IEEE-RSJ Int. Conf. on Intelligent Robots and Systems
October 17-21, 1999. Kyongju (KR).

Fast Contact Detection between Moving Deformable Polyhedra

A. Joukhadar & A. Scheuer & Ch. Laugier
Inria[a] Rhône-Alpes & Gravir[b]
Zirst. 655 avenue de l'Europe. 38330 Montbonnot Saint Martin. France
Email: [Ammar.Joukhadar, Alexis.Scheuer, Christian.Laugier]@inria.fr

June 28, 1999

Abstract

This paper presents an approach to detect and localize contact between deformable polyhedra, which can be convex or concave depending on the time step. Usual contact detection algorithms, defined for convex polyhedra, cannot be used efficiently as they would imply to compute the convex decomposition of the considered polyhedra at each time step, as it can change due to the deformability of these polyhedra. The computation of this convex decomposition being very expensive (in complexity and computation time), we propose an algorithm to detect and localize the contact in linear time w.r.t. the number of vertices. this algorithm returns the direction of this contact and the value of the maximum intersection distance between the convex hulls of the two considered polyhedra. Experimental results, taken from a dynamic simulation application, are presented with their computation time to complete the complexity analysis.

Keywords — contact detection, deformable objects.

---

[a]Inst. Nat. de Recherche en Informatique et en Automatique.
[b]Lab. d'Informatique GRAphique, VIsion et Robotique de Grenoble.

Fast Contact Detection between Moving Deformable Polyhedra

A. Joukhadar & A. Scheuer & Ch. Laugier

Inria* Rhône-Alpes & Gravir†

Zirst. 655 avenue de l'Europe. 38330 Montbonnot Saint Martin. France

Email: [Ammar.Joukhadar, Alexis.Scheuer, Christian.Laugier]@inria.fr

Abstract

This paper presents an approach to detect and localize contact between deformable polyhedra, which can be convex or concave depending on the time step. Usual contact detection algorithms, defined for convex polyhedra, cannot be used efficiently as they would imply to compute the convex decomposition of the considered polyhedra at each time step, as it can change due to the deformability of these polyhedra. The computation of this convex decomposition being very expensive (in complexity and computation time), we propose an algorithm to detect and localize the contact in linear time w.r.t. the number of vertices. this algorithm returns the direction of this contact and the value of the maximum intersection distance between the convex hulls of the two considered polyhedra. Experimental results, taken from a dynamic simulation application, are presented with their computation time to complete the complexity analysis.

1 Introduction

It is well known that collision detection is a bottle neck for a large number of geometrical based algorithms, and in particular for dynamical based simulation (a large percentage of computational time is devoted to collision detection). Several interesting results have been obtained for processing collision between rigid polyhedra: Lin and Canny [5] have proposed an incremental algorithm which, in constant time, gives the positive distance[1] between two rigid and convex polyhedra in motion. This algorithm has been extended by Kotoku [4] in order to localize the contact when the distance between the two rigid and convex polyhedra is negative[2]. Gilbert and al [3] have proposed an algorithm (called GJK) which computes, in linear time w.r.t. the number of vertices, the positive distance between the convex-hulls of two sets of points (without computing these convex hulls), and gives an approximation of the negative distance in the case of small intersections. Garcia-Alonso et al [2], proposed an algorithm which represent an object by its min-max box, by its container and by voxels.

These algorithms can be used for concave polyhedra, dividing them into convex ones and applying the algorithms to the set of convex polyhedra. However, if this method can be used for rigid bodies (whose convex decomposition will never change), it cannot be used when deformable polyhedra are considered: the convex decomposition of each deformable object can have to be recomputed, which takes a quadratic time w.r.t. its number of vertices. To avoid this problem, Baraff & Witkin [1] divide objects into convex sub-objects that can only obey to first order polynomial deformation (a facet or an edge can not be curved), which guarantees that they stay convex during the simulation. Thus a fast collision detection algorithm between convex objects may be applied for each combination of sub-objects to detect the collision between the two main objects. However, the number of these sub-objects may be large when representing a highly deformable object. To reduce the computation cost, Volino & Thalmann [6] proposed a hierarchical algorithm which use the curvature properties of a surface in order to solve the self-collision problem. We will present in this paper another method to reduce the number of sub-object pairs considered for collision.

