Fast Geometrical Wrinkles on Animated Surfaces.

Pascal VOLINO, Nadia MAGNENAT THALMANN

MIRALab, University of Geneva
CH-1211, Switzerland
Web: `http://miralabwww.unige.ch`
Email: `[pascal|thalmann]@cui.unige.ch`
Phone: `-41 (22) 705 77 63` Fax: `-41 (22) 705 77 80`

ABSTRACT

Fast and interactive animation of polygon based surface models such as cloth and skin usually require to perform the animation on a rough surface description containing as few polygons as possible.. Fine wrinkle patterns usually disappear in the process, as their deformation scale becomes incompatible with the size of the mesh elements. We propose a fast geometrical wrinkling algorithm which can be implemented on top of any rough surface deformation model, and which modulates the amplitude of a predefined wrinkle heightmap in order to simulate metric surface conservation.

A wrinkle pattern is initially applied on the animated surface mesh. The native edge length of the mesh is used to compute dynamically the amplitude of the wrinkles as the mesh is deformed using a fast and robust geometric model. Several wrinkle patterns can be combined to simulate complex deformations. The presented implementation deforms an interpolated mesh with adaptive refinement to display the wrinkles.

Keywords: wrinkles, surface animation, cloth simulation, skin deformation, smoothing, interactive display.

1. INTRODUCTION

As realistic computation times limit mechanical cloth simulations to limit the surface mesh to a few thousands of triangles, it becomes quite difficult to imagine how these techniques would allow to reproduce the wrinkles of usual garments, as each wrinkle bending would require several rows of triangles to be reproduced in an acceptable way.

We are indeed vitally concerned with this issue. Intensive research is carried out on cloth simulation, where wrinkling is essential to the visual realism of the garments generated. Our recent work on interactive clothing systems emphasizes the need for roughly discretized cloth objects that retain this realism in fast animation models. In the meantime, we are investigating skin aging effects, and wrinkle simulation on human faces requires customized techniques that can take skin deformation into account.

On one hand, reducing the number of triangles of our polygonal surfaces is a necessity whenever complexity is critical, such as simulation computation and rendering time, memory requirements,... On the other hand, dealing with rough meshes results in visual inaccuracies that we would like to avoid: The surface mesh is unable to represent shape details smaller than the size of the triangles themselves.

As a follow-up to the cloth applications presented in [VOL 95] and skin simulation developed in [WUY 97], we propose an algorithm which generates wrinkles as a layer built on a base skeleton mesh that can be animated by whatever fast technique. This algorithm dynamically generates wrinkles by modulating the amplitude of a predefined heightmap pattern, according to simple geometrical laws that reproduce how the wrinkle amplitude evolves locally upon surface compression. This algorithm only relies on the evolution of the mesh edge lengths from their native length. The amplitude is computed on each mesh edge, and then interpolated on all the surface. The wrinkles can be dynamically rendered using various techniques, such as bump-mapping [BLI 78] of the heightfield texture modulated by the local wrinkle amplitude. We propose an implementation based on adaptive mesh subdivision and interpolation.

2. WRINKLE AMPLITUDE

Wrinkles appear on a surface when, rather than being compressed with metric elasticity deformation, the surface bends to absorb the extra area. This phenomenon usually occurs with thin surfaces that have very loose curvature elasticity, such as paper, rubber or skin. While a mechanical simulation could compute wrinkle patterns at the expense of heavy and slow computation, our model makes the assumption that the wrinkle pattern is constant. This assumption is quite true for many garment and skin applications. A very simple geometrical model simulates the local wrinkle amplitude using simplified mathematical expressions.

2.1. A Geometrical Amplitude-Elongation Model

The aim of the geometrical model is to find out the wrinkle amplitude evolution during material compression or elongation. In a polygonal mesh, this elongation is typically measured by the elongation of the mesh edges from their native length, which usually corresponds to their rest length in a mechanical model. While the exact 2D metric deformation state may be obtained for a mesh triangle using the Mohr circle or the “stain rosette” formula as performed in [VOL 95] from the length of its edges, we prefer using a much simpler approach which computes the 1D metric deformation directly from the individual edges.

The most direct measure of the deformation of a mesh is found by comparing the current length of the edges to their initial, or rest length. An edge contribution **ti** to thewrinkling amplitude is computed for each edge **i** of the mesh, of initial length **Li**, depending on the current length **li** of this edge.

