The Visual Computer

# Dynamically refining animated triangle meshes for rendering

Kolja Kähler,
Jörg Haber,
Hans-Peter Seidel

Max-Planck-Institut für Infomatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken,
Germany
E-mail: {kaehler,haberj,hpseidel}@mpi-sb.mpg.de

Published online: 14 February 2003


We present a method to dynamically apply local refinements to an irregular triangle mesh as it deforms in real time. The method increases surface smoothness in regions of high deformation by splitting triangles in a fashion similar to one or two steps of Loop subdivision. The refinement is computed for an arbitrary triangle mesh, and the subdivided triangles are simply passed to the rendering engine, leaving the mesh itself unchanged. The algorithm can thus be easily plugged into existing systems to enhance the visual appearance of animated meshes. The refinement step has very low computational overhead and is easy to implement. We demonstrate the use of the algorithm in a physics-based facial animation system.

**Key words:** Triangle meshes – Surface deformation – Adaptive refinement – Real-time rendering

*Correspondence to*: K. Kähler

## 1 Introduction

In real-time computer animation, polygon meshes are a popular surface representation due to the high throughput on current hardware. The drawback is that a polygonal surface is piecewise planar, and we therefore have to find a balance between high visual quality (more polygons) and high frame rates (less polygons).

A polygon model of a static surface can be built such that a compromise between the represented amount of detail and computational complexity is achieved. However, when the surface is animated by moving the mesh nodes, the resulting mesh doesn't adapt well to the deformation. Bends and folds can only appear at the edges of the mesh and are thus limited by its initial connectivity. This leads to the unfortunate situation that the model has to provide enough polygons to accommodate all possible deformations, even though the surface can be represented well enough with a far lower number of polygons in its undeformed state. A model should thus be dynamically refined at run-time where required.

In this paper, we present a technique to render adaptively refined versions of a triangle mesh. Refinements are computed only in those areas where the mesh is deformed. Since these refinements are used for rendering only and can be easily computed on the fly, we do *not* update the original triangle mesh. This is advantageous for two reasons: first, the application doesn't have to deal with dynamic mesh connectivity, making integration into existing systems a simple plug-in operation. Second, the refined triangles are not retained between rendered frames, so additional memory usage is kept to a minimum.

Our method has been integrated into a framework for the physics-based animation of polygonal models [6] (see Fig. 1). A spring mesh is created from the initial triangle mesh: vertices correspond to point masses, and edges correspond to springs. The spring mesh deforms in the simulation loop, and its nodes are used to update the triangle mesh. During animation, the resolution of the triangle mesh was found often to be too coarse in highly deformed regions. Increasing the resolution of the triangle mesh would result in an unacceptable increase in computation time due to the higher number of simulated springs. However, using our refinement technique, the simulation can be decoupled from the rendering: without changing the resolution of the spring mesh (which defines the precision of the simulation), we can render a smoother version of the deformed triangle mesh.

The Visual Computer (2003)
Digital Object Identifier (DOI) 10.1007/s00371-002-0185-8

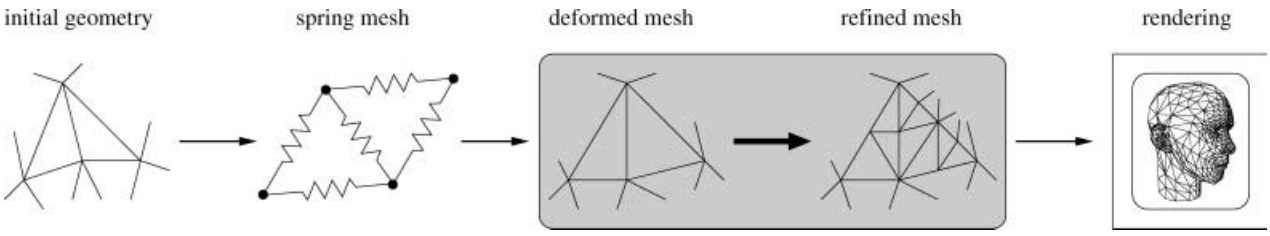



**Fig. 1.** Overview of our physics-based animation system. Without affecting the internal state of the simulation or the triangle mesh itself, refinements are computed from the deformed mesh and passed to the rendering engine

## 2 Related work

A large number of surface representations are used in computer animation. Among the most popular choices are polynomial patches, polygons, and subdivision surfaces.