*Inst. Nat. de Recherche en Informatique et en Automatique.

†Lab. d'Informatique GRAphique, VIsion et Robotique de Grenoble.

[1]The distance between two polyhedra is called *positive* when the two polyhedra do not intersect.

[2]When two polyhedra intersect, their *negative distance* is the opposite of the smallest length of the translations needed to separate the polyhedra.

Outline of our approach

Let us first define a few notations: if X is a polyhedron, f_X and v_X are its respective number of facets and vertices, F_X^i ($i \in \{1, ..., f_X\}$) is its *ith* facet and C_X is its convex hull.

Given two deformable polyhedra A and B, systematically detecting collision between each pair of facets (F_A^i, F_B^j) ($i \in \{1, ..., f_A\}, j \in \{1, ..., f_B\}$) requires $O(f_A * f_B)$ operations. To reduce this complexity, we begin by eliminating from each polyhedron, before considering these pair of facets, some facets that can not be in contact, in order to determine two minimal sets of facets on which non-collision has to be verified. This elimination of facets is made in linear time $O(f_A + f_B)$ using three criteria, defined in § 3. These criteria are based on four entities, whose computation is deduced from an extension of the GJK algorithm, which computes the real value of the negative distance (§ 4). Once the two sets of potentially colliding facets are computed, using these four entities and the three associated criteria, GJK algorithm can be used between facets of each set in order to compute the distance between the two deformable polyhedra A and B. Results obtained with this method are given (§ 5), and future works are presented (§ 6).

This algorithm has one restriction: it only works when the contact points remain on the same side of each object (case 1 in Fig. 1). In other words, it does not work when the contact points are on opposite sides of one of the polyhedra (case 2 in Fig. 1).

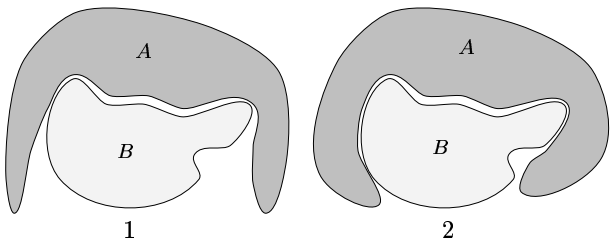


Figure 1: Restriction: the algorithm works in case 1 but not in case 2, as there exists in this case contact points on opposite sides of B.

2 Contact Definitions

Before presenting our algorithm, we need to give a few definitions about the contact of two polyhedra.

Let A and B be two polyhedra in $\mathbb{R}^3$ (whose inner parts are not empty). We say that A and B are in *contact* if, and only if, $A \cap B$ is not empty but has an empty inner part, *i.e.* $A \cap B$ is either a point, a segment or a planar region.

If $\vec{t}$ is a translation in $\mathbb{R}^3$, it is said to *bring A into contact with B* if, and only if, $\vec{t}(A)$ and B are in contact ($\vec{t}^{-1}$ then brings B into contact with A). Depending of the nature of $A \cap \vec{t}^{-1}(B)$ ($\subset A$) and of $\vec{t}(A) \cap B$ ($\subset B$), the contact is denoted as *vertex–vertex*, *vertex–edge*, *vertex–facet*, *edge–edge*, *edge–facet* or *facet–facet*.

At last, we denote as $\vec{t}_{A/B}$ the translation of smallest length ($|\vec{t}_{A/B}|$) which brings A into contact with B. The existence and unicity of this translation can easily be proved by considering each possibility of the nature of the contact.

3 Selecting Potentially Colliding Facets

Our algorithm is only used when the two convex hull C_A and C_B intersect, which can be detected using GJK algorithm in linear time ($O(v_A + v_B)$). Selection of the potentially colliding facets then uses the following entities, characterizing the contact (see Fig. 2):

- The negative distance $d_{A/B}$ between C_A and C_B is the length of the smallest translation needed to separate the two convex hulls C_A and C_B:

$$d_{A/B} = |-\vec{t}_{C_A/C_B}| = |\vec{t}_{C_A/C_B}|$$

- The contact direction $\vec{n}_{A/B}$ from A to B is the normalized vector of the previous smallest translation; it is directed from A to B:

$$\vec{n}_{A/B} = -\vec{t}_{C_A/C_B}/|\vec{t}_{C_A/C_B}|$$

- The contact plane $\mathcal{P}_{A/B}$ from A to B is the plane containing the point of C_A closest to C_B, and whose normal vector is $\vec{n}_{A/B}$; the space $\mathbb{R}^3$ is separated by this plane in two parts $\mathcal{E}^+_{A/B}$ and $\mathcal{E}^-_{A/B}$ (the later contains A and C_A).

