0, \ldots, r_h$. In all our experiments, we use $h = 8$, we set r_0 equal to the average length of edges of M_0, r_h equal to 1/4 the length of the diagonal of the bounding box, and we distribute the other radii on an exponential scale.

For estimating the S-fitmap, we fit linear functions to each neighborhood of p and we record each fitting error $E(r_i)$. Linear functions serve here as an easy and conservative surrogate to the more general bilinear patches that constitute our output mesh $\tilde{M}$. The sequence of $E(r_i)$ values provides an estimate of how the fitting error grows in the neighborhood of p.

Next, we compress information provided by these values into a single scalar value. Since we are fitting linear patches,

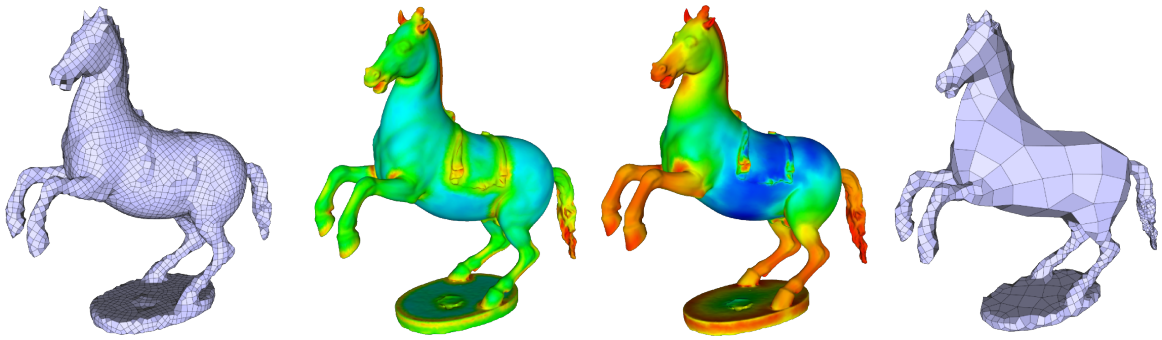


Figure 2: *From left to right: the Rampart dataset simplified wthout using fitmaps (a), the S-fitmap (b), the M-fitmap (c) and another version of the Rampart dataset simplified adaptively with the fitmaps (d).*

we expect error to increase with the quadric power of radius r, thus we model it with a simple function $E(r) = ar^2$. Having collected h measurements of errors at different radii r_i, we fit such function to these values and we recover parameter a.

We set the value for the S-fitmap $F_S(p)$ to $\sqrt{a}$ so that we obtain a function that increases linearly with the radius. In this way, if two patches centered in p_0, p_1 have radii r_0 and r_1, respectively, they contribute approximatively the same error E' if the values of $r_0 \cdot F_S(p_0)$ and $r_1 \cdot F_S(p_1)$ are equal.

The M-fitmap F_M is built together with the S-fitmap. For a given neighborhood of radius r_i, let P_i be the linear function fitted to data. Function P_i defines a plane, let n_i be its surface normal, oriented outwards from the surface. We compute the scalar product between n_i and the normal of each triangle of M_0 spanned by the given neighborhood, and we consider a face to be oriented consistently with P_i if such a value is positive. The value of $F_M(p)$ is set to the largest tested radius at which the portion of neighborhood covered by inconsistent faces is smaller than a "tolerance" threshold τ. Parameter τ can be user-defined, depending on the amount of high frequency noise expected in the input mesh, or on the amount of 3D high frequency detail that could be ignored, to avoid an excessive fragmentation of patches. All figures and experiments in this paper use $\tau = 5\%$.

Simplification based on Fitmaps. Incorporating Fitmaps into the simplification framework is very simple. The S-fitmap is used to weight the length of diagonals to be scheduled for possible collapse. Priority of a diagonal d in the heap is set to

$$|d| \cdot F_S(\phi(c)),$$

where $|d|$ denotes the length of d, c is the center of the face containing d, and $\phi(c)$ is its projection to M_0; projection ϕ is computed along the normal direction of the face containing d, by means of the spatial index that is also used during diagonal collapse and tangential smoothing.

