Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity

Pascal VOLINO, Nadia Magnenat THALMANN

MIRALab,University of Geneva

Abstract

We present a new algorithm for detecting self-collisions on highly discretized moving polygonal surfaces. It is based on geometrical shape regularity properties that permit avoiding many useless collision tests. We use an improved hierarchical representation of our surface that, besides the optimizations inherent to hierarchisation, allows us to take adjacency information to our advantage for applying efficiently our geometrical optimizations. We reduce the computation time between each frame by building automatically the hierarchical structure once as a preprocessing task. We describe the main principles of our algorithm, followed by some performance tests.

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

1. Introduction

Discretized surfaces are widely used in all kinds of computer applications, like geometrical modeling, mechanical simulation, surface rendering,... Increasing power and memory of the computers we're actually using allows us to manipulate highly discretized surfaces, for better results in rendering or mechanical calculations. Besides, the algorithms we use should also be powerful, and the processing time should not increase too much if the number of elements increases.

Most computational simulations require algorithms for collision detection. These algorithms are often the bottleneck of the applications. Several methods have been presented in the existing literature, for detecting collisions in an efficient way, more or less specific for the context of their use.

We are actually working on a project that deals with handling cloth objects not only as worn by a synthetic actor, but also as independent object that can be grasped and fold, situations where the mechanical behavior is mainly determined by multiple self-collisions. We therefore have developed this collision detection algorithm very efficient in handling self-collisions.

1.1. Our problem

Our problem is to find self-collisions on discretized surface in an efficient way, for animation purposes. Typical use for this application is mechanical simulation and animation of soft surfaces, like fabrics, cloth, skin,...

We consider a surface as being a huge collection of triangles, quadrangles (or any other simple and flat polygon) connected by proximity to several others. Moving the surface between two frames means moving the vertices that support these triangles, and for each frame, we shall calculate which triangle collides with which other.

The particularities we consider are these :

- *A big number of elements* : Our method should reach O(n) average time complexity and the time constant should be minimal.

- *Animation* : We need fast collision redetection between each frame, while all the element have geometrically moved, but without noticiable change of topology.

- *Sparse collisions* : We assume that our surfaces are quite regular, ant the number of colliding elementary polygons is small comparated to the total number of polygons.

- *High level of adjacencies* : Each elementary polygon of the surface is adjacent to the neighbouring polygons These adjacencies should not perturb our algorithm or slow it down for detecting all these "false" collisions.

We will not consider mathematical algorithms, like those presented in [BAR 90], [VHE 90], [DUF 92], [SNY 93], and handle rather mathematical and parametrical surfaces, like polynomial patches rather than polygonal surfaces, nor rasterization algorithms ([SHI 91]) suited for fast and unaccurate preevaluation. Other algorithms well suited for highly discretised surfaces and animation are tracking the shortest distance between the convex envelopes of several objects ([LIN 93], [MOO 88]), but are not suited at all for self-collision on deformable objects. Voxelisation or octree-subdivision based algorithms ([SAM 84], [YAN 93]) need reconstruction of huge data structure between each frames, and thus are not suited for animation of an all-moving thousands polygons collection.

A good basis could be the general-purpose hierarchical algorithms, like those studied by [GOL 87], and more recently by [WEB 92] and [WEB 93] for collision detection, and which apply bonding-box testings onto any element of the hierarchy for efficiently elagating the search tree. They can be efficiently used for polygonal surface animation because the local topology of the polygons, and therefore also the hierarchy structure, do not change much between the frames.

Unfortunately, much time is lost for self-collision detection problem because any adjacency between elements of the hierarchy will be considered as being potentially a collision, and therefore lead to low-level testings for only noticing adjacent polygons.

1.2. Our contribution

Our contribution consists in implementing over an hierarchical algorithmic basis a practical way of taking avantage of geometrical regularity that prevent self-collisions from happening, and avoid going down to low level interference detection just because of adjacency contact. For this, we shall :

- Define some esay-to-calculate geometrical properties that will allow us to skip detecting collisions within big surface portions, as well as between adjacent surface portions.

- Find how to quickly evaluate and store the required geometrical information in the hierarchy.

- Find how to add efficiently extra adjacency information in the hierarchy for being able to find out rapidly whether two arbitrary elements in the hierarchy are adjacent or not.

In addition, we propose an algorithm which generates automatically a suited hierarchisation out of any polygonal surface, as a preprocessing task of our collision algorithm.

The performance tests will show us how self-collision detection performance is greatly enhanced, because our algorithm is now able to skip wide "uninteresting" zones, and concentrate only on the really near-to-collide zones, not perturbated by adjacency contacts that are now used for our advantage.

