Collision and Self-Collision Detection :Efficient and Robust Solutions for Highly Deformable Surfaces

Pascal Volino, Nadia Magnenat Thalmann

MIRALAB, University of Geneva

Abstract

We present an efficient algorithm for detecting self collisions, as well as some techniques for evaluating collision inside-outside orientation in a robust way. As presented in [VOL 94], we detect collisions using a hierarchical algorithm that takes advantage of curvature properties giving us full power of hierarchical algorithms for self-collision situations. Determining the collision orientation may become a complex problem dealing with complex collisions reulting from highly deformable surfaces. We use collision remnance and consistency correction for computing collision orientation in a robust way, for accurate collision response in simulations involving highly deformed and wrinkled surfaces.

Keywords : Collision detection, self-collision, geometrical regularity, automatic hierarchisation, adjacency detection, surface orientation, consistency.

1. Introduction

Mechanical simulation of soft surfaces usually require collision detection to be performed, for avoiding the simulated objects to penetrate themselves or each other. For instance, cloth simulation require avoiding cloth-to-body penetration, but also cloth-to-cloth collisions, for instance between wrinkles.

Collision detection is often very time consuming. Dealing with discretized surfaces, it consists in computing which couples of polygons are interpenetrating. The simulation algorithm then calculates the appropriate interaction response that will correct the surface deformation accordingly.

Several algorithms have been proposed for handling efficiently collision detection, some suited to parametrical surfaces ([BAR 90], [VHE 90], [DUF 92], [SNY 93]), others based on rasterization ([SHI 91]), shortest distance tracking ([LIN 93], [MOO 88]), voxelistation or octree ([LAF 91], [YAN 93]), hierarchisation ([WEB 92]).

As we intend to cope with highy deformed surfaces, like those obtained for surface wrinkling, we need an algorithm that keeps being very efficient for self-collision detection. Using some geometrical considerations based on curvature, we have successfully extended a hierarchical algorithm to get very fast self collision evaluation on discretized surface animations. [VOL 94] includes a detailed presentation of the algorithms and the resulting efficiency.

For getting appropriate collision response, penetration orientation has to be computed for the detected collisions. Quite trivial when handling surfaces that have a well-defined inside-outside orientation (bodies, closed surfaces, ...), this problem becomes ambiguous for general situations considering surfaces that do not contain any orientation information (wrinkling fabric, ...). Techniques have been developed for computing orientation in a robust way using remanance and consistency, in order to get a very robust simulation system that can handle complex situations involving wrinkling.

2. Efficient self-collision detection

Usual collision detection algorithms are rather inefficient for handling self-collisions because of adjacency : Two adjacent subareas are virtually "colliding" by contact along their common borders, and the algorithms will spend time focusing on them. For instance, dealing with polygonisations, they will consider each edge separating polygons as potential collision areas between these polygons.

We intend to speed up self-collision detection by taking advantage of adjacency, using some geometrical properties in order to skip collision detection within some areas or between adjacent areas.

2.1. Self-collisions and curvature

It is quite evident that a very smooth and regular surface will not show many self-intersections, at least at short ranges. In fact, the only causes of self-intersection are the following:

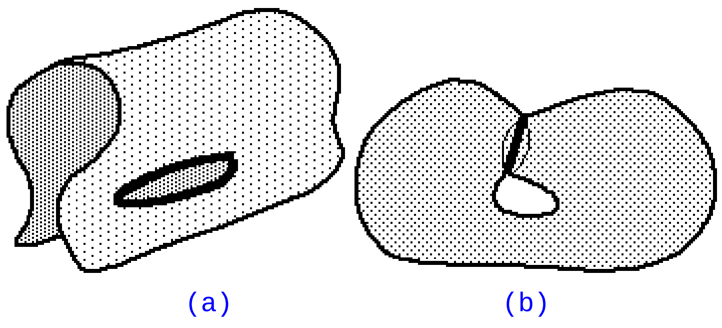


Figure 1: Self-collision occurring because of curvature (a) or contour shape (b)

(a) - The surface is curved enough for making a "loop" and hitting another part of the surface.

(b) - The contour of the surface has such a shape that a minimal fold will bring superposition and collision of the surface.