The edge contribution **ti** is a normalized function defined by the current edge length variation. Typically, the wrinkle amplitude varies from maximal amplitude (normalized to **1**) for maximal compression (**li = 0**) to null for high elongation. Negative values mean that stretching tend to make the wrinkles disappear. We model it by building the following simple mathematical expression relating these properties in a simple and continuous way:

$$\mathbf{ti} = 2\,\lambda\left(\frac{\mathbf{1}}{\mathbf{1}+\left(\mathbf{li}/\mathbf{Li}\right)^{2}}-\mathbf{1}\right)+\mathbf{1} \qquad (1)$$

The **ti** curve depicted in Fig.1 illustrates how wrinkling amplitude evolves with surface elongation. Multiplied by the edge initial length **Li**, it captures the effect of surface length conservation along an edge as its current mesh length changes. From a maximal amplitude of **1** corresponding to maximal compression, it decreases to **0** and then to negative values, which has the effect of eliminating wrinkles when the surface is highly stretched, as shown in Fig.1.

The λ parameter controls the degree of edge elongation that is required for the surface to produce wrinkles. With a value above **1**, some compression is required to produce wrinkles, whereas below **1**, the undeformed surface is already wrinkled.

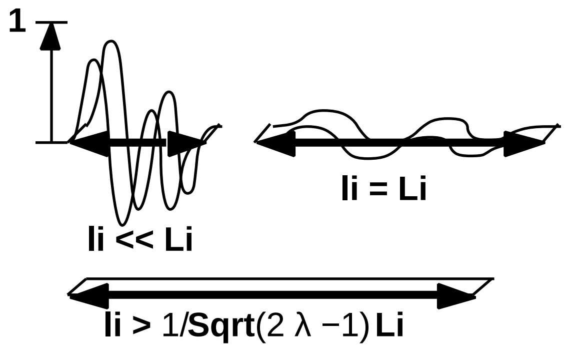


Fig.1: The evolution of the wrinkles and edge contribution with respect to the current edge length.

Obviously, wrinkles do not all react in the same way to deformations, varying according to their shape and their alignment with the directions of elongation. A wrinkle shape coefficient **mi** is introduced, which controls how the wrinkle pattern reacts to the elongation of edge **i**. As shown in Fig.3, the more the wrinkle shape orientation runs orthogonal to the direction of an edge, the more it will react to variation in that edge's length. The shape coefficient **mi** is modeled by the following simple mathematical expression:

$$\mathbf{mi} = \mathbf{1} - \frac{\mathbf{1}}{\mathbf{1}+\nu\ \mathbf{Ai}} \qquad (2)$$

Ai is a measurement of the wrinkle evolution amount along the considered edge. We could have integrated the heightfield map gradient over the mesh triangles adjacent to the edge to compute this shape function. A simple 1-dimensional discrete integration along the edge, however, has proved to suffice for this evaluation. Given a discretization of

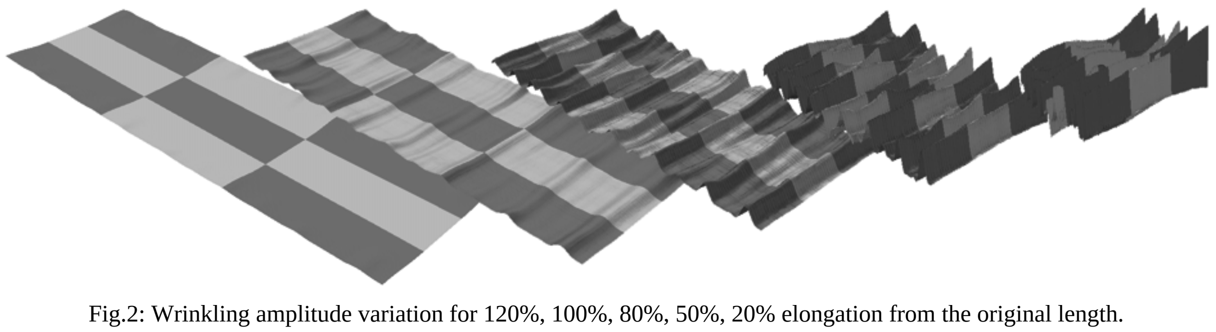


Fig.2: Wrinkling amplitude variation for 120%, 100%, 80%, 50%, 20% elongation from the original length.

the edge into **n** segments, **Ai** is **n** times the square of the wrinkle texture map height differences at the segment extremities, summed over all segments. The ν parameter is introduced to control the evolution of the shape coefficient with respect to the wrinkle complexity.