Polynomial patches have long been used in modeling and animation [4, 15]. Surfaces built from such patches are defined by a relatively coarse control mesh. Animation of the surface can be achieved by deforming this mesh. The generated surface is inherently smooth; but, for complex geometry built from multiple patches, preservation of smoothness conditions across patch boundaries becomes difficult. Patches can be refined globally via knot insertion [2] or locally using hierarchical methods [5]. For real-time rendering, patches are usually tessellated using uniform [11, 16] or adaptive [14] schemes to exploit fast polygon-rendering hardware. As the surface deforms, the vertices of the tessellation have to be recomputed. For refined surfaces, the computations become more complicated and expensive.

Polygonal models are popular due to their simplicity, flexibility, and the availability of efficient graphics hardware. Adaptive refinement of arbitrary triangles meshes is a recent topic in multiresolution editing [9, 10]. These methods are very powerful, but the underlying machinery is complex and currently not applicable to real-time environments.

In physics-based animation, spring meshes are typically composed of quadrilaterals or triangles, and the mass points of the spring mesh are identified with the vertices of the rendered surface [8, 12, 20]. Adaptive refinement of such a mass-spring system is non-trivial [7]. Volino et al. propose an efficient method to smoothing polygonal geometry, which is applied to deformations caused by a mass-spring simulation [21]. Their method interpolates the interior points of arbitrary polygons, given their vertices and vertex normals. Using a regular subdivision of the initial geometry, smooth surfaces can be generated on the fly for rendering, similar to our approach. The tessellation does not adapt to surface curvature, though. In the context of facial animation, Seo et al. describe the application of level-of-detail techniques, generating not only coarser geometry but also coarser animation control for far-away viewpoints [18].

Subdivision surfaces bridge the gap between spline patches and polygon meshes in many respects, combining the easy handling of meshes with the well-defined properties of a parametric surface [13, 17]. Defined over an initial quadrilateral or triangle control mesh, an arbitrarily close approximation to a smooth limit surface can be generated by repeatedly refining the mesh using simple rules. The limit surface can either interpolate or approximate the control mesh nodes, depending on the subdivision rules. Subdivision surfaces are also suitable for use in computer animation [3]. A very regular mesh of subdivision connectivity is required, which often makes an initial remeshing step necessary when an irregular mesh is given. In general, the refinement operator is applied uniformly to a subdivision surface. Zorin et al. describe a method (*adaptive synthesis*) that selectively computes refined triangles by temporarily creating the needed parent triangles [23].

## 3 Method overview

A generic adaptive refinement algorithm employing some surface curvature criterion can be stated recursively:

```
refine(region r):
 c := curvature(r)
 if (c > threshold)
  subdivide(r, c)
  for all subregions s in r
   refine(s)
 else
  draw(r)
```

Even if the tail recursion is flattened by transformation into a loop, two cost factors remain: the curvature has to be evaluated multiple times on the initial region (albeit on smaller and smaller parts); and the changes caused by a subdivision step have to be stored in the geometry before the subregions can be examined (or temporary storage must be allocated per subregion on each level of recursion). In our approach, we minimize these costs by evaluating the curvature only once: based on the outcome, we perform up to two refinements in one step, thus eliminating the need for storing the altered geometry for further evaluation. It is directly drawn and discarded:

```
refine(region r):
 c := curvature(r)
 if (c > threshold)
  region s = subdivide1or2(r, c)
  draw(s)
 else
  draw(r)
```

In our implementation, the refinement procedure is applied to the triangle mesh just before rendering, as shown in Fig. 1. We make use of a number of adjacency relations that are defined on a triangle mesh, such as circulating through the vertices adjacent to a given vertex, finding the triangles sharing a given edge, etc. To this end, we use a data structure based on half-edges as described by Campagna et al. [1].
Each edge of the given deformed mesh is examined to decide whether it should be split into two or more parts, causing subdivision of the adjacent triangles. Using the new degrees of freedom provided by the split vertices, we compute a smoother retriangulation approximating the input mesh. For the smoothing, simple local rules are used that borrow from the subdivision idea. We don't generate any new vertices in the interior of an original triangle, thus avoiding evaluation of new interior edges and keeping the number of possible new triangulations manageable. The retriangulation is efficiently created by a table-lookup operation. The resulting triangle set is then rendered instead of the original triangle; unsplit triangles are rendered as usual. Figure 2 shows examples of split configurations.