- The impact zone $\mathcal{Z}_{A/B}$ from A to B is the projection on $\mathcal{P}_{B/A}$ of the intersection of A and $\mathcal{E}^-_{B/A}$.

Given these four entities, one can eliminate, in linear time, all facets from the polyhedron A that can not be in contact with the polyhedron B. Three criteria are evaluated for eliminating such facets:

- If $\vec{n}_{A/B} \cdot \vec{n}_{F_A^i} \leq 0$, where $\vec{n}_{F_A^i}$ is the external normal on the facet F_A^i, then the facet is invisible to the polyhedron B and can be eliminated; this is the case of the facets F_A^1, F_A^2, F_B^1, F_B^2, F_B^3 in Fig. 2.

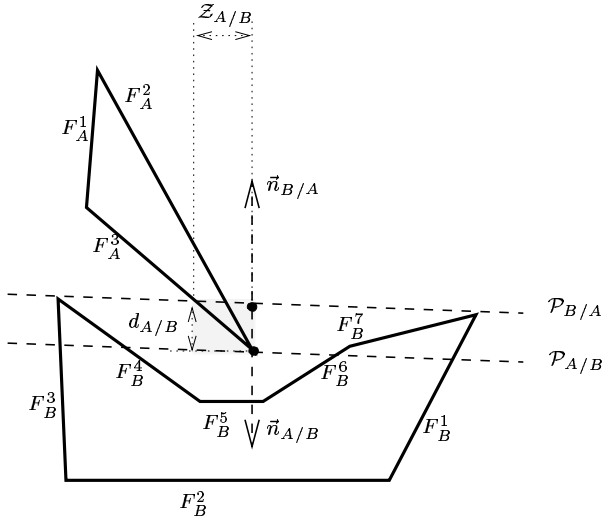


Figure 2: The facets F_A^1, F_A^2, F_B^1, F_B^2, F_B^3 can not be in contact, because they do not have the same side of the collision. The facets F_A^1, F_B^2, F_B^5 can not be in contact because they are far from the other polyhedron. The facets F_B^1, F_B^3, F_B^6, F_B^7 can not be in contact because they are far from the impact zone of the other polyhedron. Only F_A^3 and F_B^4 may be in contact.

- If $F_A^i \subset \mathcal{E}_{B/A}^+$, then the facet cannot be in contact with B as B is contained in $\mathcal{E}_{B/A}^-$; this is the case of the facets F_A^1, F_B^2, F_B^5.

- If the projection of F_A^i on $\mathcal{P}_{A/B}$ does not intersect $\mathcal{Z}_{B/A}$, then the facet can be eliminated; this is the case of F_B^1, F_B^3, F_B^6, F_B^7.

After applying these three criteria to each facet of each polyhedra (A and B), the algorithm returns two sets of facets which have not been eliminated. In the case illustrated in Fig. 2, these sets are $\{F_A^3\}$ and $\{F_B^4\}$.

The two first criteria are atomic and can be made in constant time $O(1)$ for each facet, once $\vec{n}_{A/B}$ and $\mathcal{E}_{B/A}^+$ are computed. To perform the facet selection is linear $O(f_A + f_B)$, the third criteria also needs to be computed in constant time. This is not possible if $\mathcal{Z}_{B/A}$ is computed exactly, as it will have $O(v_B)$ vertices: computation of the intersection will then cost $O(v_B)$. To obtain constant time, the impact zone $\mathcal{Z}_{B/A}$ will be approximated by a disk.

In that case, facet selection will be obtained in linear time $O(f_A + f_B)$ if the entities ($d_{A/B}$, $\vec{n}_{A/B}$, $\mathcal{E}_{B/A}^+$ and the disk approximation of $\mathcal{Z}_{B/A}$) can be computed in linear time $O(f_A + f_B)$.