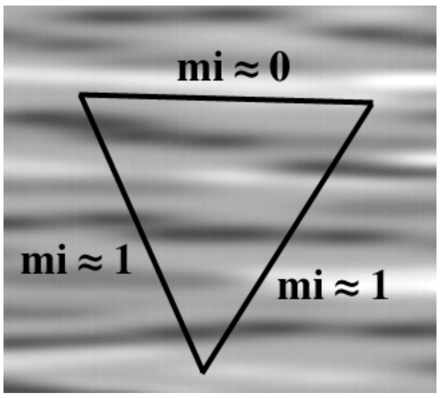


Fig.3: Shape coefficients for triangle edge with respect to a given texture map.

The shape coefficient depends only on the mesh topology and on how the wrinkle texture is applied to the mesh. It is used to render the wrinkle behavior anisotropically, according to the wrinkle orientation and the orientation of the surface deformation, as in the example shown in Fig.4. There is no need to recompute it if the texture pattern is not modified on the mesh surface.

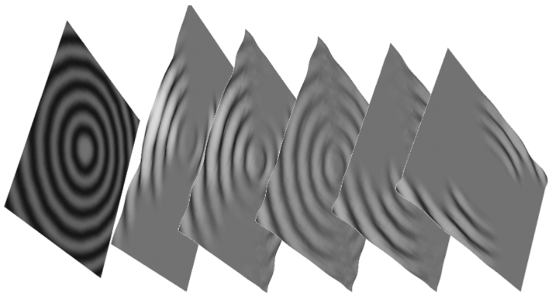


Fig.4: Wrinkle amplitude depends on wrinkling orientation with reepect to deformation direction.

Finally, we multiply the normalized product **ti mi** by the native edge length **Li** to obtain the final deformation amplitude homogeneous to the mesh element size.

2.2. Multilayer Wrinkle Patterns

One problem might result from having one single wrinkle pattern to modulate: Only that particular pattern will appear on the fabric, and it is usually only suitable for one particular deformation direction.

Complex wrinkle patterns are often a combination of wrinkles in different directions, each of them reacting differently to the current deformation state. In order to reproduce such patterns accurately, several wrinkle structures have to be computed concurrently and combined to obtain the final deformation. Our wrinkle texture has several channels, each of them containing a wrinkle structure.

Typically, we use the RGB channels of a 24-bit color image to define three wrinkle structures. The shape factor **mi** (2) has a different value for each channel, and these values are computed separately. Thus, for a surface point, the wrinkle amplitude has a different value for each wrinkle channel. These different wrinkle contributions all contribute to the global surface deformation, as shown in Fig.5.

Using this multichannel approach, we are able to design different wrinkle patterns, each of them appearing or disappearing as the surface is stretched in different directions.

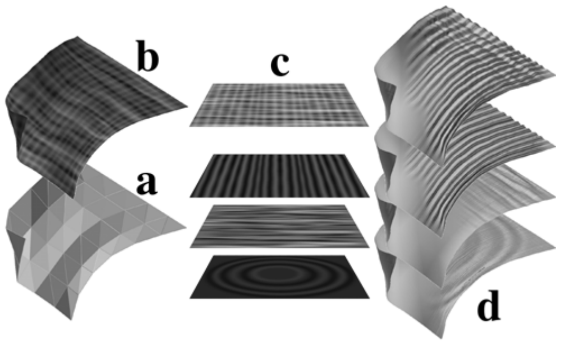


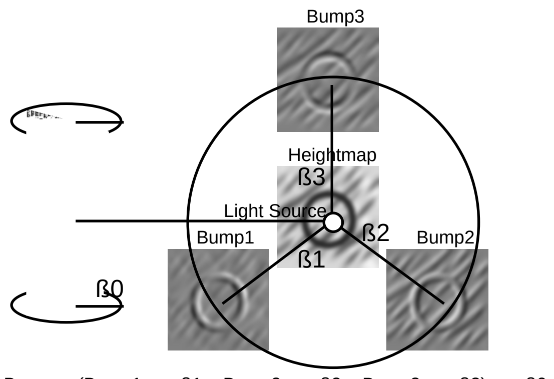
Fig.5: Multichannel wrinkle generation: (a) The initial mesh with edge deformations and (b) the interpolated and textured mesh. (c) Three wrinkle patterns as texture RGB channels, (d) wrinkle computation for each channel and blending.