2. Detecting self-collisions

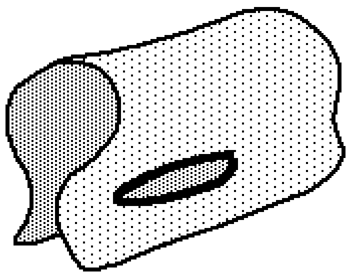
In this first part, we will explain the main principles and a precise description of our self-collision detection algorithm.

2.1. A fundamental geometrical property

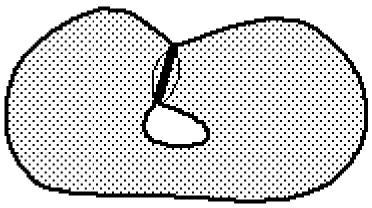
It is quite evident that a very smooth and regular surface will not show many self-intersections, at least at short ranges. In fact, the only causes of self-intersection are the following :

(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)



(b)

Fig. 1
Self-collision occuring because of curvature (a) or contour shape (b)

The algorithm we propose will take advantage of these facts.

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.

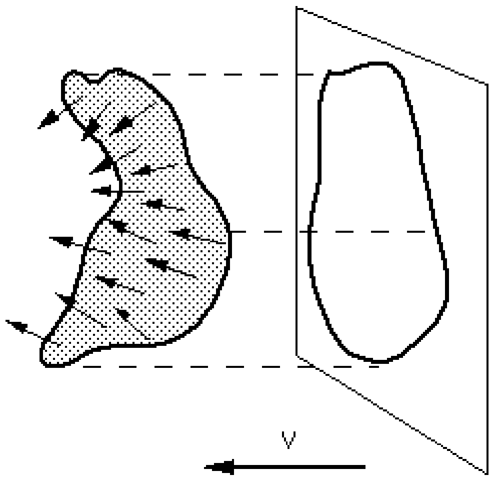


Fig. 2
Geometrical conditions
for no self-collision

Elements of demonstration : such a surface can be expressed as a function z(x,y), x,y being parametrization variables of the considered plane and z the height of the corresponding surface point from its projection on the plane according to the chosen direction.

2.2. Taking advantage of that property

This property has a fundamental impact for self-collision detection on surfaces :

2.2.1. 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, and test a 2D contour for collisions (that should contain at most O(sqrt n) elementary segments) also in an efficient way. 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.

Another consequence of this property is :

2.2.2. Collision between two surface areas

(II) - if we 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.

Now, we will study how to efficiently apply those properties.

2.3. Hierarchical collision detection using adjacency information

We will choose hierarchisation as being the underlying basis of our algorithm, because :

- It's a good way for speeding up collision detection when the number of objects becomes big. In our case, the number of polygon primitives resulting from a good surface discretisation in generally quite big.

- It's also a good way to divide a surface into regions on which we will try to apply our geometrical optimizations.

- We can efficiently include extra adjacency information in the structure tree, as we will discuss later.

2.3.1. Hierarchisation for discretized surfaces.

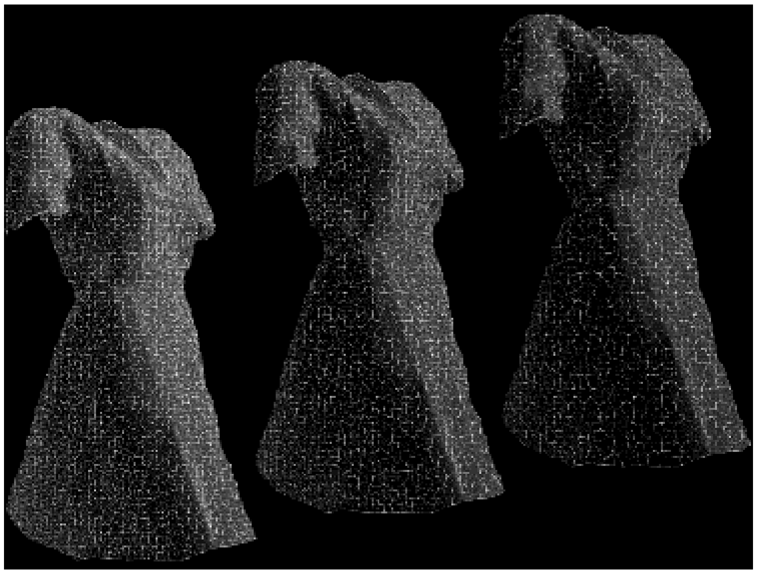
Surface hierarchisation has been used since now mostly for subjects that include topographical information storage, texture simulation, or rendering algorithms. Applications for collision detection appeared only recently ([WEB 93]).

For optimal processing during collision detection, a "good" hierarchisation tree should :

- Have O(1) maximal number of children for every node (The best value is 2).