4 Computing Contact Entities with a GJK Extension

As we already said, GJK algorithm computes, in linear time $O(v_A + v_B)$, the Euclidean distance between two convex hulls defined by their vertices in m-dimensional space. Algebraic tools are used in order to find iteratively the minimum (positive) distance between these two convex hulls, without considering the $v_A * v_B$ couples of vertices.

However, for our work, this algorithm has two main drawbacks illustrated in Fig. 3:

- It computes the distance between convex hulls instead of between polyhedra;

- It returns the Euclidean distance, and fails to find the negative distance when the convex hulls intersect.

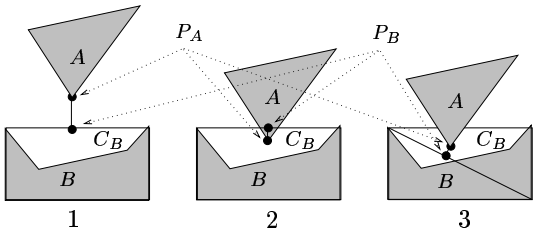


Figure 3: GJK algorithm gives the distance between two convex hull and returns the two nearest points P_A, P_B (1). When the two convex hulls interpenetrate (2), the algorithm may give a non correct negative distance (3), because it uses a local test to compute this distance.

The first drawback limits the use of GJK in our case: it checks whether the convex hulls C_A and C_B intersect, and in that case starts our collision detection algorithm. Indeed, as shown in Fig. 3, C_A and C_B can present a significant intersection while A and B are still apart.

The nearest points P_A and P_B returned by GJK cannot be used to compute the negative distance $d_{A/B}$ between C_A and C_B or the contact direction $\vec{n}_{A/B}$ from A to B, as the iterative nature of GJK does not garantee that these points are the right ones: P_A and P_B are not always respectively on C_A and C_B (*cf.* Fig. 3-3). We will show how this algorithm has to be extended in order to find the contact entities.

4.1 Finding the contact direction

The contact direction $\vec{n}_{A/B}$ from A to B can only be deduced correctly from the couple (P_A, P_B) return by GJK algorithm when the convex hulls C_A and C_B do not intersect, *i.e.* when the contact direction is not need (no collision can occur between A and B). When

C_A and C_B do intersect, we use a fast algorithm to incrementally transform $\vec{n}_0 = \overrightarrow{P_A P_B}/P_A P_B$ into the contact direction $\vec{n}_{A/B}$. This algorithm is based on two properties of intersecting polyhedra.

First of all, the nature of the contact between $\vec{t}_{C_A/C_B}(C_A)$ and C_B can only be vertex–facet, edge–facet or facet–facet. This can be proved considering the Minkowski set difference between C_A and C_B[3], denoted as $C_A \ominus C_B$: the contact between the origin O (which is inside $C_A \ominus C_B$ as C_A and C_B intersect) and $\vec{t}_{C_A/C_B}(C_A \ominus C_B)$ can only be of nature vertex–facet. Thus contact between $\vec{t}_{C_A/C_B}(C_A)$ and C_B always imply a facet.

We can thus prove that there exists a compact set $S_{A/B}$ of normalized vectors around $\vec{n}_{A/B}$ such that, for any $\vec{s} \in S_{A/B}$, if A is translated along $\vec{s}$ far enough to separate C_A and C_B, the new contact direction $\vec{n}$ (given by GJK, as C_A and C_B do not intersect any more) will be closer to $\vec{n}_{A/B}$ than $\vec{s}$ was.

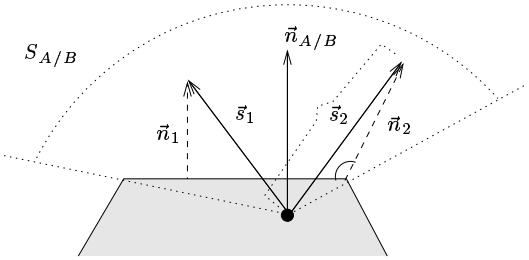


Figure 4: If $\vec{s}_i \in S_{A/B}$, any translation of A along $\vec{s}_i$ will give a new positive contact direction $\vec{n}_i$ closer to the previous exact contact direction $\vec{n}_{A/B}$.