A more formal formulation of this geometrical property would be :

* Let S be a continuous surface in the Euclidean space delimited by one contour C.

if :

- There exists a vector V for which N.V > 0 at (almost) every point of S (N being the normal vector of the surface at the considered point)

and :

- The projection of C on a plane orthogonal to V along the direction of V has no self-intersections

then :

- There are no self-collisions on the surface S.

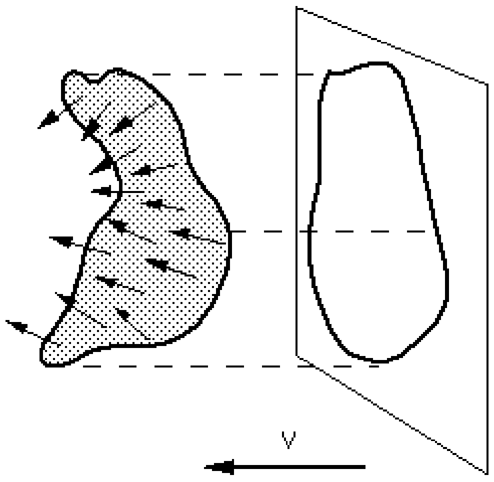


Figure 2 : Geometrical conditions for no self-collision

We take advantage of this property in two different ways, as described below:

2.1.1. Self-collision into a surface area

(I) - If can be found an area for which (a) there exists a vector that has positive dot product with the normals of every triangle of the area and if (b) the 2D projection of its contour along this direction does not intersect itself, then we need not look for self-intersections within that area.

This has great impact : We can check for normals in O(n) time (see 2.2.2), and test a 2D contour for collisions (that should contain at most O(sqrt n) elementary segments) also in an efficient way (see 2.2.3). That means that we can discard big areas from self-collision detection in O(n) time. Furthermore, normal calculation is a task that has often to be performed for many other reasons, like rendering or mechanical calculation, and the collision-specific calculation time is therefore reduced by this amount.

2.1.2. Collision between two surface areas

(II) - if can be found two areas that are adjacent (connected by at least one vertex), then if (a) there exists a vector that has positive dot product with the normals of every triangles of both areas and if (b) the 2D projection of their contours along this direction do not intersect each other, then we need not look for intersections between these two areas.

By this, we can also efficiently discard intersection tests between two adjacent subparts of our surface. Adjacency test may be performed in O(log n) time (see 2.2.1).

2.2. Efficient implementation using a hierarchical algorithm

Hierarchical algorithms are well known solutions for reducing worst-time computations to O(n log n) when dealing with a huge set of objects, such as all the polygons of a discretized surface. Still, even considering O(n) average time can be heavy computation, for example if a self-collision algorithm focuses on each edge of a polygonisation, even when the surface contain only sparse collisions.

We extend the traditional hierarchical collision detection algorithm in order to take advantage of the geometrical properties mentioned above, in order to get full efficiency of the algorithm even for self-collision detection, that is a computation time in average proportional to the number of colliding polygons.

Thus, we use respectively the (I) and (II) properties for skipping self-collisions within one area of the hierarchy, or between two areas if they are adjacent.

Three practical problems still remain :

2.2.1. Adjacency storage and evaluation in a hierarchical surface

Brute force storage of every adjacent subsurfaces around every subsurface in the hierarchy wouldn't lead to efficient storage space, nor to acceptable adjacency evaluation time.

Instead, we only store the vertices separating adjacent subsurfaces *of highest level* in the hierarchy around each subsurfaces. This computation may be performed in O(log n) time. The number of such vertices is in average constant whatever the hierarchy level of the subsurface (around 6).

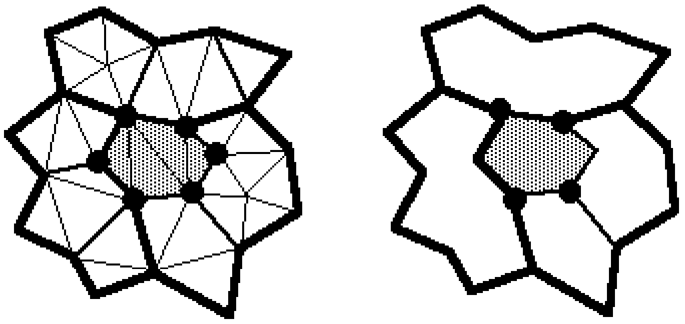


Figure 3 : Storing adjacencies: vertices to be stored around a subsurface

We can then prove that two given subsurfaces are adjacent (i.e. have at least one common vertex) if and only if there is at least a common vertex in their stored vertices. The number of them being in average constant, the evaluation time is in average constant.