2.3. Rendering Wrinkles

Having obtained the wrinkling amplitude for all edges of the mesh, the issue is to display the wrinkles from their original heightfield definition using a local scaling based on interpolated values of the amplitude. Several approaches are available, either based on shading techniques (bump mapping), either on surface deformation techniques (displacement mapping).

Introduced in [BLI 78], bump mapping techniques are based on rendering each surface point using a local surface normal “deformed” to reflect the surface orientation to simulate. From a texture heightmap, the gradient is extracted and translated into a tangent contribution to the surface normal vector. This contribution is scaled by the amplitude the heightfield has to represent. In our case, the scaling factor is naturally the computed wrinkle amplitude factor **Li ti mi** interpolated on the surface mesh triangle.

Among the various solutions suitable for rendering bump mappings, ray-tracing is the simplest, as the shading is directly computed from the surface normal at each rendered point of the surface. Hardware solutions have also been proposed [PEE 97]. Using a fast triangle-based renderer, an efficient solution is to compute the bump shading as a linear combination of textures, each of them being the precomputed heightmap gradient texture for different light angles. The combination coefficients are computed from the relative position of the light on the surface as shown in Fig.6, and of course modulated by the wrinkle amplitude.



Bump = (Bump1 cos ß1 + Bump2 cos ß2 + Bump3 cos ß3) cos ß0

Fig.6: Computing the bump shading contribution of a light source from a set of three precomputed bump samples of a texture heightmap.

3. REFINING WRINKLED MESHES

We have implemented our wrinkling process through the displacement mapping approach, using an dynamically adaptive rediscretization scheme of the initial mesh, based on its curvature, wrinkle pattern and amplitude.

As the mesh is to be rendered, the process is performed as follows:

A. *The wrinkle amplitude is computed for each mesh vertex.*

B. *The discretization of each mesh triangle is determined, using its curvature, its wrinkle pattern and amplitude.*

C. *The discretized mesh triangles are rendered, using curvature interpolation and displacement along the Phong normals with the interpolated amplitudes from the triangle vertices.*

3.1. Wrinkle Amplitude Computation

The described wrinkle amplitude computation determines the wrinkle amplitude along surface mesh edges. This is not very convenient for most interpolation. Having them computed for each vertex would be better for this purpose, and would furthermore fair the wrinkle amplitude by removing the artifacts related to particular edge directions. We compute the vertex amplitude **sa** on the vertex **Pa** by an average of its adjacent edge **I** amplitudes (1) (2):

$$\mathbf{sa} = \sum_{\mathbf{I}\ \mathbf{Pa}} \mathbf{Li\ mi\ ti} \Big/ \sum_{\mathbf{I}\ \mathbf{Pa}} \mathbf{1} \qquad (3)$$

3.2. Adaptive Mesh Rediscretization

Each mesh triangle is dynamically rediscretized in the rendering process to produce the wrinkles. The discretization rate is computed from the initial wrinkle pattern, usually during preprocessing while computing the edge wrinkle shape coefficient **mi** (2), in order to have at least four discretized elements across each significant individual wrinkle. Additional parameters such as current wrinkle amplitude and surface curvature (measured from the vertex surface normals) can dynamically be taken into account.

3.3. Curvature Interpolation using Vertex Normals

To improve the visual smoothness of the surface, we take advantage of the rediscretization to smooth the surface curvature by interpolating the intermediate rediscretized vertices and turn the mesh triangles into smooth curved patches. As shown in Fig.7, vertex positions and surface normals are used as control points similarly to the problematic solved in [BAJ 92]. In contrast to approximation schemes, this approach is local to individual triangles, each of which can be smoothed independently. However, as the smoothing has to be performed dynamically on the fly at each rendering, only simple and fast algorithms are suitable. Recursive subdivision schemes are also inappropriate as they require a fixed rediscretization scheme incompatible with the variable discretizations of our wrinkling algorithm. By dropping some continuity requirements that do not usually affect the perceived smoothness of the surface, it is possible to simplify the process considerably, reducing computation and limiting it to simple vector operations on triangle positions and normals. This scheme is extensively detailed in [VOL 98].

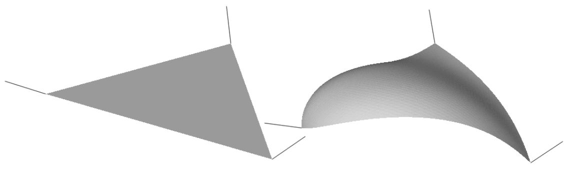


Fig.7: The initial triangle, and the interpolation surface orthogonal to the vertex normals.