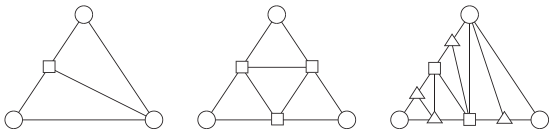


**Fig. 2.** Examples of split configurations. *From left to right*: one single edge split, three edges split once, and two edges split twice

The computed refinements are not reflected in the input mesh, they are computed dynamically for each frame and discarded after rendering. Thus, undoing refinements is not necessary, and the input mesh remains unaltered. We also do not retain any information about splits between frames.

## 4 The algorithm

Our method creates subtriangles by splitting triangle edges once into two parts or twice into four parts. We start by iterating over all edges of the input mesh, deciding whether to split them once or twice. Since the splitting of each edge is carried out in one single pass, there is no recursion involved. In a second pass, the retriangulation of each triangle is obtained from the split configuration along its edges.

### *4.1 Splitting criterion*

We assume that the quality of the triangulation of the undeformed mesh is good enough for the intended application. Therefore, we only want to split an edge if the curvature of the surrounding mesh region has increased during mesh deformation. As a simple and efficient test, we use the dot product between the vertex normals at both ends of an edge. If this scalar value drops below the value that has been precomputed for the undeformed geometry, there is more "bending" and the edge is marked for splitting once or twice, depending on the difference of the dot products.
This criterion only uses the vertex normals of the existing nodes in the mesh. More complex criteria can

be used as well, such as measuring discrete curvature on the mesh [19]. The vertex normal dot product has proven to be sensitive to the kind of deformations that occur in our application. Additionally, it has the advantage of extremely low evaluation cost, provided that vertex normals have been computed before.

### 4.2 Vertex smoothing

After having determined the split configuration for each triangle, the vertex positions of the resulting subtriangles are computed. We want to make sure that this operation has the following characteristics:

- computationally cheap;
- only changes the surface locally;
- results in a close approximation of the input mesh; and
- handles mesh boundaries correctly.

Our approach to selective refinement is inspired by Loop subdivision [13] in the variant proposed by Warren [22]. The Loop subdivision scheme also applies mid-edge splitting, and computing the refined surface only requires quick averaging of old and new vertices with their immediate neighbors. We'd like to point out that our method doesn't produce surfaces with any particular degree of continuity, but just a smoother-looking approximation of the original.

Vertex positions corresponding to the first (i.e. mid-edge) split of the triangle edges are calculated similarly to the Loop scheme by weighted averaging. Figure 3 shows the vertices that take part in these computations and the associated weights. Since we don't want to change the input mesh, the new positions for the original vertices are temporarily buffered.

For triangle edges that have been split twice, the vertex positions corresponding to the second-level splits are obtained in a similar fashion (see Fig. 4). Here, we have to use the previously computed positions of the first-level split vertices. Since generally not all edges of a triangle are split, some of these vertices may not have been computed before. In this case, we simply take the mid-point of the respective original edge.

Furthermore, the original mesh vertices are not smoothed again for second-level splits, contrary to the proper Loop subdivision scheme. In this way, we avoid complicated updates involving adjacent triangles and keep these vertices closer to their original locations.