2.2.2. Curvature criteria evaluation in a surface polygonalisation

We find a common direction yielding positive dot product with the normals of every polygon of a discretized surface using direction sampling : For instance, we can use a set of 14 unit vectors oriented to the vertices and face middles of a cube. For each polygon we evaluate which vectors give positive dot product with the normal, and we propagate this information up in the hierarchy tree. We then can evaluate quickly if the common direction can be found by looking at the remaining sample vectors in the hierarchy.

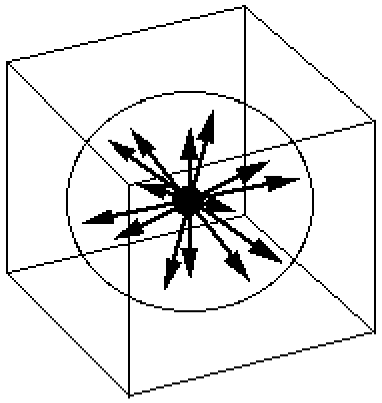


Figure4

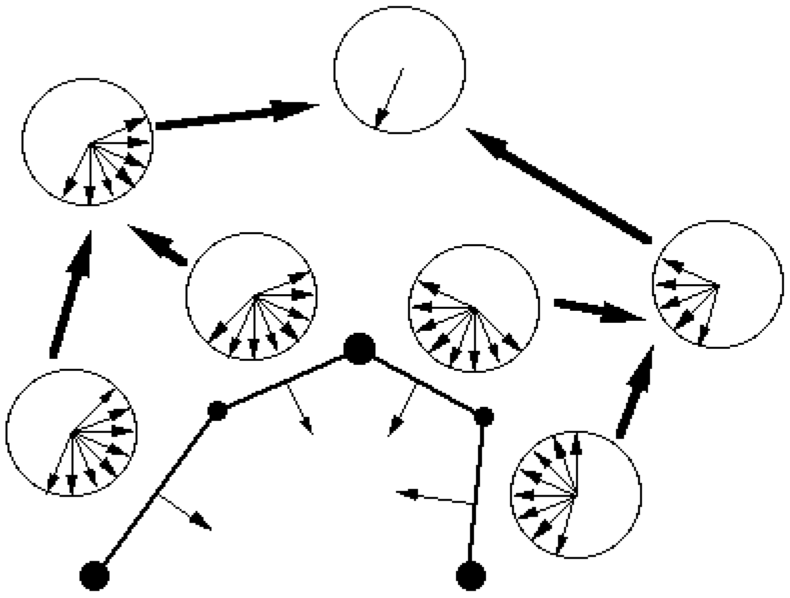
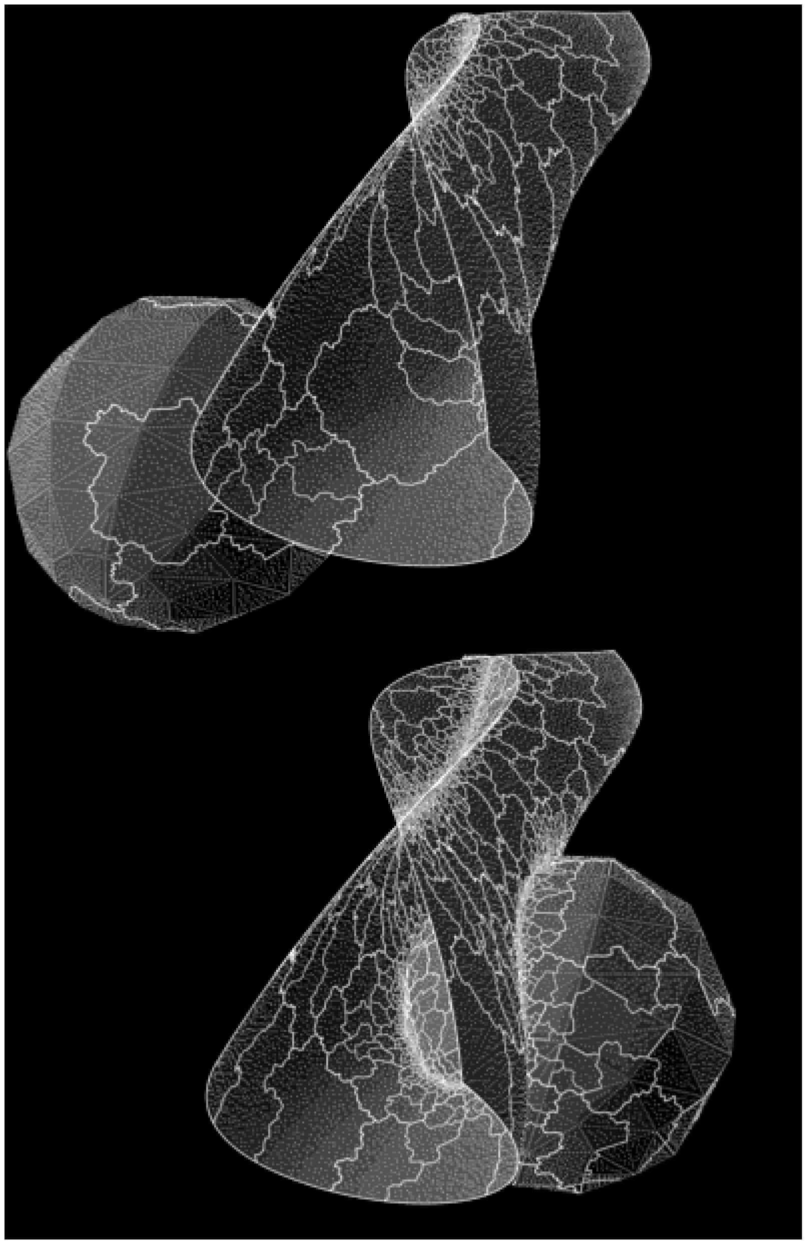