Given the triangle defined by the vertices **Pa,Pb,Pc** and their respective normals **Na,Nb,Nc**, our goal is to compute an interpolated point **Q** expressed by the triangle barycentric coordinates **ra,rb,rc**. We note **P** the corresponding point of the triangle surface and **N** the corresponding Phong normal, computed by linear combination of the vertices using the barycentric coordinates, as shown in Fig.8.

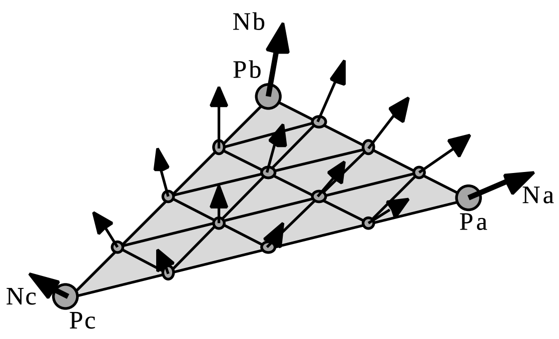


Fig.8: Interpolating Vertices and Normals.

As shown in Fig.9, a contribution surface is computed for each vertex, which describe a continuous curved surface orthogonal to the Phong normals of the triangle and passing through the corresponding vertex. For the vertex **Pa**, we compute the intermediate value **Ka** and then the surface contribution point **Qa** corresponding to the triangle point **P** and the Phong normal **N** as follows:

$$\mathbf{Ka} = \mathbf{P} + \left(\left(\mathbf{Pa} - \mathbf{P}\right) \cdot \mathbf{N}\right) \mathbf{N}$$

$$\mathbf{Qa} = \mathbf{Ka} + \frac{\left(\mathbf{Pa} - \mathbf{Ka}\right) \cdot \mathbf{Na}}{2 + \mu \left(\left(\mathbf{N} \cdot \mathbf{Na}\right) - 1\right)} \mathbf{N} \tag{4}$$

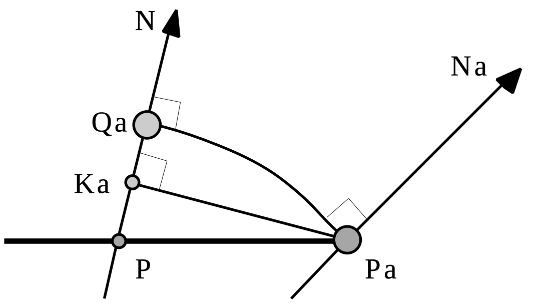


Fig.9: Computing the contribution **Qa** of the vertex **Pa** corresponding to the interpolated point **P**.

As shown in Fig.10, We finally blend the three surfaces into the final interpolation surface using normalized quadratic contributions of the corresponding barycentric coordinates (with $\mathbf{f(x) = x^2}$):

$$\mathbf{Q} = \frac{\mathbf{f(ra)\,Qa + f(rb)\,Qb + f(rc)\,Qc}}{\mathbf{f(ra) + f(rb) + f(rc)}} \tag{5}$$

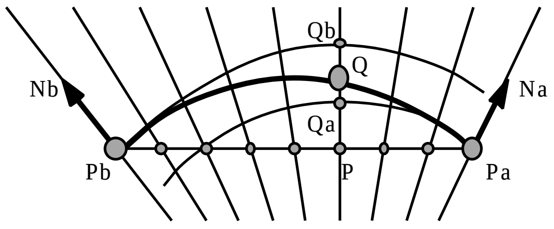


Fig. 10: Interpolation between vertex contributions.

While not ensuring strict C1 continuity because of the simple blending function, this approach however allows a very fast interpolation using any arbitrary set of intermediate vertices for subdividing a mesh triangle. The interpolation quality is very satisfying, particularly in the cases where the curvature is well distributed between the mesh triangles. This is usually the case for most modeled objects when using shading vertex normals for performing the interpolation.

3.4. Rendering Wrinkles

On the interpolated surface, wrinkles are then generated by displacing the interpolated vertices along the direction defined by the Phong normals on the mesh triangles.

As shown in Fig.12, for any point **P** of the triangle surface defined by its barycentric coordinates **ra,rb,rc**, the vertex amplitudes **sa,sb,sc** computed in formula (3) are smoothly interpolated to a value **s** using the normalized quadratic blending shape function $\mathbf{f(x) = x^2}$ shown in Fig.13 defined on the barycentric coordinates:

$$\mathbf{s} = \frac{\mathbf{f(ra)\,sa + f(rb)\,sb + f(rc)\,sc}}{\mathbf{f(ra) + f(rb) + f(rc)}} \tag{6}$$

This interpolation allows good amplitude continuity on the wrinkle amplitude between the mesh triangles.