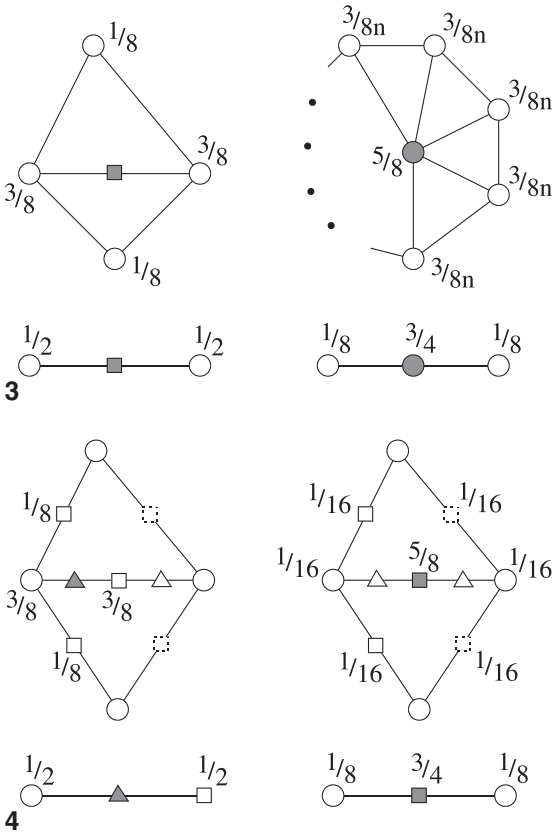
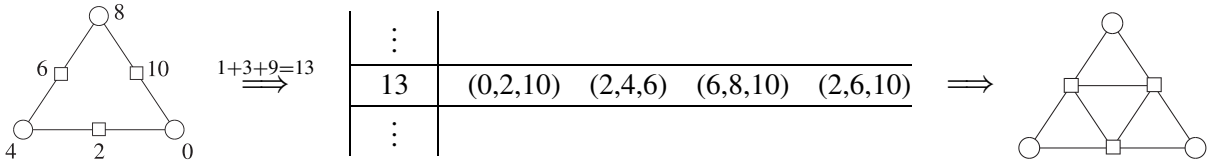



**Fig. 3.** *Left*: computing the position of a first-level split vertex (■). *Right*: displacing a vertex in the input mesh (●). The *top row* shows the weights for an interior vertex; the *bottom row* shows the boundary case

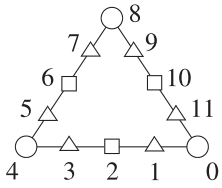
**Fig. 4.** *Left*: computing the position of a second-level split vertex (▲) from original mesh vertices (◯) and first-level split vertices (□). *Right*: displacing a first-level split vertex (■). Non-existent first-level vertices (*stippled* □) are approximated by linear interpolation

### 4.3 Generating subtriangles

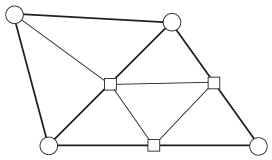
Once the new vertex positions have been computed, subtriangles are created by connecting these points and then passed to the rendering engine. To speed up the retriangulation step, we use a lookup table that has one entry per split configuration. Each entry contains a sequence of vertex indices, which rep-




**5**




**6**



**7**

**Fig. 5.** A triangle is split and retriangulated using a lookup table

**Fig. 6.** Numbering of original (◯) and split vertices (□, △), starting at the first vertex of edge 0 in a triangle

**Fig. 7.** Adjacent triangles are split consistently to avoid T-junctions

resents a valid tessellation of the original triangle. The points in each triangle are indexed according to Fig. 6. If $s_i \in \{0, 1, 2\}$ denotes the number of splits that have been applied to edge $i \in \{0, 1, 2\}$ of the current triangle, the index into the table is computed from the ternary digits $s_i$ as $s_0 + 3s_1 + 9s_2$, yielding 27 possible combinations. Figure 5 illustrates the table-lookup mechanism.

Each input triangle can be split into a maximum of ten subtriangles. No cracks appear in the generated mesh, since adjacent triangles have a common edge and thus share the split configuration along this edge (see Fig. 7).