The M-fitmap is used to try avoiding collapses that hinder projectability. Given a potential collapse, we evaluate the M-fitmap at the center of surrounding faces that the collapse would extend, and we perform the collapse only if, at each such face, the M-fitmap is smaller than the radius of the face, measured as the maximal distance between its center and one of its corners.

The M-fitmap can be also used to set an automatic halting condition for the simplification loop, instead of a user-defined criterion. In this case, simplification is halted when no feasible collapses remain.

Implementation. The simplification algorithm has been implemented in C++ as a plugin for Meshlab [CCR08]. It will be released in the official Meshlab distribution soon. The plugin allows to start from a triangle mesh and to convert it to a pure quad mesh using the algorithm of [TPC*10]. Two simplification modes are available: the non-adaptive simplification, that halts when a user-defined number of faces is reached, and the adaptive simplification, which does not require any parameter.

4. Experiments

The proposed method was tested on several datasets coming from range scanning. Results are shown in Figure 5. All experiments have been performed on an Intel i5 2.5 Ghz 4.00 GB, using a single core.

In this section, all tables report the computation times required for simplifying the initial dataset, the vertex valency (max valence and % of regular vertices), the homeometry as in [TPC*10] (min and max, both normalized with ideal length μ) and the Hausdorff distance (computed with MeshLab [CCR08]), with respect to bounding box diagonal.

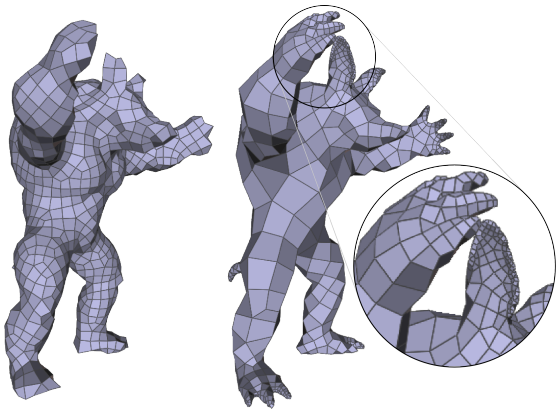
	#faces	Time (s)	val max	reg (%)	Homeometry min	Homeometry max	Dist 10^{-3}
ideal values:		—	—	—	1	1	0
Moai	*8.2K*	13.38	6 - 9	71 - 44	0.34	1.87	0.4
(25k)	3.3K	4.16	5 - 6	71 - 64	0.53	1.77	0.5
	0.6K	2.31	5 - 6	72 - 62	0.59	1.71	3.0
Pensatore	*15K*	33.97	6 - 8	71 - 48	0.43	1.82	0.6
(48k)	10K	5.31	5 - 7	71 - 61	0.43	1.79	0.8
	5K	5.23	5 - 7	72 - 57	0.55	1.76	1.3
	2K	3.11	5 - 6	72 - 68	0.44	1.85	2.4
	1K	1.02	5 - 7	72 - 64	0.44	1.67	3.7
Gargoyle	*11K*	15.01	7 - 7	70 - 61	0.44	2.84	0.8
(24k)	4K	8.05	5 - 7	73 - 57	0.54	1.76	1.8
	2K	2.31	5 - 7	72 - 54	0.58	1.81	3.1
Bunny	*11K*	11.41	8 - 7	75 - 69	0.48	2.64	0.3
(22k)	5K	6.43	5 - 7	72 - 68	0.44	1.89	0.7
	3K	2.16	5 - 6	72 - 61	0.58	1.76	1.2
Fertility	*22K*	6.39	7 - 6	73 - 63	0.39	2.13	0.1
(28k)	5K	18.53	5 - 6	72 - 67	0.51	1.88	0.7
	3.3K	1.98	5 - 7	72 - 71	0.54	1.75	1.0
	2K	1.50	5 - 6	72 - 67	0.57	1.69	1.6
Rampart	*20K*	23.91	7 - 7	68 - 75	0.23	10.49	0.4
(38k)	10K	13.01	7 - 7	71 - 62	0.52	7.45	0.7

Table 1: *Comparisons with Practical Quad Mesh Simplification [TPC*10]. Columns* val *and* reg *reports both our values (on the left) and values from previous work (on the right).*