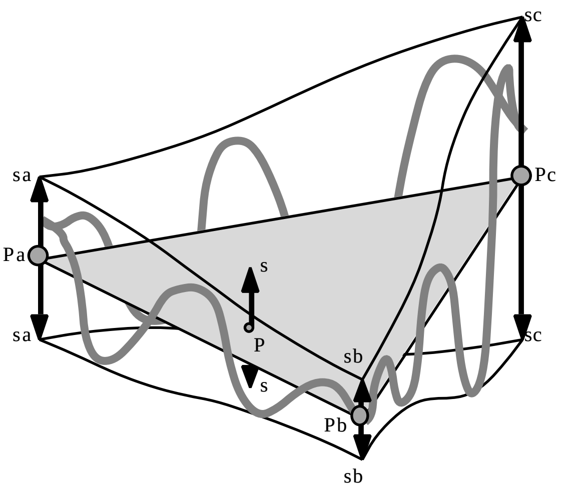


Fig.12: Wrinkle amplitude interpolation between the vertices of a triangle, and the shape function.

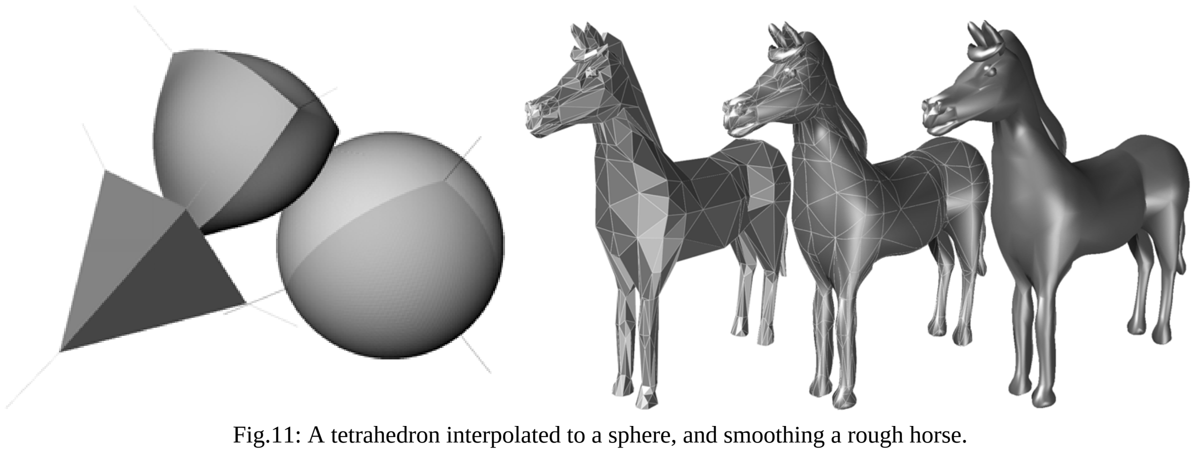


Fig.11: A tetrahedron interpolated to a sphere, and smoothing a rough horse.

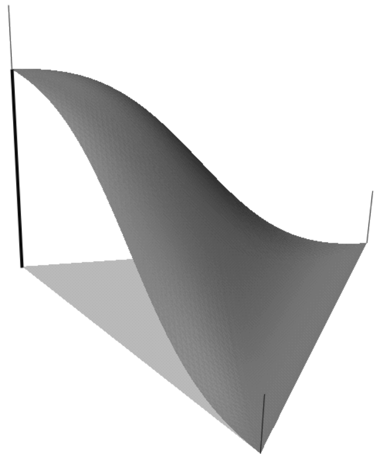


Fig.13: The blending shape function.

4. RESULTS

In a polygonal mesh model of the human face, facial expressions are simulated through mesh deformations. The pattern of facial wrinkle is defined on the face texture for creating volumic and shading features. Animation of the face elongates or compresses some regions of the mesh, automatically varying the wrinkles through our algorithm.

Integrating the interpolation algorithm into our cloth simulation and animation software has enabled us to generate cloth with considerably enhanced realism. The rough garment models we favor for fast cloth generation can now be displayed with a higher degree of realism. Again, interpolation improves the quality of the rendering.