### *4.4 Time-coherent splitting*

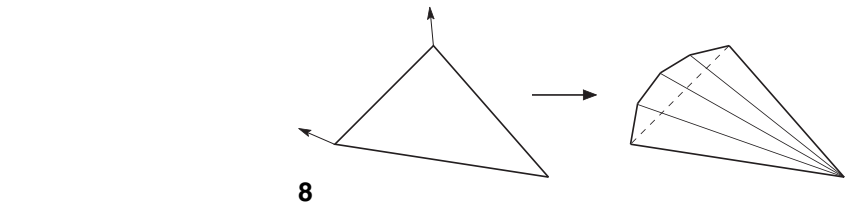
For proper shading and texturing, the vertex normal and texture coordinates of a new vertex are interpolated linearly from the neighboring vertices along the edge. The neighboring vertices are either original mesh vertices or previously created split vertices. Due to the nature of intensity-value interpolation in Gouraud shading, the retriangulation pattern in the lookup table does not affect the rendered output. For flat shading, however, triangle normals have to be computed for the generated subtriangles. If the triangulation is chosen only on the basis of the triangle's current split configuration, shading artifacts may appear: a split that is introduced from one frame to the next may lead to a completely different triangulation, causing abrupt changes in the surface normals of the subtriangles. This can be alleviated by taking the history of refinements on a triangle into account. If an edge is split further than in the previous frame, a retriangulation is generated that is equivalent to refining that previous triangulation pattern.

Though we can't avoid maintaining some sort of history, we still don't have to store split triangles between frames: it is feasible to enumerate all *sequences* of splits that can be applied to a triangle, from the zero-length sequence containing no splits at all to the sequences of length six containing two splits on each edge in every possible order. We can construct a table of 271 entries, where each line corresponds to one of these sequences. Each table entry is automatically constructed by an algorithm that splits and subdivides a triangle following the corresponding sequence.

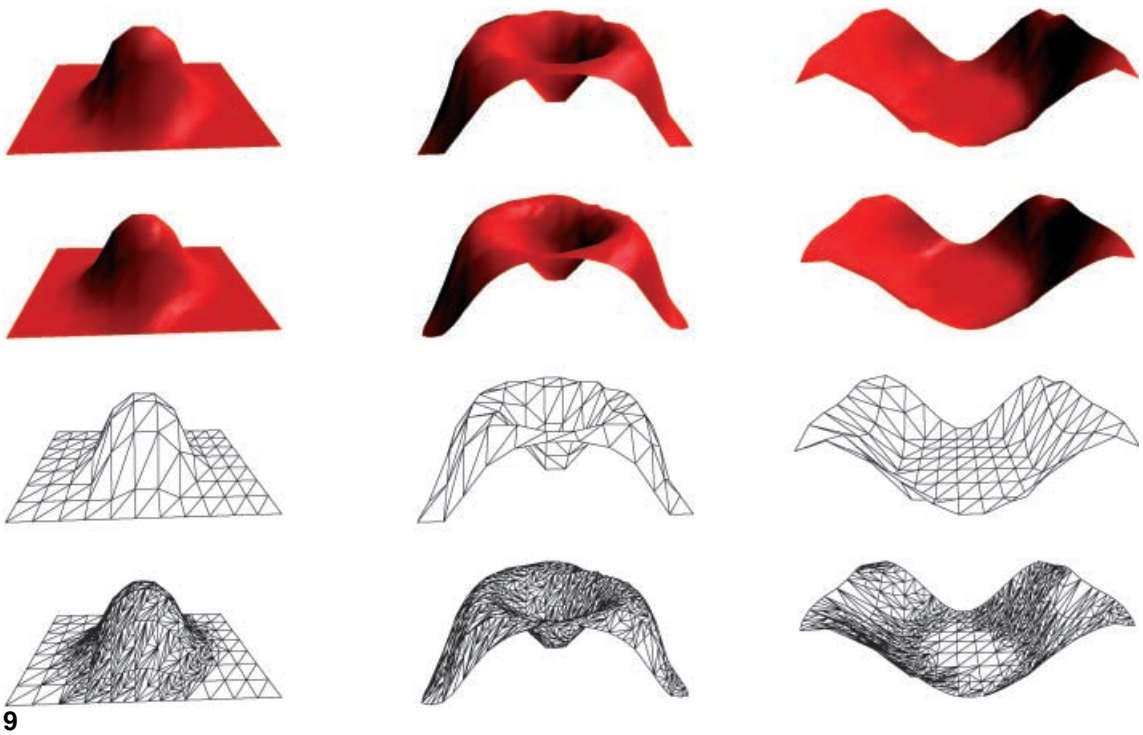
In this way, if the history of applied splits is stored along with the mesh, one can generate time-coherent retriangulations. However, splits can only be taken back in reverse order, otherwise artifacts may again appear. Additional overhead is induced by the more complicated maintainance of data structures and the bigger lookup table (deteriorated data locality). In practice, we usually avoid the overhead of time-coherent splitting, since flat shading is rarely used in our applications.

