Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2008)
M. Gross and D. James (Editors)

Image-based Collision Detection and Response between Arbitrary Volume Objects

François Faure[1,2,3] and Sébastien Barbier[1,2,3] and Jérémie Allard[2] and Florent Falipou[2]

[1]Grenoble Universities
[2]INRIA [3]LJK-CNRS

Abstract

We present a new image-based method to process contacts between objects bounded by triangular surfaces. Unlike previous methods, it relies on image-based volume minimization, which eliminates complex geometrical computations and robustly handles deep intersections.
The surfaces are rasterized in three orthogonal directions, and intersections are detected based on pixel depth and normal orientation. Per-pixel contact forces are computed and accumulated at the vertices. We show how to compute pressure forces which serve to minimize the intersection volume, as well as friction forces.
No geometrical precomputation is required, which makes the method efficient for both deformable and rigid objects. We demonstrate it on rigid, skinned, and particle-based physical models with detailed surfaces in contacts at interactive frame rates.

1. Introduction

Collision detection and reaction is often the most computation-intensive task in physically based animation, and it has been thoroughly investigated for the last twenty years in several scientific communities including computer graphics, robotics and virtual reality.

Most of the methods proposed so far have been based either on geometric constraints between surface elements, or on the distance from points to surfaces inside volumetric fields. Surface-based methods process pairs of geometric primitives, such as triangles or spheres, which are actually or potentially intersecting each other. Potential interactions are typically detected using bounding volumes. Contacts are geometrically modeled based on close features and surface orientations. Reaction is implemented using forces or impulses, as illustrated in the left of Figure 2(a), where an ellipsoid repels a rectangle on its left. When the detection or the reaction fail due to large velocities or numerical errors, points cross a collision surface and several of them might end up far from it. Due to the short range of surface-based reactions, the simulation may not easily recover from this inconsistent state, as illustrated in the right side of the ellipsoid in Figure 2(a). Applying an adaptive time step is thus recommended to avoid deep intersections. Sometimes, proximity detection requires time steps so small that real-time simulations become impossible.

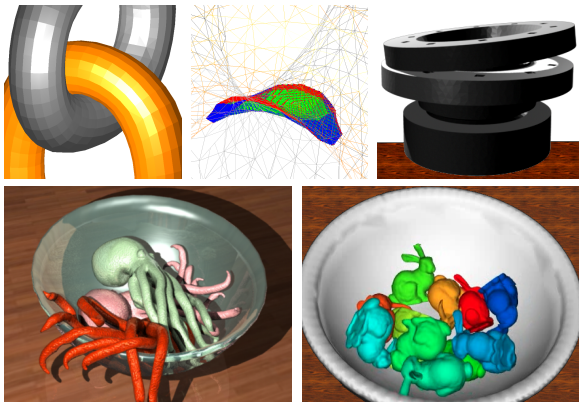

Figure 1: *A volume intersection sampled by the graphics hardware. Applications to rigid, skinned and FEM objects.*

Distance-based methods rely on a spatial map encoding the signed distance from a given point in 3D space to the surface of an object. Contacts are geometrically modeled

based on the distance and its gradient. Points are repelled using forces or projections to the surface, as illustrated in Figure 2(b). These methods are more robust to deep intersections than surface-based methods because they have a longer range. Unfortunately, the computation of the map is complex and can require a fine sampling of the volumes. In practice, the application of distance-based methods has been mostly limited to rigid objects, for which the map can be precomputed and stored in a local reference frame.

We present a new approach based on volume minimization, as illustrated in Figure 2(c). The intersection volume of two objects is modeled by pixels on the boundaries of the three-dimensional intersection volume to minimize. We introduce a new image-based method to efficiently model the intersection volume, as well as its derivatives with respect to vertex coordinates. The volume size and its gradient are then used to derive repulsion forces. A pressure increasing along with the size of the intersection volume is applied to the pixels and accumulated in their associated mesh vertices. No volumetric data structure needs to be built and maintained, since the intersection volume is entirely defined by its surface. This approach thus is applied to deformable objects as well as rigid ones, similarly to surface-based methods. Deep intersections are robustly handled and larger time steps are allowed, similarly to distance-based methods.

Given mesh geometries that model the contact surfaces of arbitrary volume objects, our method proceeds in two steps. First, the surfaces of the potentially colliding objects are rasterized into multiple layers of images. Then, the images are analyzed to detect intersections and compute forces. Our main contributions are:

- A new image-based model of the intersection volume between two polyhedra, using surface rasterization in three orthogonal directions;
- The integration of pressure forces over the pixels of the intersection volume, to compute forces applied to the vertices of the polyhedra.

Our method handles deformable as well as rigid objects without any precomputation. Applied to deformable volume objects, our approach combines the speed of surface-based methods with the robustness of distance-based methods. We obtain fast and robust interactive simulations in scenes including intertwined, colliding and self-colliding objects with detailed geometry.