Comparison with practical quad mesh simplification. In Table 1 we show the results of our algorithm on the same test cases of Table 1 of [TPC*10]. The same datasets have been used and the simplification has been stopped at the same number of faces. Simplification times are comparable with the original algorithm, but our method does not require computing a parametrization for the tangent space smoothing phase. Meshes produced by the proposed algorithm contain about 20% less extraordinary vertices than the original algorithm and they often have a maximum vertex valence of 5. This is due to the new criterion used for edge rotation that strives to produce regular vertices whenever possible. The homeometry is similar with both methods: this is interesting since, in the proposed algorithm, neither the edge rotations nor the smoothing phase explicitly optimize homeometry. Still, the results are comparable, and sometimes even better, than [TPC*10], meaning that our criteria indirectly optimize homeometry.

Adaptive simplification. Statistics of our adaptive simplification algorithm are shown in Table 2. Figure 3 shows that with an adaptive mesh it is possible to better preserve features than with a uniform mesh with the same budget of quads. Small quads are placed on the fingers, ears, nose and tail of the armadillo to better preserve the shape. Big quads are used to cover the legs and torso, because they are sufficient to approximate almost flat regions of the armadillo.

In this case, the simplification is completely automatic and the stopping criterion is provided by the M-fitmap. As it can be seen from Figures 3 and 5, the simplification is very adaptive: the difference in the area of patches varies significantly as the it proceeds (see Figure 4), producing a good approximation of the original shape even with a low number of quads.



(a) (b)

Figure 3: *Two simplified versions of the Armadillo dataset with the same number of faces. (a) is made of equally sided quads. (b) is an adaptive mesh that shows that even with a small number of faces it is possible to preserve small features as the fingers of the armadillo.*

	#faces	Time (s)	val max	reg (%)	Homeometry min	Homeometry max	Dist 10^{-3}
ideal values:		—	—	—	1	1	0
Moai	*94*	23.97	5	70	0.39	1.73	14.52
Pensatore	*116*	75.08	6	68	0.12	2.32	16.29
Gargoyle	*1.5k*	38.46	6	69	0.17	3.60	4.21
Bunny	*681*	33.64	6	69	0.17	4.27	8.71
Rampart	*1.4K*	73.21	6	71	0.10	6.08	4.62
Armadillo	*1.5K*	141.64	6	69	0.09	3.79	4.32

Table 2: *Statistics on experiments with completely automatic adaptive simplification guided by the fitmaps.*

The regularity of the adaptive meshes are inferior compared to the non adaptive meshes of Table 1. This is a side effect of the adaptivity of the mesh. In fact, a transition from a region with small faces to one with big faces in a mesh necessarily occurs through irregular vertices. however, our algorithm generates adaptive meshes with less irregular vertexes than the uniform meshes produced by [TPC*10].

Comparison between homogeneous and adaptive meshes. Finally, in Table 3 we show the difference between

	$\|M\|$	Time (s)	val max	reg (%)	Homeometry min	Homeometry max	Dist 10^{-3}
ideal values:		—	—	—	1	1	0
Moai	*94*	20.72	5	72	0.62	1.57	14.24
Pensatore	*116*	50.30	5	72	0.55	1.82	13.64
Gargoyle	*1.5k*	26.04	5	72	0.52	1.80	3.83
Bunny	*680*	22.77	5	71	0.06	2.15	10.28
Rampart	*1.4K*	49.09	6	71	0.29	2.37	12.82
Armadillo	*1.5K*	90.82	6	73	0.50	1.93	5.96

Table 3: *Statistics on experiments with uniform meshes with the same number of faces of meshes in Table 2.*