Figure5 : 12 vectors direction sampling Finding a valid direction for a surface

2.2.3. Evaluating collisions in the contour projection.

This can be performed rather efficiently by taking advantage of the already existing bounding box hierarchy. However, practical tests have shown that this criteria as almost never decisive, and can be skipped for all non-pathological collision situations involving acceptable surface shapes and hierarchisation.



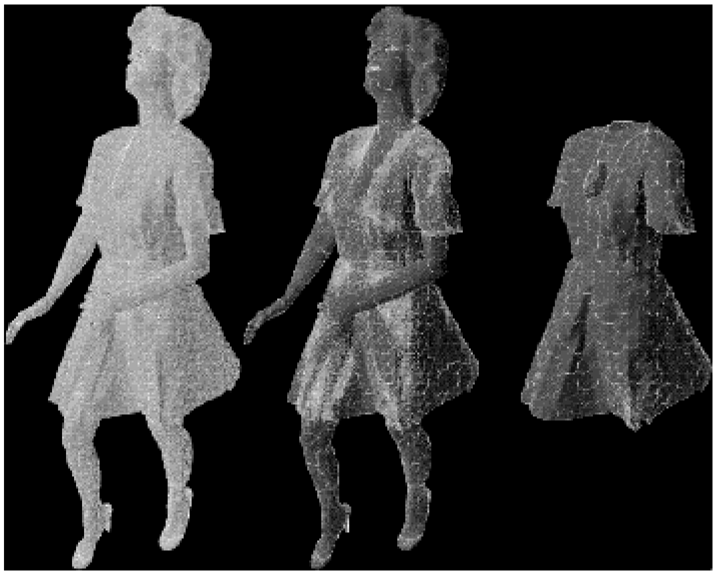


Figure 6 : Collision and self collision detection

3. Robustness and orientation consistency

Once collisions are detected, correct response has to be computed in order to provide the collision feedback. For instance, collision between surfaces animated using physically based algorithms have to yield mechanical contact interaction.