## 5 Results

In our main application, the animation system, there was no noticeable difference in frame rate when run-



8



9

**Fig. 8.** Thin triangles that are generated during adaptive refinement are aligned to the direction of minimum curvature

**Fig. 9.** A triangulated plane, undergoing a strong sine-shaped deformation. *First* and *third row*: mesh rendered as-is; *second* and *fourth row*: dynamically refined mesh. Ridges are smoothed out in the curved areas of the refined mesh. The Gouraud-shaded model also exhibits less shading artifacts and crisper highlights

ning with or without dynamic refinement. This was to be expected in a simulation context, where the computational load is mainly caused by the evaluation of the physics model and not by the rendering stage. Figure 10 shows a detail of the deformed mouth region during animation. The rendered mesh is visibly smoothed, reducing shading artifacts and improving the silhouette of the opened mouth. The achieved frame rate of 60 fps does not differ noticably between static and refined rendering. Obviously, the increased rendering time per triangle will lead to a significant drop in the frame rate for applications where rendering makes up the largest part of the overall computation time, such as the artificial example shown in Fig. 9. In our animation system, increased visual quality can be gained at nearly no cost.

In Table 1, static and dynamic mesh rendering are compared for the two test cases depicted in Figs. 9 and 10. The first column lists the number of triangles of the static mesh; in the second column, the maximum number of triangles generated by

**Table 1.** Comparison of triangle rendering performance with and without dynamic refinement. See text for detailed explanation

| | #tri static | max. #tri dyn. | $t$ static (ms) | $t$ dyn. (ms) | $c_{\mathrm{rt}}$ |
|---|---|---|---|---|---|
| face | 2030 | 3120 | 1.3 | 3.5 | 1.75 |
| plane | 154 | 1328 | 0.09 | 2.0 | 2.58 |

the dynamic refinement during animation is shown. Columns three and four compare the time spent in the rendering code, with and without refinement of the triangles to the given maximum. The last column shows the factor $c_{\mathrm{rt}}$ by which the rendering time per triangle increases with dynamic refinement switched on. All timings have been measured on a 1.7 GHz Pentium 4 PC with a GeForce3 graphics board.

We found that the explicit implementation of one to two refinement levels paid off, because there is no overhead for recursion and maintaining dynamic data structures. This will of course only hold under the assumption of a sufficiently tessellated undeformed mesh. Though the method could be extended to more than two splits per edge, our experiments have shown that this level of refinement is sufficient for the moderate deformations our initial model experiences.

When looking at the refined triangle meshes as shown in Fig. 10, one clearly notices many long and thin triangles. Usually, this is an indication of a badly generated triangle mesh. Here, however, the thin triangles are exactly what we want. Figure 8 shows that the automatically generated subtriangles are aligned to the direction of minimum curvature, thus mimicking the alignment of folds on real skin.