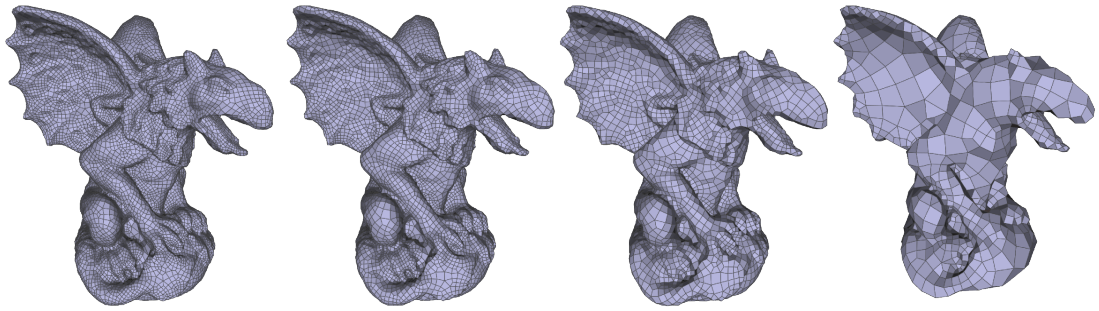


Figure 4: *The gargoyle datasets simplified at 18k, 12k and 6k faces. The adaptivity of the mesh increases during the simplification.*

regular and adaptive meshes generated with our simplification algorithm. The table shows the statistics computed on a set of meshes with the same number of faces of the same meshes adaptively refined (Table 2). The uniformly refined meshes have greater homeometry and regularity that the corresponding adaptive meshes but an higher Haursdorff distance from the original mesh. This is due to the loss of features that occurs when a limited number of faces is available and all faces must have the same size. This does not happen in simple shapes, such as those represented in Moai or Pensatore datasets, but it is extremely pronounced in more complex ones, for example in Bunny and Rampart meshes.

5. Conclusions

We have presented an algorithm for quad mesh simplification that is faster and simpler than [TPC*10], while producing meshes with higher regularity and similar homeometry. We have shown that four local operators are sufficient to progressively simplify a quad mesh.

Fitmaps have been integrated in the simplification framework, leading to an effective algorithm for adaptive simplification. At the best of our knowledge this is the first algorithm that produces adaptive quad meshes. Adaptivity on quad meshes always implies an increase in the number of irregular vertices, since they are required in transition zones between quads of different sizes. In our experiments, we measure a 3% average increase in extraordinary vertices for adaptive meshes, in comparison to non-adaptive ones with the same number of quads.

Our methods still lack alignment to feature lines and preservation of sharp features. As future work, we plan to use different heuristics for the selection of local operators and a different smoothing algorithm that is able to use a cross field defined on the mesh to produce quads aligned to it. This extension requires further research, since it increases the complexity of an already hard problem. In fact, we should optimize at the same time the homeometry, the regularity and the alignment to the cross field.

References

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3 (2009), 1–10. 2

[CCR08] CIGNONI P., CORSINI M., RANZUGLIA G.: Meshlab: an open-source 3d mesh processing system. *ERCIM News (73)* - http://meshlab.sourceforge.net/ (2008), 45–46. 5

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J.: Spectral surface quadrangulation. *ACM Trans. Graph. 25*, 3 (2006), 1057–1066. 2

[DSC09] DANIELS J., SILVA C., COHEN E.: Localized quadrilateral coarsening. *Comput. Graph. Forum 28*, 5 (2009), 1437–1444. 2

[DSSC08] DANIELS J., SILVA C., SHEPHERD J., COHEN E.: Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, 5 (2008), 1–9. 2

[HZM*08] HUANG J., ZHANG M., MA J., LIU X., KOBBELT L., BAO H.: Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph. 27*, 5 (2008), 1–9. 2

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384. 2

[LRC*02] LÜBKE D., REDDY M., COHEN J., VARSHNEY A., WATSON B., HÜBNER R.: *Level Of Detail for 3D Graphics*. Morgan Kaufmann, 2002. 2

[PPT*10] PANOZZO D., PUPPO E., TARINI M., PIETRONI N., CIGNONI P.: Automatic construction of adaptive quad-based subdivision surfaces. Submitted for publication, 2010. 1, 2, 4

[RLL*06] RAY N., LI W.-C., LÉVY B., ALLIEZ P., SHEFFER A.: Periodic global parameterization. *ACM Trans. Graph.* (2006). 2

[SDW*09] SHEPHERD J., DEWEY M., WOODBURY A., BENZLEY S., STATEN M., OWEN S.: Adaptive mesh coarsening for quadrilateral and hexahedral meshes. *Finite Elements in Analysis and Design 46*, 1-2 (2009), 17 – 32. 2

[TPC*10] TARINI M., PIETRONI N., CIGNONI P., PANOZZO D., PUPPO E.: Practical quad mesh simplification. *Computer Graphics Forum (Eurographics 2010) 29*, 2 (2010), 407–418. 1, 2, 5, 6, 7