Dealing with mechanical surfaces, an important difficulty bay be to compute the correct collision orientation: As a given element may be detected to be geometrically close to a surface, shall we consider this element at the correct side of the surface (where it has always been and should remain), or at the other side (oops, it has crossed the surface, lets put it back)? Except for some closed surfaces like body skin, balls, that have an already predefined inside-outside orientation and for which the "normal" position of other objects is outside, collision situations can become very ambiguous when two deformed surfaces interpenetrate: At which side of each surface should the other surface be considered, and thus, how to compute the collision interaction orientation correctly?

In order to get a robust system for which collisions are always computed in a consistent way, several techniques have been implemented.

3.1. Collision remnance

This first technique is quite easy to implement : Once detected, a collision is stored in a structure and tracked during the animation according to the positions of the concerned objects. In-out orientation is updated according to the moves of the objects, such as when the conserned elements cross.

Any detected collision remains in the structure for a given time even if it is not actually occurring, providing extra robustness when two object parts in contact are scattered by any kind of simulation artifact.

3.2. Orientation consistency tracking

Remanance is a quite simple technique for maintaining the consistency of an initially correct situation. However, the system should also be able to recover from an initially inconsistent situation, or a situation that has become inconsistent after some severe simulation trouble.

The basic idea of this technique is to keep track of all the collision areas of the scene. Thus, we group every detected collisions that are part of the same "contact region", and we compute the "most probable" orientation for this collision region by considering statistically the individual collision orientations. Then, the common collision orientation is given for every collision.

The most important difficulty is to track efficiently the collision regions. We use an incremental process, based on the labeling of the collisions handled by the remanance mentioned above, using neighborhood walking along the colliding surfaces.

If any ambiguous and inconsistent collision situation happens, the colliding surfaces will then "choose" the most probable collision orientation, and each individual collision within the considered area will then behave in a consistent way.

Combined with remanance this orientation consistency correction gives us a very robust system, able to recover most of the inconsistencies concerning colliding non oriented surfaces.

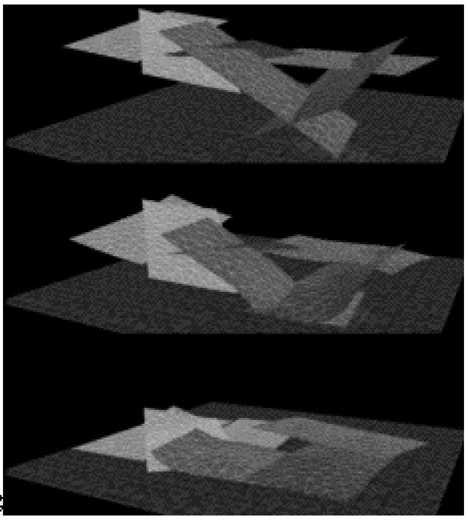


Figure 7 **Figure 8**

Consistent and inconsistent collision orientations. Consistency correction a set of intersecting falling surfaces.

3.2.1 - Implementing collision consistency correction

Collisions are of two kinds:

* **Proximities**: Elements from two different surface regions that are separated by a distance smaller than a given detection distance. They can be vertex-polygon, edge-edge, and more marginally vertex-edge and vertex-vertex proximities.

* **Interferences**: Elements from two different surface regions (an edge and a polygon) that are interpenetrating each other. They reveal that the surface regions are crossing each other.

Each element of the surface mesh data structure contains a list of the collisions in which it is involved (including remnant collisions).

A proximity is represented by a couple of elements, one from each colliding surface region, and a distance, which represents the distance of the two concerned elements. Positive distance means that the elements are at the right side of each other, and negative means they should cross for being at the right side.

Figure 9 : Different kinds of collisions.

In case surface orientation information is lacking, collision detection returns only positive distance collisions, assuming the surfaces are at the correct side of each other. However, if interferences are detected as well (i.e. edges crossing polygons), we then know that the situation is inconsistent and some collision orientations have to be reversed.

The first task is to group the detected collisions into sets characterizing collision regions. A collision region between two surfaces is the area where the two surfaces are "in contact", according to the collision detection distance. The second step is to reorient all the collisions of a given region so that they all behave in the same way, according to some statistical considerations.