- Have O(log n) maximal depth (the tree should be equilibrated).

Our automatical hierarchisation algorithm works upwards, from the lowest to the highest level. The main task consists in grouping several elements (a practical good value is 3) of a given level into a new element of above level, and repeat this process until the hierarchisation is complete. For getting correct geometrical regularity (rather round-shaped than irregular and narrow elements), we perform the grouping trying to improve as well as possible the shape factor of our elements `sqrt(S)/D`, `S` being the area and `D` the perimeter of our elements.



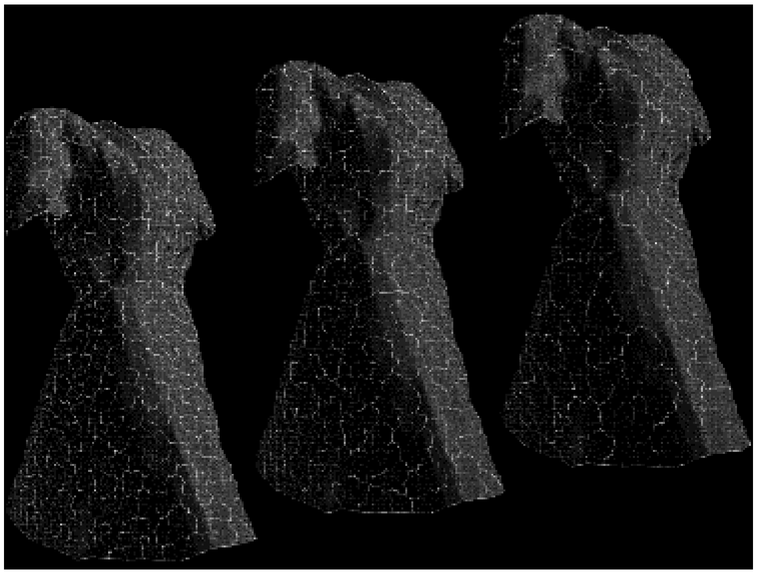


Fig. 3
Automatical hierarchisation of a 50000 triangle cloth. Shown levels : 5 to 10.

2.3.2. Collision detection for hierarchized objects

The classical, general purpose algorithm for collision detection in a hierarchized data structure using bounding boxes consinst in :

- (a) : Within an element, checking collision within any subelements (a) and between any couple of subelements (b).

- (b) : Between two elements, after bounding-box test for determining if collision could be possible, checking collision between any subelement of the biggest element with the smallest element (b), or calling the geometrical polygon-polygon collision check when botton hierarchy level is reached.

The power of hierarchisation comes from the cut in the search tree provided by bounding-box testing. But, as we said before, this is still inappropriate when detecting self-collisions on discretized surfaces, because bounding boxes will always find contacts between adjacent subobjects, should they really collide or not. Thus, we will loose much time detecting all the subobject adjacencies, having to go down to elementary primitive collision test for finally "noticing" that they were "only" adjacencies.

We can include the (I) optimization in (a) by skipping further collision checkings when the geometrical hypothesis that allow collision are not verified, i.e. if we can find a vector that has positive dot product with the normals of any polygon of this element, and then checking that the projected contour of the element along this direction does not self-intersect.

By the same way, we can include the (II) optimization in (b) by skipping further collision checkings if we can find a vector that has positive dot product with the normals of any polygon of these elements, and then checking that the projected contours of the elements along this direction do not intersect each other.

Now, three more difficulties remain :

- How to know if two subobjects of the hierarchy are adjacent or not ?

- How can we extract a vector for which the dot product is positive for each elementary polygon of a subobject ?

- How can we efficiently check if a projected subobject contour does have self-intersections or if two projected subobject contours do intersect each other ?

We shall now discuss about these problems.

2.3.3. Efficiently looking for adjacencies in a hierarchized surface

Our problem is this : Given two subsurfaces in the hierarchy, how can we know that these surfaces are adjacent (have at least one common vertex) or not ?

We should also respect these constraints :

- Keep O(n) global storage space, meaning average O(1) adjacency information for each subsurface.

- Keep O(log n) search time complexity.

This point is the major difficulty of our collision detection algorithm.

The storage

For each subsurface of the hierarchy, we define a circular list that contains the adjacent subsurfaces around its contour. We do not consider *all* these adjacent subsurfaces, but only the *highest level* subsurfaces that the original subsurface is *not* part of. (If the subsurface contour hits the real surface boundary, we consider "outside" as being a special subsurface that is not part of anything.)