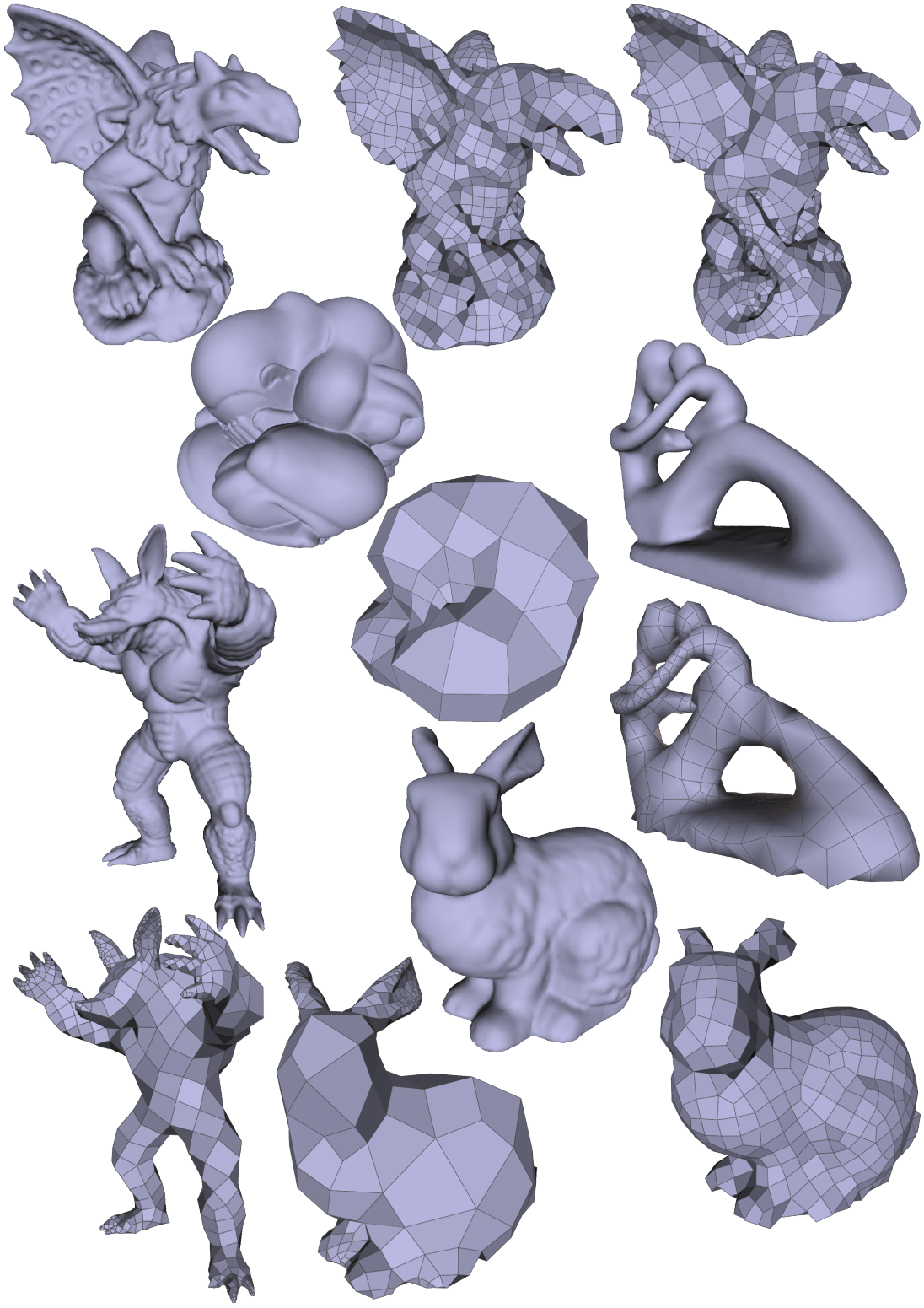


Figure 5: *Different datasets simplified with our method. The majority of them, with the exception of the gargoyle in the top center and the bunny in the bottom left, are adaptive models simplified using the Fitmaps.*