The remainder of this paper is organized as follows. Section 2 summarizes previous work and presents the image-based method which we extend in our work. Section 3 presents a new contact model based on pairs of pixels computed in the GPU. Implementation details are given in Section 4, along with a comparison between a CPU- and a GPU-implementation of the second step of the method. Results are shown and discussed in Section 5. Finally, we conclude in Section 6.

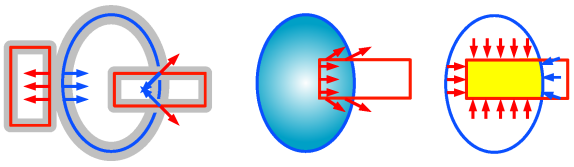

Figure 2: *Reaction methods. (a): surface-based. (b): distance-based. (c): our volume-based approach.*

2. Background

In this section, we only give a brief overview of collision detection and response techniques. For more details, we refer excellent surveys [JTT01, LM03, TKH*05] on the topic.

The inherent quadratic complexity of collision detection is typically alleviated using bounding volumes. As a consequence, intersection tests are applied only to the potentially colliding geometrical primitives. Robust collision detections can be performed using distance fields, which associates a closest feature on the surface to any points in space, thereby allowing deep intersections. The computation of such distance fields is generally a complex task, and in practice these methods have been mostly applied to rigid objects, since the distance field can be precomputed during the initialization of the simulation. Approximated distance field updates have been presented [FL01], based on volumetric meshes of object volumes, to accelerate the computation. Another difficulty comes from the discontinuities of the mapping from a 3D space to a surface, which can generate unstable responses. Propagation schemes avoiding inconsistent reactions have been proposed [HTK*04]. Alternatively, repulsions can be stabilized using forces parallel to the normals of the vertices penetrating the volume [BJ07].

Applying forces to the surface of the intersection volume using a polygonal mesh has been proposed [HFS*01], based on distance. Our image-based technique avoids both the complexity of creating and updating the polygonal mesh at each time step, and distance computations. The minimization of the intersection volume has been applied to the case of two-dimensional objects in cloth animation [VMT06]. This idea has also been applied to simple shapes in the mechanics community [CW05].

In recent years, image-based methods have been introduced to exploit the computational power of graphics hardware and their ability to rasterize polygons efficiently. Given mesh geometries, these methods return pairs of geometrical primitives which are then processed by the CPU to compute contact forces. Since they do not use precomputed volumetric data structures, they can be efficiently applied to both deformable and rigid objects, and they have been used to produce very impressive interactive simulations. The contact pairs can be composed of triangles [GLM05] or points

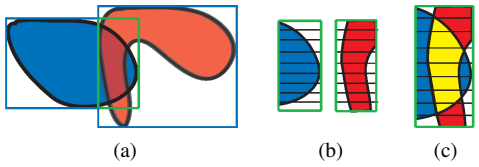

Figure 3: *The LDI process. (a): AABB computations, (b): Rasterization of each object, (c): Volume intersection.*

in volumes [VSC01, HTG03]. Extensions to continuous-time detections have been also proposed [GKLM07]. After the detection, additional CPU computations are required to model the intersections and the contact forces, with the same stability issues as in full CPU methods. Encoding geometric details on rigid bodies in textures have been addressed [OJSL04], using a rasterization-based penalty method for computing contact forces based on a penetration depth and direction estimate. The GPU can also compute distance fields using a uniform spatial grid [SOM04]. However this operation remains expensive for deformable bodies.

The computation of contact forces is most often performed and discussed independently of the collision detection. While contact response between rigid bodies is generally based on constraints [Bar94, MC95], contact responses for soft bodies can rely on penalty forces [BW98, VMT00], constraints [Pro97], or a sophisticated mix of them [BFA02].

Our intersection model uses *Layered Depth Images* (LDI), which were introduced to compactly represent multiple layers of geometry seen from one viewpoint [SGwHS98]. The approach was extended by Heidelberger *et al.* [HTG03, HTG04] to build geometrical models of volume intersections. Pairs of colliding axis-aligned bounding boxes (AABB) of the objects are computed (Figure 3(a)). Then, for each pair of potentially colliding objects, a viewing axis is chosen and the whole surface of each object is rendered in an array of textures encoding the LDI (Figure 3(b)). The viewing volume is set to the intersection of the bounding boxes of the objects, possibly augmented in the viewing direction to include all the surface layers. Collision Detection is then performed considering pairs of consecutive texels, or 3-tuples for self-collisions, along the viewing axis, computing the volume intersection with Boolean or per-vertex operations (Figure 3(c)). This fast and simple method allows collision and self-collision queries, along with vertex against volume tests, and requires no surface preprocessing. This makes it a perfect candidate for the simulation of complex deformable bodies. However, collision response is not addressed, and intersection depth in other directions than the viewing axis can not easily be inferred from the LDI.