In a more practical way, we consider the circular oriented list (not explicitly stored) of elementary edges that makes the contour of a subsurface, and we calculate for each edges the highest level subsurface that contains the elementary polygon behind the edge and does not contain the concerned subelement itself. We store explicitly the vertices (corresponding to the edges with orientation) for which the calculated subsurface is different from the calculated subsurface of the previous edge **(fig. 4)**. We could prove that any couple of subsurfaces do have *at least one* of these vertices in common. For multi-contour subsurfaces (i.e. holes), we will get several circular lists.

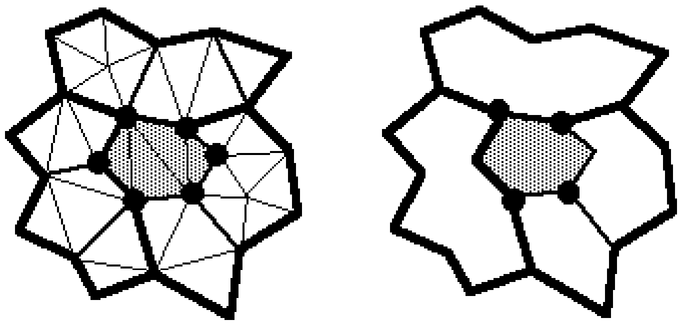


Fig. 4
Storing adjacencies: vertices to be stored around a subsurface

Practical tests has shown that for a "correct" hierarchisation, the number of adjacency elements in a list is O(1) and hardly exceeds 6.

The adjacency test

Testing two subsurfaces for adjacency is now quite easy : for each subsurface, we go through all (O(1)) adjacency vertices, and we test if any of the adjacent elementary polygons (O(1) if the polygonalisation is not pathological) belongs to the other subsurface (O(log n) testing by calculating the parents of the polygon).

2.3.4. Finding a common normal direction

The work is now to find a vector that dot product with the normals of all the elementary polygons of a given subsurface is positive.

Direction sampling

We represent the direction space by the unit sphere, and we sample that sphere with a collection of well-distributed normalized vectors, like for example the summit and face middle directions of a regular polyedra **(Fig. 5)**.

Finding a correct direction

For each subsurface of our hierarchy, we associate a bitfield sized to the number of sampling vectors, that contains, for this subsurface, whether each vector is valid or not.

For each frame, we evaluate for each elementary polygon the value of its bitfield according to its normal (a half-sphere will be discarded), and we propagate the calculation up in the hierarchy by logically AND-ing the bitfields **(Fig. 6)**.

If the value of the resulting bitfield is 0, there is no valid direction vector. But if not, any remaining valid vector can be chosen. Of course, this testing is done in O(1).

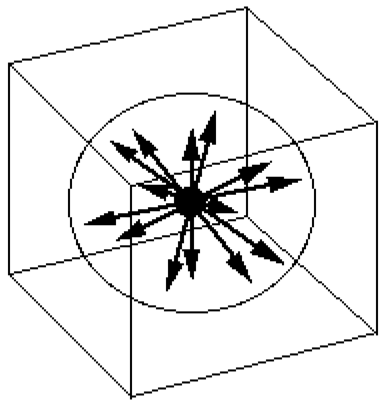


Fig.5
12 vectors direction sampling

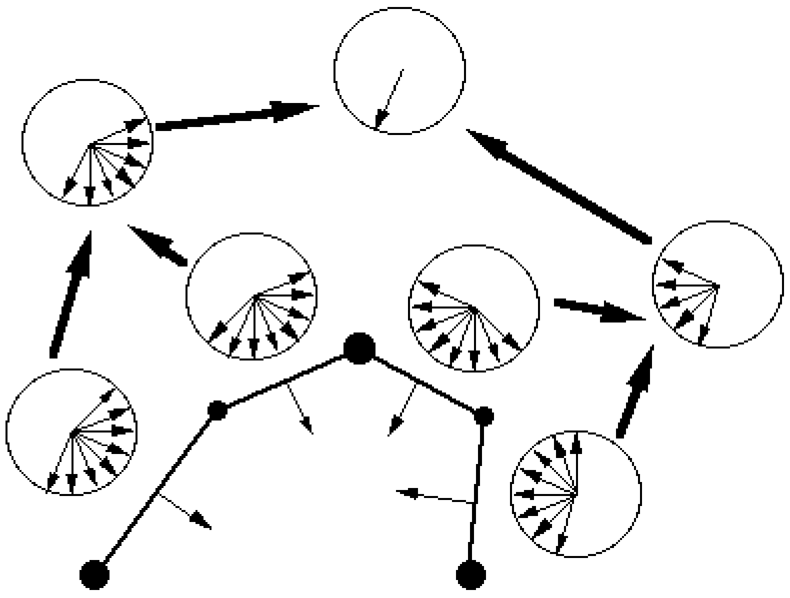


Fig. 6
Finding a valid direction for a surface

2.3.5. Checking for 2D contour collisions

Given a projection direction, we should now determine whether the 2D projected contour of a subsurface do self-intersect or not.