The labeling process is performed by the following algorithm:

```
LabelCollision(coll,group) (
    if coll is not yet labelled (
        label coll to group
        for every col in the neighborhood of coll (
            LabelCollision(col,group)
        )
    )
)
```

```
LabelAllCollisions() (
    while exists coll not yet labelled (
        create a new group
        LabelCollision(coll,group)
    )
)
```

Two collisions are considered as neighbors if their elements are neighbors. We find all the neighbor collisions of a given collision by scanning all the collisions concerning the neighboring elements of both collision elements.

Then, for each group, we compare the surface orientation and collision the orientation of each collision in it. By numbering how many collision push in which direction of the surfaces, we consider the majority direction, and we force every collision of the group to that direction. While not being the most reliable criteria (considering collisions in the border of the group is certainly better), the majority criteria has shown to give good results for the mechanical simulations that have been tested, as in those cases only a small amount of crossings occur.

Figure 10 : The collision orientation consistency correction process.

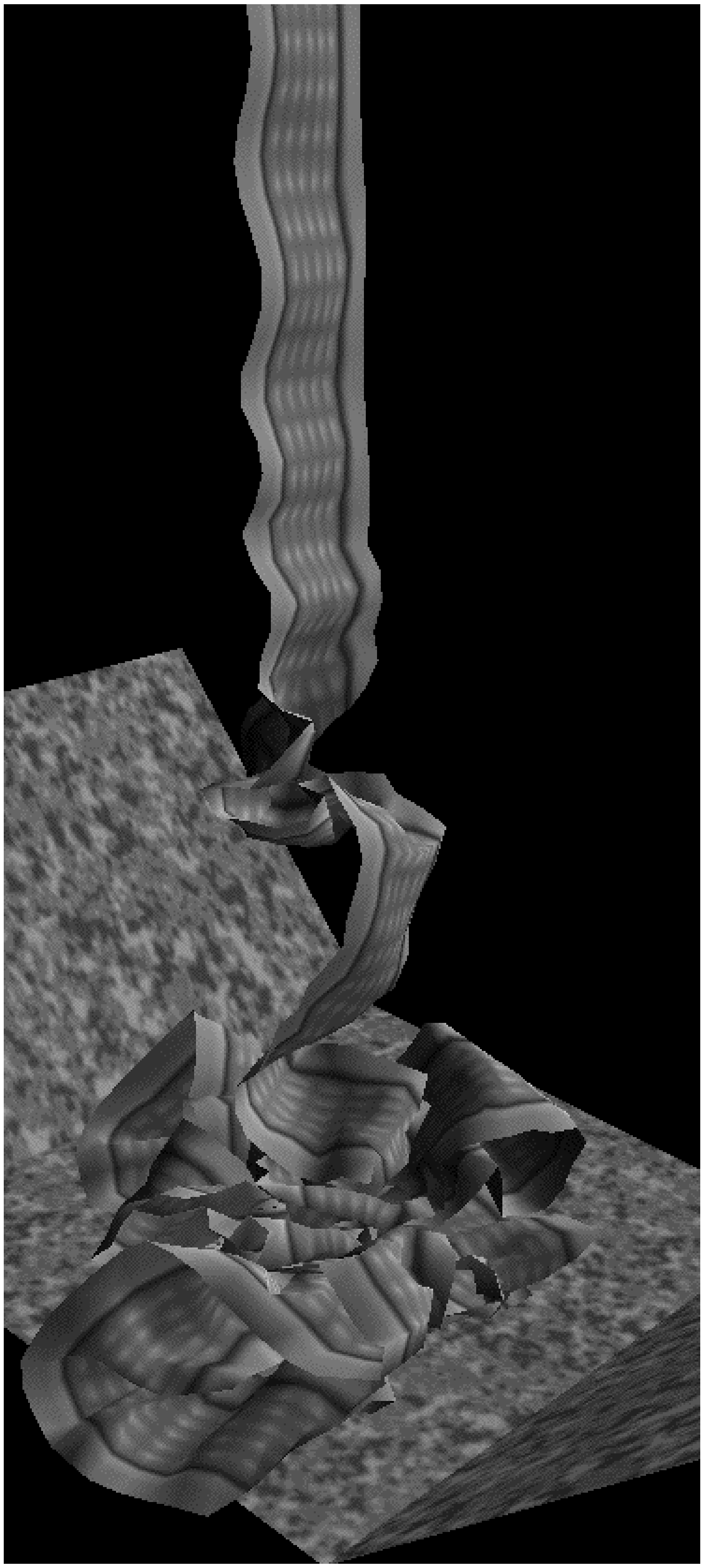
Some future work will consider maintaining consistency between different collision groups, as they may interfere each other when some elements get involved into several collisions.