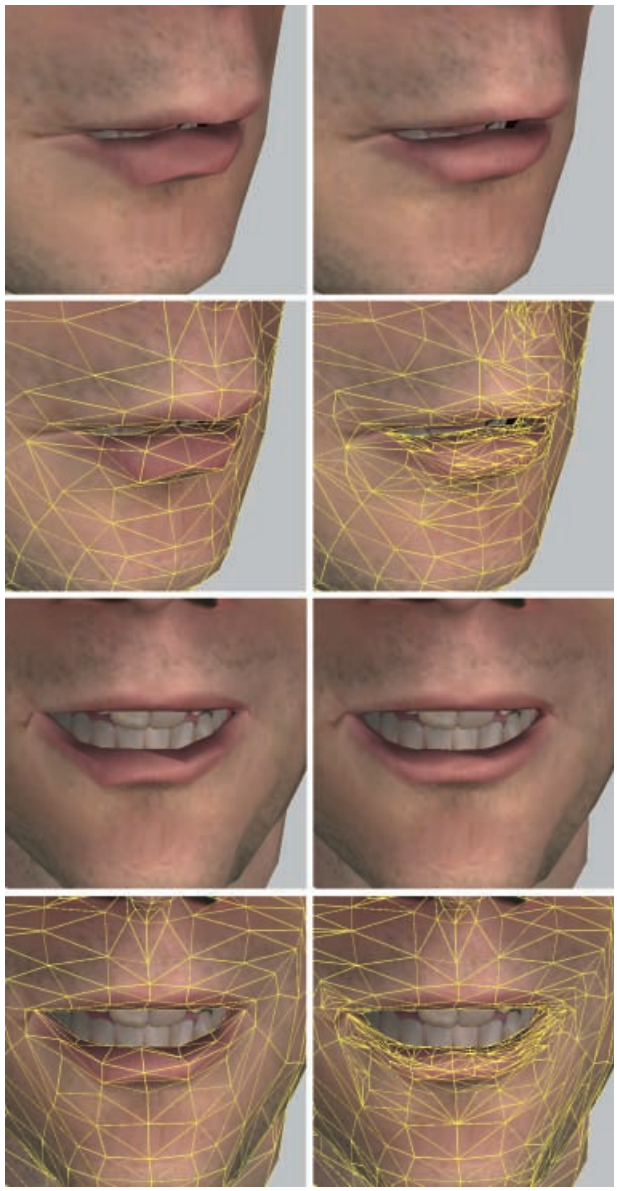


**Fig. 10.** Snapshots from animation of a face mesh. *Left*: static mesh (2030 triangles). *Right*: dynamically refined mesh. The animation runs at approximately 60 fps on an 1.7 GHz PC in both cases

## 6 Future work

We would like to extend our method in several ways. Visual quality can be further improved by refining the mesh along silhouette edges. To achieve this, appropriate splitting criteria have to be developed. Also, it would be interesting to investigate the effects of other smoothing schemes, since we currently don't interpolate, but only approximate, the input surface. If the input geometry has been produced from a finer mesh, one could do even better than smoothing: detail information can be stored locally and used to place generated split vertices on the surface, as has been exercised in multiresolution editing.

Finally, the rendering performance can be improved by generating retriangulations that can be encoded as triangle strips and/or triangle fans.