We could use an O(n log n) time 2D polygon intersection algorithm ([SHA 85]), or, better, an O(n) improved algorithm derived from convex hull search and taking advantage of the already existing bounding box hierarchy. But practical testings have shown that for allmost every common data configuration, this part of detection is not discriminent. In fact, a subsurface is quite unlikely to be bended slightely for self-colliding just because of its contour.

A good alternative is to remove this contour collision test, after ensuring that the contour shapes of the subsurfaces is "regular" enough, and do not contain deep and narrow concavities (for such pathological elements, the optimisation should be disabled).

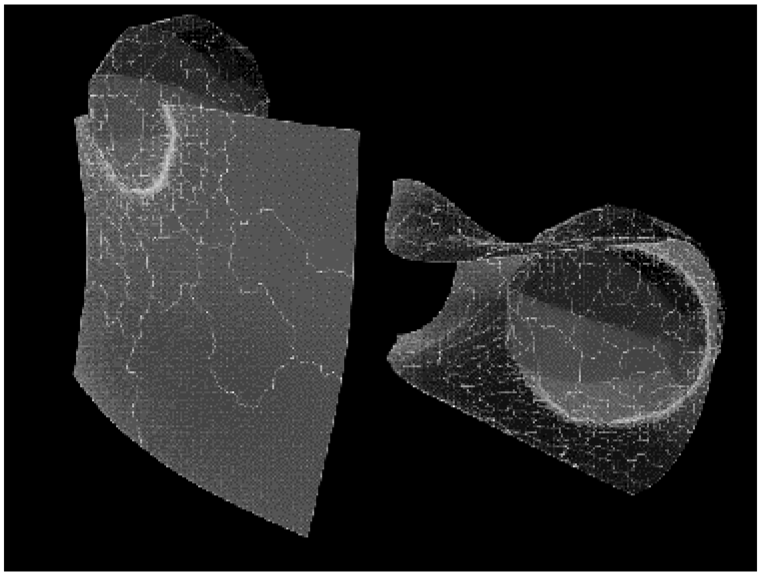
2.3.6. Multi-object collision support

The simplest way for generalizing multi-object collision detection would be to consider each object as a subsurface, grouped in the hierarchisation into a single root "metasurface". We handle that metasurface in the same way that we do for any other subsurface, except that we should not test for adjacency between its components. This naive solution can be applied if the number of objects remains small.

For a big number of objects, we could build a well-organized hierarchy of these metasurfaces optimized by the bounding box testings, with optionally dynamical updating of this hierarchy ([WEB 92]) whereas the objects are moving or are added between each frame. As the data of each object is rather independent, a new object can easily be inserted at any time in the scene.

2.3.7. Collision handling

This algorithm outputs all the couples of polygons that are potentially colliding in the scene. Out of this can be implemented the low-level procedure that, from these intersections, will calculate the collision information required by the host application (edge-polygon intersections, vertex-polygon proximities, dynamical collisions,...).



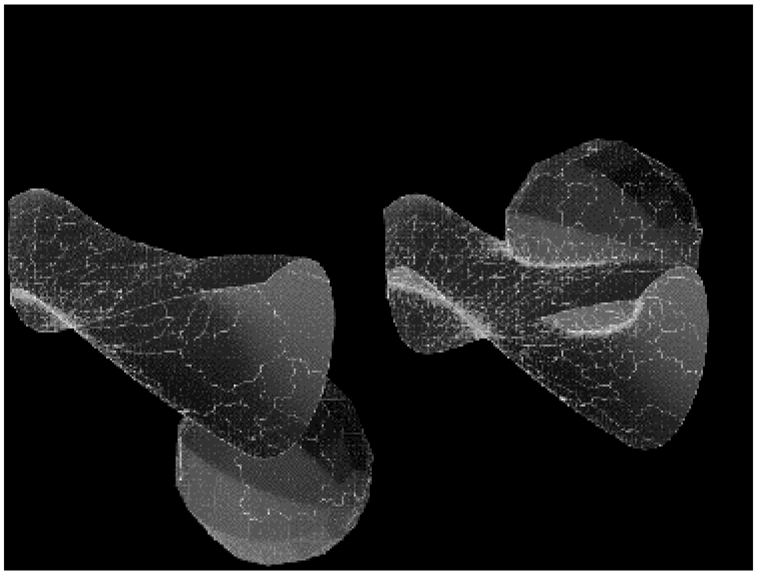


Fig. 7
A surface selfcolliding, a sphere and a surface colliding. The hierarchy elements used for testing collisions are shown.