Skin aging can be simulated by simply increasing the mesh rest length at certain locations, while for

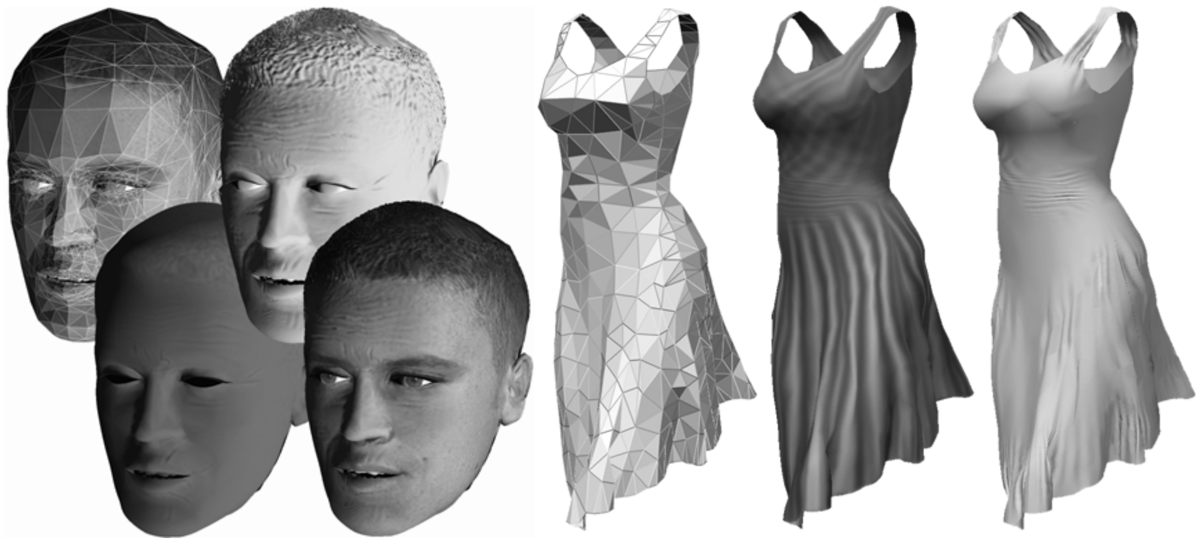


Fig.14: Facial wrinkling and garment simulation. The original mesh, the interpolated mesh with wrinkle texture displayed, the interpolated and wrinkled face without and with texture.

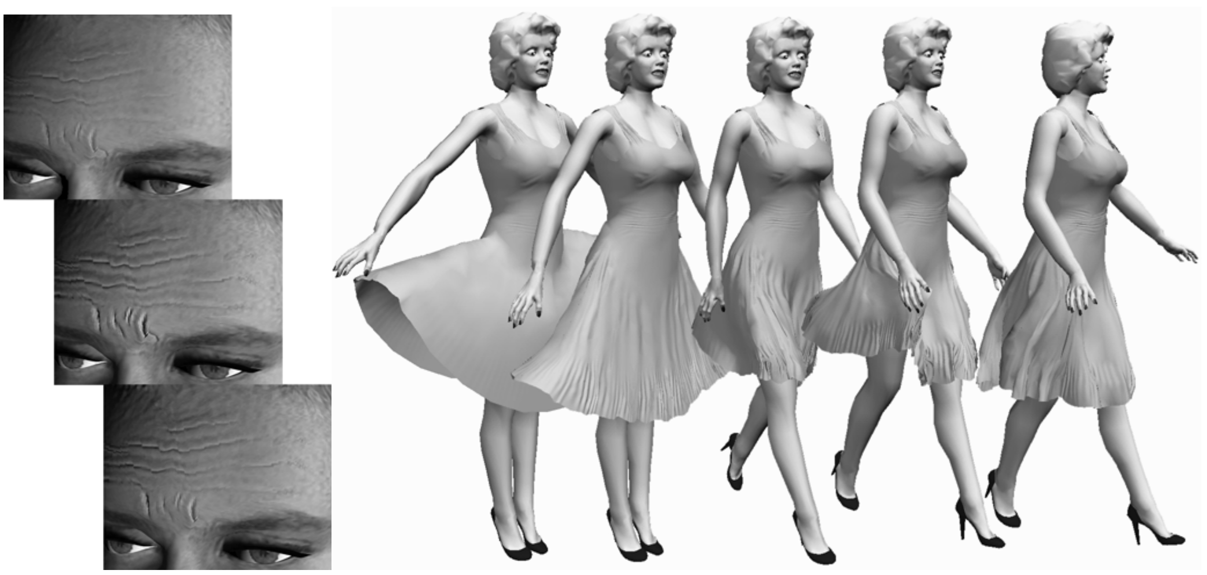


Fig.15: The wrinkles change amplitude anisotropically when the skin surface or cloth is dilated. This is a way to simulate aging. and animating a dress, dynamically wrinkled according to the mesh deformation.

cloth animation, material compression dynamically modulates wrinkling amplitude.