The second property is that the contact direction $\vec{n}_{A/B}$ does not change when A (or B) is translated along $\vec{n}_{A/B}$.

Using these two properties, and an initial value $\vec{n}_0$ of the contact direction being given in $S_{A/B}$, $\vec{n}_{A/B}$ can be found using the following algorithm:

VECTOR Find_Contact_Direction($\vec{n}_0$)
{
Separate C_A and C_B according to $\vec{n}_0$.
Apply the Gilbert algorithm to get the new positive contact direction $\vec{n}_1$.
IF $(\vec{n}_0 = \vec{n}_1)$ **THEN**
return $\vec{n}_0$
ELSE
return Find_Contact_Direction $(\vec{n}_1)$
}

[3]The Minkowski set difference between two sets X and Y is defined as $\{x - y, x \in X, y \in Y\}$.

Remark: Direct computation of the contact direction would have a quadratic time $O(v_A * v_B)$, either using the Minkowski set difference $C_A \ominus C_B$ or considering C_A and C_B, as the contact sets between $\vec{t}_{C_A/C_B}(C_A)$ and C_B have to be computed.

In the contrary, the previous algorithm find the contact direction in linear time $O(v_A + v_B)$ as long as the initial value of the contact direction is in the inner part of $S_{A/B}$. If the choice of the contact direction given by GJK has experimentally proved to be a good initial value (the algorithm nearly always only need one iteration to find the contact direction), theoretical justifications of this quality of this choice should be point out for the final version of this article.

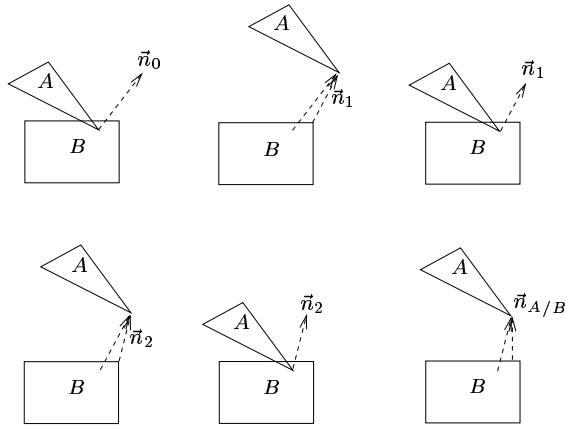


Figure 5: Translating A in the direction $\vec{n}_i$ and applying the Gilbert algorithm gives $\vec{n}_{i+1}$ as contact direction. In this example, $\vec{n}_3$ is equal to $\vec{n}_{A/B}$.

The behaviour of this algorithm is illustrated by an example in Fig. 5. This behaviour depends on two parameters:

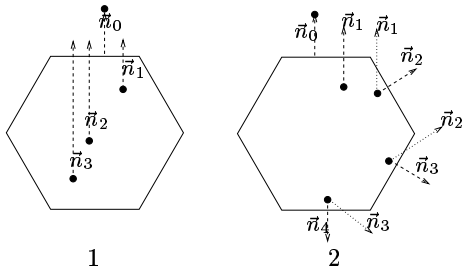
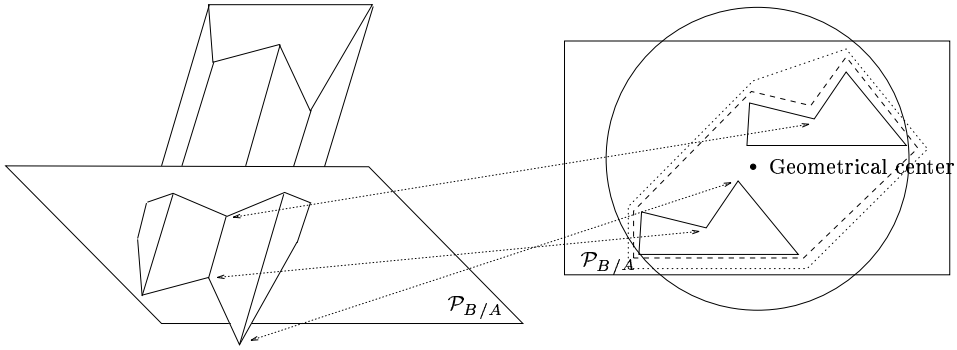


Figure 6: (1) The deepest point of C_A in C_B goes in the depth inside C_B: although it becomes sometimes closer to an other side of C_B, the algorithm always converge toward the side from which C_A came in, because it uses the last contact direction in order to initialize itself. (2) The deepest point follows the border of C_B: when $\vec{n}_i$ passes by through a new facet, the algorithm converges toward the normal of this facet $\vec{n}_{i+1}$.