3. Results and efficiency of our implementation

We have implemented a working version of our algorithm, using standard C language, on a *Silicon Graphics Indigo Elan* workstation.

For testing the efficiency of our implementation, we have run the algorithm on several data sets, varying the total number of polygons of the surfaces by changing the discretisation rate, and looking for the number of calls to the collision detection procedure (including recursive calls)

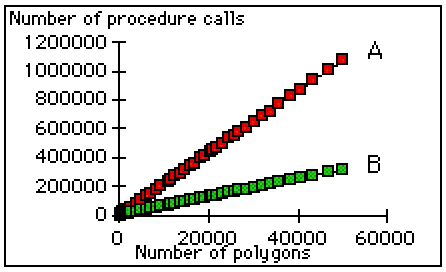
The discretisation is performed by incremental Delaunay subdivision, and we use automatical hierarchisation. We compare our algorithm with a standard hierarchisation and bounding-box algorithm, obtained by removing from our algorithm the geometrical and adjacency detections.

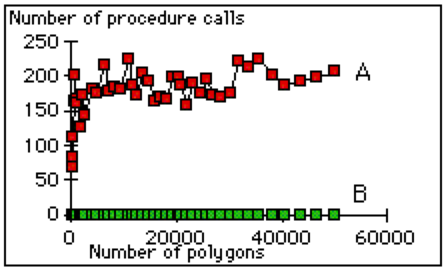
3.1. Sperical surface

Fig. 8 : A spherical surface obviously does not have any self-collisions... This data set is used to see how efficiently regular non-colliding parts of surfaces are removed from the collision detection process.

Tab. 1a reveals that for the classical hierarchical bounding-box algorithm, the calculation charge of the collision detection increases *linearly* with the total number of elements, for the same basic shape using different levels of discretisation. For instance, for the spherical surface, the collision procedure is called at average 20 times the total number of elementary polygons, and the elementary polygon intersection procedure is called 6 times that number, because of the "false" intersections caused by adjacencies.

At the opposite, **Tab. 1b** shows that our algorithm performs the job with *very little* and roughly *constant* calculation charge, not depending the discretisation level. For instance, for the spherical surface, we get an average 300 calls to the collision procedures, and the elementary polygon intersection procedure is *never* called. **Fig. 8** shows that the number of hierarchy elements considered during collision detection does not increase with the smoothness of discretisation.





Tab. 1a Tab. 1b
Number. of recursive calls to hierarchical collision detection procedure (A) and elementary interference procedure (B)
vs. number of polygons for a spherical surface, using :
(a) a classical hierarchical algorithm. (b) our geometrical algorithm.

This shows all the power of our algorithm :

** When there are no intersections, the self-collision search tree is truncated at constant depth level because of the regularity of the corresponding subsurfaces, whatever the discretisation rate.*

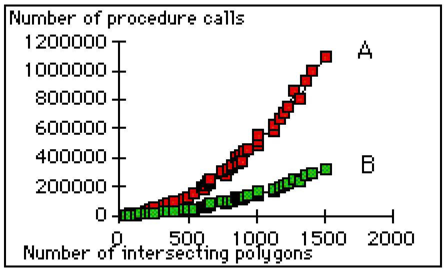
Thus, the time spent for noticing that a surface region has no self-collisions becomes ridiculously small, almost constant.

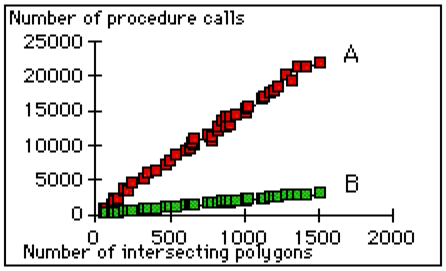
3.2. Self crossing surface

Fig. 9 : We use a regular surface that have a self-intersection by forming a "loop", a classical shape encountered when simulating for example fabric deformations. This data set is used for checking the detection efficiency regarded to the number of colliding polygons. For better comparison, we use the same discretisation rates as the ones for the spherical surface.

Tab. 2a shows that the classical hierarchised bounding-box algorithm is still mainly affected by the total number of polygons, thus giving performances close to those observed for the spherical surface (**Tab. 1a**).

At the opposite, **Tab. 2b** shows that collision detection is now performed at *linear* calculation charge regards to the number of *colliding* polygons. For instance, the collision procedures are called at average 20 times the number of colliding polygons, and the polygon intersection procedure is called 3 times that number. **Fig. 9** shows that high discretisation affects the search only on regions close to the collision points.