Moreover, the generation of wrinkles is carried out “on the fly” on the deforming polygonal mesh at display time, without the need for explicit storage of the interpolated and wrinkled structure. The wrinkles are modulated dynamically as display “postprocessing”. Hence no fundamental adaptation of our existing cloth simulation software was required. The only issues we had to deal with were related to collision detection, where reaction distance was increased to prevent interpenetrations caused by wrinkle deformations and, in the mechanical model, a significant decrease of the compression rigidity parameter, to take into account the equivalent surface rigidity after wrinkling, allowing the wrinkles to appear.

Our techniques have greatly diminished the consequences of the realism-speed compromise in our clothing system. It is now possible to use very rough triangular meshes for generating realistic and attractive garments. For instance, the cloth shown in fig.14 was used to compute the sequence in fig.15, which lasts 10 seconds. For the whole sequence, the mechanical computation itself took only around 15 minutes on a 200Mhz R5000 SGI O2 using a cloth surface mesh containing about 500 triangles, with full collision and self-collision detection. On the other hand, the displayed surface virtually contains about 10 000 triangles to render the wrinkle details.

4.3. In a nutshell...

Rough meshes are ideal for efficient object modeling and fast simulation tasks, such as those encountered in VR applications. Through our smoothing and wrinkling techniques, they are no longer incompatible with high quality rendering and visual realism.

Our operations are carried out as postprocessing at rendering time, using very simple and general algorithms. These techniques can thus be adapted in a very general manner to an extended range of computer graphics and animation applications. We have illustrated this with examples of facial wrinkle simulation and cloth animation.

As a component of our constant striving towards VR systems where a good animation frame-rate should not be synonymous with rendering of poor visual quality, we intend to integrate our algorithms into many aspects of an animation framework capable of generating realistic and well-populated virtual worlds with a minimum of computational effort. Virtual worlds to which computer programmers, designers and VR users could all contribute peacefully.

Acknowledgments

We are grateful to all the people who have contributed to this work, for the technical content as well as for the illustration design and the text reviewing.

BIBLIOGRAPHY

[BAJ 92] : **C.L. Bajaj, I. Ihm**, *“Smoothing Polyhedra using Implicit Algebraic Splines”*, Computer Graphics (SIGGRAPH proceedings 1992), 26 (2), pp 79-88, 1996.

[BLI 78] : **J.F. Blinn**, *“Simulation of Wrinkled Surfaces”*, Computer Graphics (SIGGRAPH proceedings 1978), USA, 12, pp 286-292, 1978.

[LOD 93] : **S. Lodha**, *“Filling N-sided Holes”*, Modeling in Computer Graphics, IFIP Series on Computer Graphics, Springer-Verlag, pp 319-345, 1993.

[PEE 97] : **M. Peercy, J. Airey, B. Cabral**, *“Efficient Bump-Mapping Hardware”*, Computer Graphics (SIGGRAPH proceedings 1997), Los Angeles, USA, pp 303-306, 1997.

[PHO 75] : **B.T. Phong**, *“Illumination for Computer Generated Pictures”*, Communications of the ACM, 6, pp 311-317, 1975

[SAB 86] : **M. Sabin**, *“Recursive Subdivision”*, The Mathematics of Surfaces, Clarendon Press, Oxford, England, pp 269-282, 1986

[VOL 98] : **P. Volino, N. Magnenat-Thalmann**, *“The SPHERIGON: A Simple Polygon Patch for Smoothing Quickly your Polygonal Meshes”*, Camputer Animation Proc. 98, pp 269-282, 1998.

[WUY 97] : **Y. Wu, P. Kalra, N. Magnenat-Thalmann**, *“Physically-based Wrinkle Simulation & Skin Rendering”*, Computer Animation Proc. 97, Geneva, Switzerland, pp 69-79, 1997.

[ZOR 96] : **D. Zorin, P. Schröder, W. Sweldens**, *“Interpolating Subdivision for Meshes with Arbitrary Topology”*, Computer Graphics (SIGGRAPH proceedings 1995), New Orleans, USA, pp 189-192, 1996.