## References


1. Campagna S, Kobbelt L, Seidel H-P (1998) Directed edges: a scalable representation for triangle meshes. J Graph Tools 3(4):1–11
2. de Boor C (1978) A practical guide to splines. Springer, Berlin Heidelberg New York
3. DeRose T, Kass M, Truong T (1998) Subdivision surfaces in character animation. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques. ACM Press, New York
4. Farin G (1993) Curves and surfaces for computer aided geometric design. Academic Press, San Diego, Calif.
5. Forsey DR, Bartels RH Hierarchical B-spline refinement. In: Proceedings of the 15th annual conference on computer graphics and interactive techniques. ACM Press, New York
6. Haber J, Kähler K, Albrecht I, Yamauchi H, Seidel H-P (2001) Face to face: from real humans to realistic facial animation. In: Proceedings of 3rd Israel–Korea binational conference on geometrical modeling and computer graphics, Seoul, Korea, 11–12 October 2001
7. Hutchinson D, Preston M, Hewitt T (1996) Adaptive refinement for mass-spring simulation. In: Boulic R, Hegron G (eds) Seventh international workshop on animation and simulation. Springer, Berlin Heidelberg New York
8. Kähler K, Haber J, Seidel H-P (2001) Geometry-based muscle modeling for facial animation. In: Proceedings, graphics interface. National Research Council of Canada, Ottawa
9. Kobbelt L, Bareuther T, Seidel H-P (2000) Multiresolution shape deformations for meshes with dynamic vertex connectivity. Comput Graph Forum 19(3):249–260
10. Kobbelt L, Campagna S, Vorsatz J, Seidel H-P (1998) Interactive multi-resolution modeling on arbitrary meshes. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques. ACM Press, New York
11. Kumar S, Manocha D, Lastra A (1995) Interactive display of large-scale NURBS models. In: 1995 Symposium on interactive 3D graphics 1995, Monterey, Calif., 9–12 April 1995. ACM Press, New York
12. Lee Y, Terzopoulos D, Waters K (1995) Realistic modeling for facial animations. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques. ACM Press, New York
13. Loop CT (1987) Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics
14. Peterson JW (1994) Tessellation of NURB surfaces. In: Heckbert P (ed) Graphics gems IV. Academic Press, Boston
15. Piegl L, Tiller W (1997) The NURBS book, 2nd edn. Springer, Berlin Heidelberg New York
16. Rockwood A, Heaton K, Davis T (1989) Real-time rendering of trimmed surfaces. In: Proceedings of the 16th annual conference on computer graphics and interactive techniques. ACM Press, New York
17. Schweitzer JE (1996) Analysis and application of subdivision surfaces. PhD thesis, University of Washington
18. Seo H, Magnenat-Thalmann N (2000) LoD management on animating face models. In: Feiner S, Thalmann D (eds) Proceedings IEEE virtual reality 2000, New Brunswick, N.J., 18–22 March 2000. IEEE Computer Society, Los Alamitos, Calif.
19. Taubin G (1995) Estimating the tensor of curvature of a surface from a polyhedral. In: Proceedings international conference on computer vision. IEEE Computer Society, Los Alamitos, Calif.
20. Van Gelder A (1998) Approximate simulation of elastic membranes by triangulated spring meshes. J Graph Tools 3(2):21–41
21. Volino P, Magnenat-Thalmann N (1998) The SPHERIGON: a simple polygon patch for smoothing quickly your polygonal meshes. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques. ACM Press, New York
22. Warren J (2001) Subdivision methods for geometric design. Morgan Kaufmann Publishers. Preprint available at http://www.cs.rice.edu/∼jwarren/papers/book.ps.gz.
23. Zorin D, Schröder P, Sweldens W (1997) Interactive multiresolution mesh editing. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques. ACM Press, New York


Photographs of the authors and their biographies are given on the next page.

KOLJA KÄHLER graduated in 1996 with a master's in computer science from the Technische Universität Berlin, Germany. He has then been employed in the computer graphics industry, working on virtual reality applications, and also gained hands-on experience in the implementation of real-time character animation in a game development context. In 1999, he joined the graphics group at the Max-Planck-Institut für Informatik in Saarbrücken, Germany, where he is currently pursuing his PhD. His research interests focus on the generation of animated face models from scan data, head growth simulation using anthropometric information, modeling facial muscles, and physics-based simulation of skin deformations.

JÖRG HABER is a senior researcher at the Max-Planck-Institut für Informatik in Saarbrücken, Germany. He received his master's (1994) and PhD (1999) degrees in mathematics from the Technische Universität München, Germany. During the last seven years, he did research in various fields of computer graphics and image processing, including global illumination and real-time rendering techniques, scattered data approximation, and lossy image compression. For the last two years, his major research interests concentrate on modeling, animation, and rendering of human faces.

HANS-PETER SEIDEL is the scientific director and chair of the computer graphics group at the Max-Planck-Institut für Informatik and a professor of computer science at Saarland University, Saarbrücken, Germany. He is an adjunct professor of computer science at the University of Erlangen, Germany, and at the University of Waterloo, Canada.

Seidel's current research interests include computer graphics, geometric modeling, freeform curves and surfaces, surface reconstruction, efficient polygonal meshes, mesh reduction, multiresolution modeling, image synthesis, global illumination computations, image-based and hardware-accelerated rendering, facial simulation and animation, visualization of complex medical and engineering data, 3D image analysis and synthesis, and foundations of virtual reality. He has published some 150 technical papers in the field and has lectured widely on these topics. He has regularly served on various editorial boards and on the technical program committees of some of the leading computer graphics conferences such as ACM SIGGRAPH, Eurographics, Graphics Interface, and Pacific Graphics. He has received various grants from the German National Science Foundation (DFG), the German Federal Government (BMBF), the European Community (EU), NATO, and the German–Israel Foundation (GIF), among others.