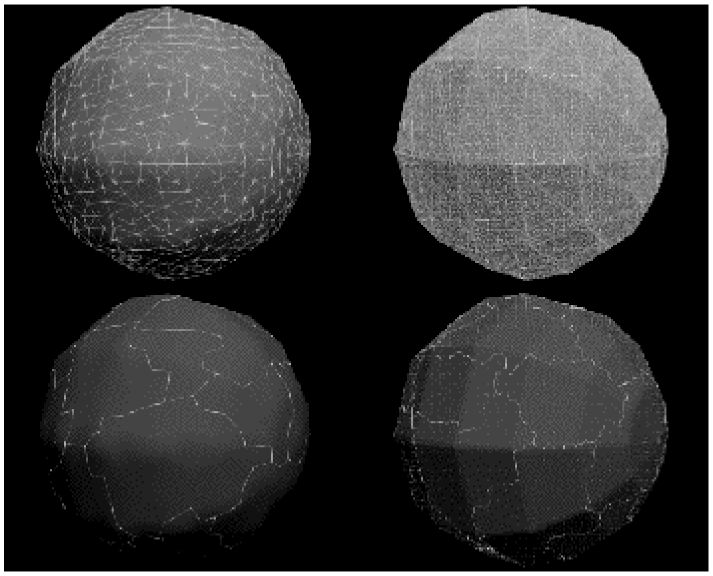




Tab. 2a Tab. 2b

Number. of recursive calls to hierarchical collision detection procedure (A) and elementary interference procedure (B)
vs. number of colliding polygons for a self-intersecting surface, using :
(a) a classical hierarchical algorithm. (b) our geometrical algorithm.

** The calculation charge of our self-collision algorithm is only* proportional *to the number of* colliding *polygons of the surface, not depending of the number of total polygons caused by the discretisation rate of the original shape.*



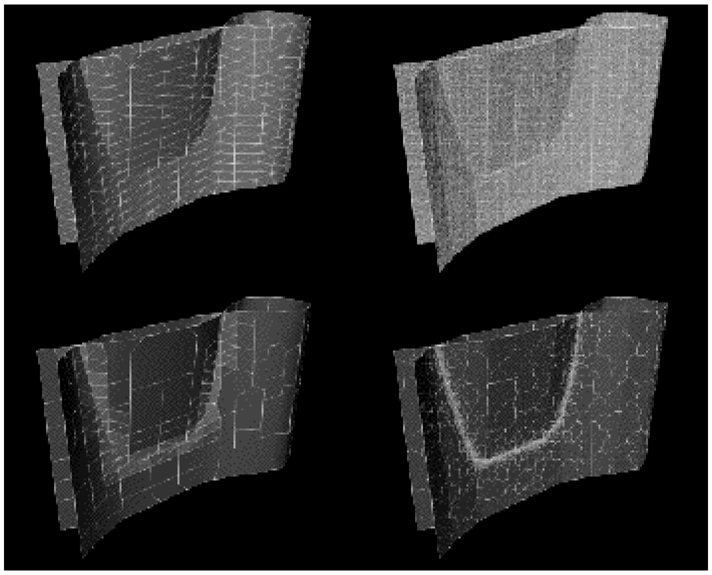


Fig. 8 Fig. 9
Detecting self-collisions for various discretisations on a spherical (8) and a self-intersecting (9) surface.
The used and shown hierarchy elements illustrate the influance of discretisation on the detection process.

3.3. Body-and-Cloth surfaces

Fig. 10 : This "real case" data set will show us the behavior of our algorithm when surfaces are close to each other. We detect collisions between the body and the cloth as well as self-collisions on the cloth. Thanks to Mr. Yang and M. Werner (MIRAlab Geneva) for providing the data.

We test for collisions on slightely rediscretised objects (about 3800 triangles for the cloth, 17200 triangles for the body) and on heavily rediscretised objects (about 56400 triangles for the cloth, 111000 triangles for the body).

Timing tests	Normal - BBox calculation time	Body-Cloth detection time	Cloth-Cloth detection time
Slight discretisation	**0.4 s.**	**0.4 s.**	(a) **0.7 s.** (b) **0.1 s.**
Heavy discretisation	**5.0 s.**	**6.0 s.**	(a) **10.0 s.** (b) **0.5 s.**

As we see, our algorithm increases highly the self-collision detection time, which was before the slowest process (a) and now becomes the fastest (b).

The figure shows the hierarchy elements used for the body-cloth collision detection, and as well the hierarchy elements for the cloth self-collision detection with our algorithm. We see that for self-collision detection, the number of considered elements remains small, concentrating only on in the places likely to contain collision, whereas the standard hierarchical bounding-box algorithm would have separated all the elementary triangles.