The convex hull of the impact zone ------ The impact zone

Figure 9: The impact zone is approximated by a disk, in order to keep a linear complexity of the algorithm.

- The initial direction $\vec{n}_0$: Because of the continuity of the motion, the obtained contact direction can be used as an approximation of the contact direction in the next time step. If initially the distance between the two objects were positive then the Gilbert algorithm can give an initial value of this direction, we can use this value to compute the contact direction when the distance is negative and use this one to compute the contact direction in the next time step, etc... The fact that our algorithm has a local convergence (it converges always toward the facet by which passe $\vec{n}_0$) make it very robust for a significant amount of interpenetration (see Fig. 6).

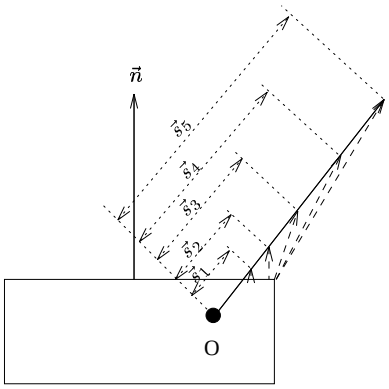


Figure 7: Influence of the translation length on the quality of the next contact direction obtained.

- The translation length at each iteration: The quality of the next contact direction obtained is inversely proportional to the difference between the translation length and the contact distance (which cannot be computed in linear time). This property is illustrated in Fig. 7. Thus we choose a translation length proportional to the negative distance obtained in the last iteration. If this value is not large enough, GJK algorithm will detect a negative value and our algorithm will duplicates this displacement.

4.2 Finding the Negative Distance $d_{A/B}$

To find the value of the negative distance $d_{A/B}$, the polyhedron A is translated of a distance d along $\vec{n}_{B/A}$, and GJK algorithm is used to compute the new positive distance d_p *cf.* Fig. 8). The original negative distance $d_{A/B}$ is equal to $d - d_p$.

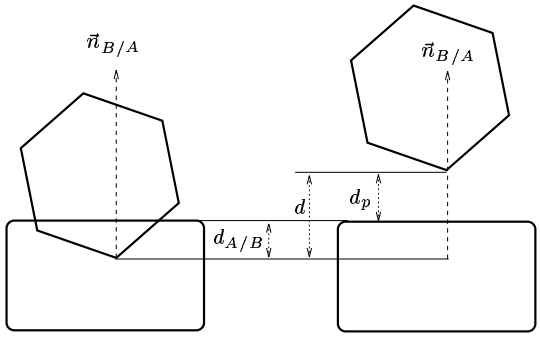


Figure 8: Computation of the negative distance.

4.3 Finding the Contact Region $\mathcal{E}^-_{A/B}$

When two polyhedra A and B have a very different dimension, it is very useful not to consider as potentially colliding all the facets of the bigger polyhedron but only those which are close to the smaller one. A wheel rolling on this ground, both being represented as polyhedra, is a good example of this situation.

An other simplification for the collision check is to only consider the part of B which intersects $\mathcal{E}^-_{A/B}$ (as defined in § 3). The contact direction $\vec{n}_{A/B}$ being given, the plane $\mathcal{P}_{A/B}$ can be computed in linear time $O(v_B)$. The contact region $\mathcal{E}^-_{A/B}$ is then deduced from $\mathcal{P}_{A/B}$ in constant time.