4. Results and conclusion

The collision detection algorithm has been tested on various situations, from detecting self-collisions on a sphere up to complex collision situations involving wrinkling cloth. The result is quite obvious : We now get an average self-collision detection time proportional to the number of colliding elements. In situations like Figure 5, less than 10 % of the detection time is spent for cloth self-collisions. We obtain full power from the hierarchical algorithm, which is now not fooled by all the fake collisions resulting from adjacent elements.

Orientation consistency detection improves drastically simulation robustness for non oriented surfaces. As shown by Figure 4, a mechanical simulation on an initially inconsistent (interpenetrating) set of surfaces yields a consistent result. Each surface have been forces to "choose" the most probable side for each ambiguous collision.

The combination of our self-collision detection algorithm and the collision orientation consistency correction yields us a very efficient and robust system for simulating highly deformable surface. This can be included in a powerful cloth software, able to handle severe mechanical contexts and complex wrinkling situations. Way is then open for complex cloth simulation systems, that may include actors putting on and off cloth, or any other situation beyond the simply worn cloth situation.



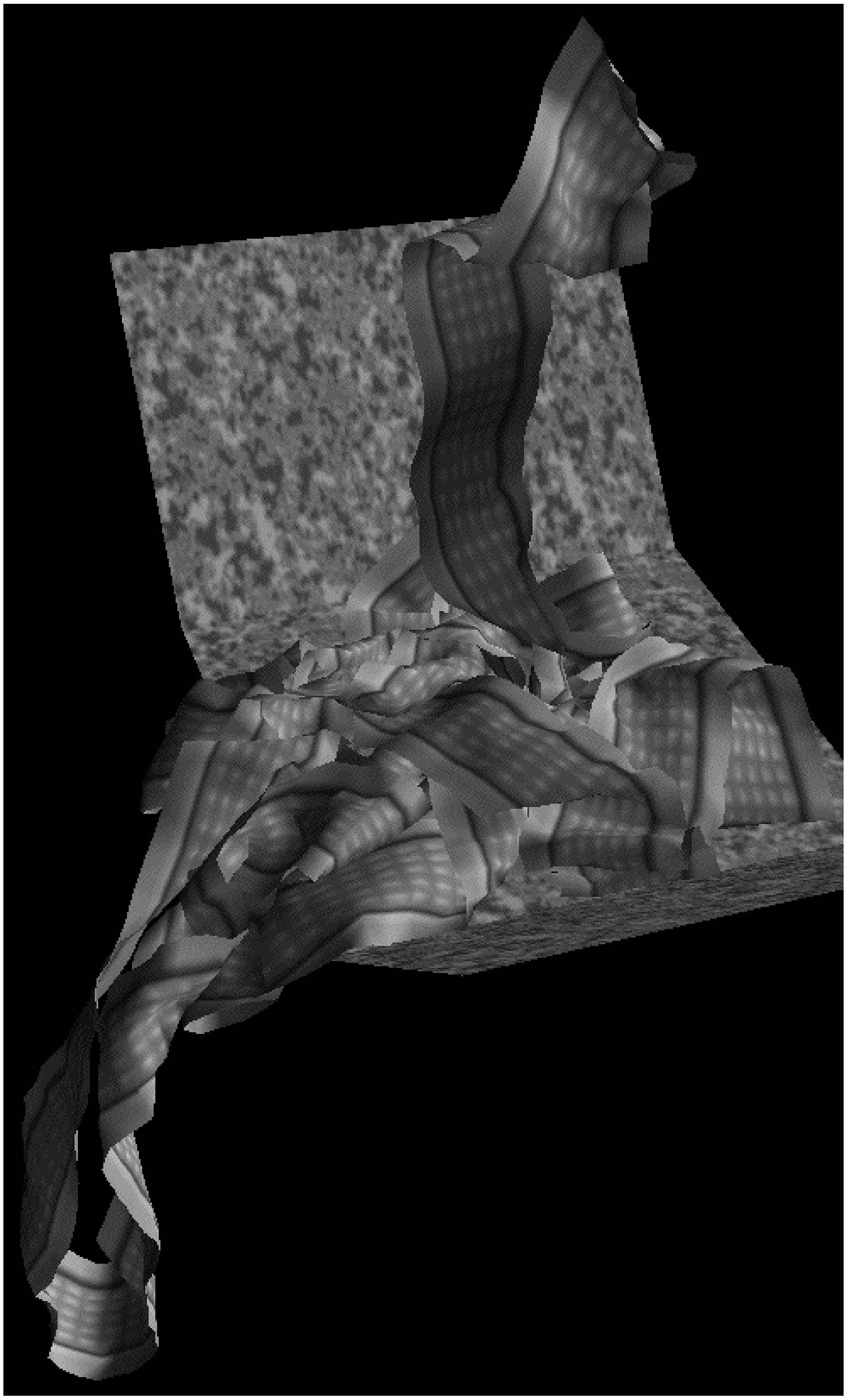


Figure 11

Acknowledgements

This work is supported by the Swiss National Foundation and the European ESPRIT HUMANOID 2 project. Special thanks to all the people that have contributed to it.

Bibliography

[BAR 90] : D. Baraff, "Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation", Computer Graphics, 24(4), pp 19-28, 1990.

[BAR 92] : D. Baraff, A. Witkin, "Dynamic Simulation of Non-Penetrating Flexible Bodies", Computer Graphics, 26(2), pp 303-308, 1992.

[CAN 91] : J.F. Canny, D. Manocha, "A new approach for Surface Intersection", International journal of Computational Geometry and Applications, 1(4), pp 491-516, 1991.