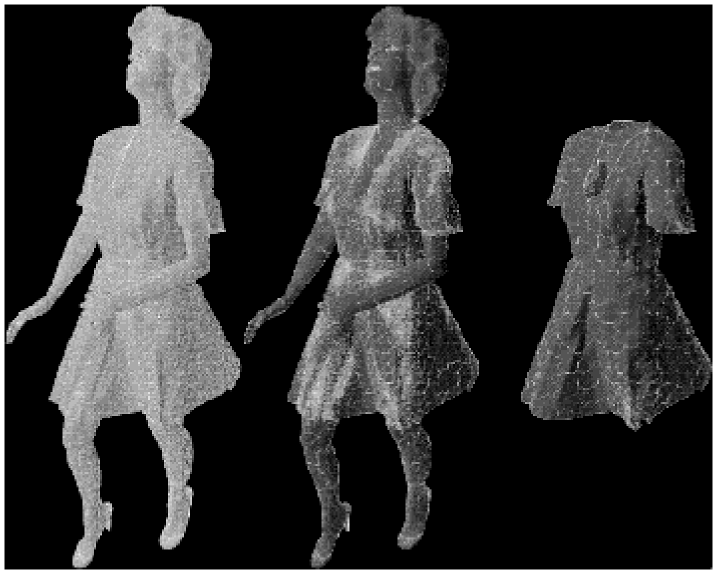


Fig. 10 : Cloth collision detection

4. Conclusion

We have developed a collision detection algorithm very efficient for animation of smoothly discretized surfaces. The main idea was to take advantage of the surface regularity that allows it to skip efficiently big surface regions during self-collision detection, providing drastical increase of speed. This could be done by making powerful use of adjacencies between parts of the surface, which usually represented rather a problem for earlier algorithms because of the "false" collisions they generated.

Thus, our algorithm skips efficiently non interesting parts of the surface, and concentrates only on the parts which are likely to collide.

Our algorithm is constructed on hierarchisation of the surface, that allows us to perform our optimization on each element of the hierarchy. We also take advantage of extra adjacency information included in the hierarchy. The data structure construction, of O(n) memory size, is only performed once, as a preprocessing task, with a suited automatical hierarchisation algorithm.

Between each frame, this algorithm is shown, for traditional discretized surfaces, to detect collisions in O(n) average time. Most of this time is spent updating normals and bounding boxes for each elementary polygon of the surface, and propagating the information in the hierarchy;

these operations are simple, and can be greatly optimized using triangle-specific procedures, or hardcoding (several actual graphic boards perform normal calculation for rendering purposes, and bounding box calculations are also easy to hardcode). The detection procedure duration is only roughly proportional to the number of colliding (or nearly to colliding) polygons, and generally *do not depend of the total number of polygons*, thanks to our geometrical optimizations.

This algorithm still remains quite general, and can easily be extended or optimized to perform some specific tasks (triangle discretisation, mechanical response, multi-object handling,...).

Our work is a good basis for any kind of application dealing with discretized surface animations (C.A.D., mechanical simulation, fabric-cloth-skin animation,...). Future work will integrate this algorithm into a mechanical system that will handle any kind of fabric objects as autonomous objects that can be grasped, manipulated, fold.

Bibliography

[BAR 89] : **D. Baraff**, *"Analytic Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies"*, Computer Graphics, 23(3), pp 223-232, 1989.

[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 86] : **J.F. Canny**, *"Collision Detection for Moving Polyhedra"*, IEEE transactions on Pattern Analysis and Machine Intelligence, 8(2), pp 200-209,1986.

[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.

[GOL 87] : **J. Goldsmith, J. Salmon**, *"Automatic Creation of Object Hierarchies for Ray Tracing"*, IEEE Computer Graphics and Animation, 7(5), pp 14-20, 1987.

[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.

[SAM 84] : **H. Samet**, *"The Quadtree and related Hierarchical Data Structures"*, ACM Computing Surveys, 16(2), pp 187-260, 1984.

[SHA 85] : **M.I. Shamos, F.P. Preparata**, *"Computational Geometry An Introduction"*, Springer-Verlag, N.Y., 1985.

[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.

[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.

[WEB 93] : **R.C. Webb, M.A. Gigante**, *"Distributed, Multi-Person, Physically-Based Interaction in Virtual Worlds"*, Visual Computing (CGI proceedings 1993), pp 41-48, 1993.

[YAM 85] : **F. Yamaguchi**, *"An Unified Approach to Interference Problems using a Triangle Processor"*, Computer Graphics (Siggraph proceedings 1985), 19, pp 141-149, 1985.

[YAM 86] : **F. Yamaguchi, T. Tatemichi, R. Ebisawa**, *"Applications of the 4*4 Determinant Method and the Triangle Processor to Various Interference Problems"*, Advanced Computer Graphics (Tokyo Computer Graphics proceedings 1986), pp 335-347, 1986.

[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.

Back Home