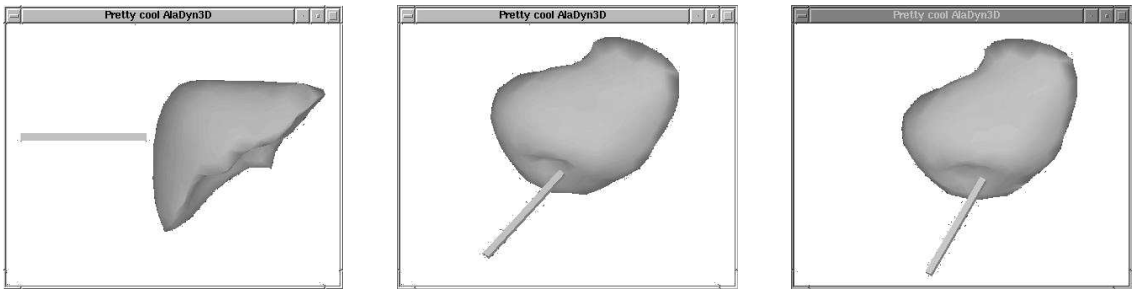


Figure 10: Real time deformation of a liver using AlaDyn3D.

4.4 Finding the Impact Zone $\mathcal{Z}_{A/B}$

The impact zone $\mathcal{Z}_{A/B}$ can be constructed from $\mathcal{P}_{B/A}$ and $\mathcal{E}^-_{B/A}$ in linear time $O(v_A)$. However, as we already noticed at the end of § 3, the complexity of this region is to high (its number of vertices is equivalent to v_A) to compute its intersection with B's facets in constant time (which is needed to have a linear complexity for the general algorithm).

Thus, $\mathcal{Z}_{A/B}$ is approximated, once computed, by the smaller disk which can contain all its vertices (*cf.* Fig. 9). The intersection of B's facets with this disk is then verified in constant time.

5 Implementation and tests

This distance algorithm is used in AlaDyn3D, a generic dynamic simulation model written in $C++$, to compute the amount of interpenetration between two objects, from which collision penalty models compute the collision response.

This allows to simulate the deformation by rigid tools of soft objects. This deformation is obtained in real time: a liver, represented by a polyhedra with 500 facets, is deformed by a rigid surgical tool with a rate of 12 images per second (*cf.* Fig. 10). The contact direction was monitored during these demonstrations and the average number of iterations needed to find the contact direction (in the Find_Contact_Direction algorithm, § 4.1) was 1.

An other application of this fast collision detection is the simulation of wide deformable objects, as soft clothes. A simulated contact between a sphere and a very flexible cloth has been experimented. Even when the sphere is completely inside the flexible cloth, the contact direction is still the same as in the beginning. While the shortest translation to remove the sphere from the cloth is **through** the cloth, the contact direction remains in the other direction and pushes the sphere **out** of the cloth, thanks to the local convergence of the Find_Contact_Direction algorithm (*cf.* Fig. 6).

6 Conclusion

This paper presented an incremental algorithm to detect and localize contact in linear time between two deformable polyhedra. This algorithm can be used to confirm collision when the convex-hulls of the two polyhedra intersect, without computing the convex decomposition of each polyhedra, this decomposition changing at every time step. The algorithm returns a sets of potentially colliding facets for each polyhedron and the translation of the length needed to separate the two convex-hulls. Collision between these facets can then be verified using classical collision detection algorithms (the facets are and remain convex).

The complexity of our algorithm is linear w.r.t. the number of facets (or of vertices) of both polyhedra, the linear coefficient depending on the precision of the estimation of the minimum translation's direction. Using a classical algorithm (Gilbert, Johnson and Keerthi) for this estimation, experimental results show that the average value of this linear coefficient is nearly minimum.

References

[1] D. Baraff and A. Witkin. Dynamic simulation of non-penetrationg flexible bodies. *Computer Graphics*, 26(2), July 1992. Proc. of Siggraph'92.

[2] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Journal on Computer Graphics and Applications*, 1994.

[3] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[4] T. Kotoku, K. Komoriya, and K. Tanie. A force display system for virtual environments and its evaluation. In *IEEE Int. Workshop on Robot and Human Communication (RoMan'92)*, Tokyo (JP), September 1992.

[5] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 4, pages 1008–1014, Sacramento, CA (US), April 1991.

[6] P. Volino and N. Thalmann. Efficient self-collision detection on smoothly discretised surface animations using geometrical shape regularity. In *EuroGraphics, Computer Graphics Forum*, 1994.