[DUF 92] : T. Duff, "Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry", Computer Graphics 26(2), pp 131-138, 1992.

[LAF 91] : B. Lafleur, N.M. Thalmann, D. Thalmann, "Cloth Animation with Self-Collision Detection", Proc. of the IFIP conference on Modeling in Computer Graphics, pp 179-187, 1991.

[LIN 93] : M.C. Lin, D. Manocha, "Interference Detection between Curved Objects for Computer Animation", Models and techniques in Computer Animation (C.A. proceedings 1993), pp 43-55, 1993.

[MOO 88] : M. Moore, J. Wilhelms, "Collision Detection and Response for Computer Animation", Computer Graphics, 22(4), pp 289-298, 1988.

[SHI 91] : M. Shinya, M.C. Forgue, "Interference Detection through Rasterisation", The journal of Visualisation and Computer Animation, 4(2), pp 132-134, 1991.

[SNY 93] : J.M. Snyder, A.R. Woodbury, K. Fleisher, B. Currin, A.H. Barr, "Interval Methods for Multi-Point Collisions between Time-Dependant Curved Surfaces", Computer Graphics annual series, pp 321-334, 1993.

[VHE 90] : B. Von Herzen, A.H. Barr, H.R. Zatz, "Geometric Collisions for Time-Dependant Parametric Surfaces", Computer Graphics, 24(4), pp 39-48, 1990.

[VOL 94] : P. Volino, N. Magnenat Thalmann, "Efficient Self-Collision Detection on Smoothly Discretised Surface Animations using Geometrical Shape Regularity", Computer Graphics Forum (EuroGraphics Proc.), 13(3), pp 155-166, 1994.

[WEB 92] : R.C. Webb, M.A. Gigante, "Using Dynamic Bounding Volume Hierarchies to improve Efficiency of Rigid Body Simulations", Communicating with Virtual Worlds, (CGI proceedings 1992), pp 825-841, 1992.

[YAN 93] : Y. Yang, N.M. Thalmann, "An Improved Algorithm for Collision Detection in Cloth Animation with Human Body", Computer Graphics and Applications (Pacific Graphics proceedings 1993), 1, pp 237-251, 1993.

[ZYD 93] : M. Zyda, D. Pratt, W. Osborne, J. Monahan, "Real-Time Collision Detection and Response", The journal of visualisation and Computer Animation, 4(1), pp 13-24, 1993.

[1] Published in Proc. Eurographics Workshop on Computer Animation and Simulation `95, pp . 55-65.

Back Home