3. Pixel-based Contacts

In this section, we propose a new geometrical contact model based on pixels. We extend the LDI method presented in the previous section to compute not only the intersection volume, but also its derivative with respect to the vertex coordinates. Deriving contact forces applied to the vertices is then straightforward.

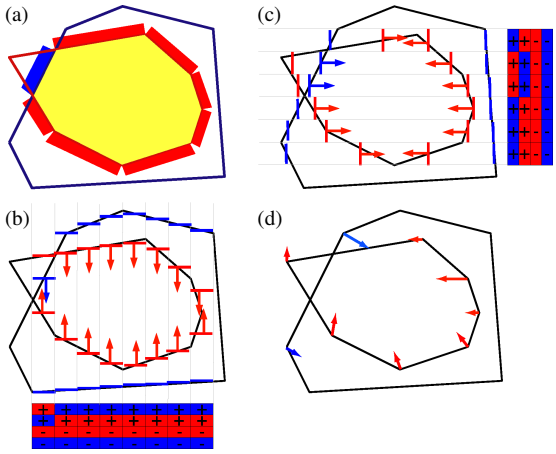

Figure 4: *Overview of the contact detection and modeling algorithm. (a): two intersecting objects, and their intersection volume in yellow, undergoing pressure forces (red and blue rectangles) on its surface. (b): rasterization of the surfaces in the vertical direction, and the intervals used to detect collision. The arrows denote repulsion pressure. The colors denote object ids while the signs denote normal orientations. (c): rasterization in the horizontal direction. (d): the pixel forces mapped to the vertices.*

3.1. Overview

Figure 4 illustrates our method in two dimensions for two intersecting polygonal objects (Figure 4(a)). The boundary of the intersection volume undergoes pressure forces which act to reduce the volume. Our method maps these pressure forces to the polygon vertices.

Polygons are rasterized in two orthogonal directions and stored in several images (the layers) to model the complete geometry. As a consequence, the volume of each object is represented by depth intervals at each pixel index. When the intervals of two bodies intersect, a collision is detected, and a pair of contact pixels is created. A pressure is thus applied in order to reduce the gap between the contact pixels in the viewing direction. The associated force is one-dimensional, parallel to the viewing axis (Figures 4(b) and (c)). Each pixel finally dispatches its contact force over the vertices using its barycentric coordinates. The resulting forces are illustrated in Figure 4(d). The net forces applied to each object are necessarily opposed, since they are the integrals of pairwise opposed pressure forces. We set the pressure proportional to the size of the intersection volume, using a contact stiffness parameter. Friction is handled using a simple extension presented in Section 3.3.

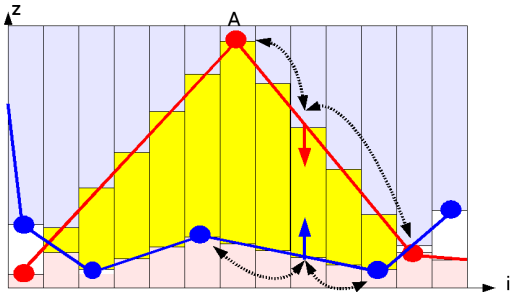

Figure 5: *A slice of our contact model. Vertices and triangles appear as disks and lines, respectively, while the intersection volume appears in yellow.*

3.2. The Intersection Volume and its Gradient

At each time step, the polygons of the intersection surfaces are rendered by the GPU into LDIs representing the different surface layers in a given direction. Three LDIs are computed, one for each direction of space. Figure 5 shows a slice of an intersection volume. Each vertical column corresponds to a different (i, j) pixel index. Each pixel depth represents one bound of an interval which defines an axis-aligned box in 3D space. The intersection volume is modeled by the union of box intersections, depicted in yellow in the figure. Its size is the sum of the box lengths multiplied by the area of one pixel:

$$\mathcal{V} = a \sum_{(i,j)\in\mathcal{C}} (-1)^d z_{ij}$$

where a is the area of one pixel, z is the depth and $\mathcal{C}$ is the set of indices of the fragments which cover the surface of the intersection volume. Exponent d is 2 for the upper pixel and 1 for the lower. The derivative of the volume with respect to the depth of a given vertex A represents the variation of the volume size corresponding to a unit displacement of the vertex in the viewing direction:

$$\frac{\partial \mathcal{V}}{\partial z_A} = a \sum_{(i,j)\in\mathcal{C}} (-1)^d \frac{\partial z_{ij}}{\partial z_A}$$

Notice that $\frac{\partial z_{ij}}{\partial z_A}$ is simply the Gouraud shading (*i.e.* barycentric) coefficient used to interpolate depth and color at pixel (i, j) based on the value at vertex A. We compute all these coefficients by rendering each triangle with one red, one green and one blue vertex and store the colors using additional channels in the LDIs. Each color channel defines the barycentric coefficient of a pixel with respect to one of its associated vertices. This extension of the standard LDIs allows us to map vertex displacements to the pixels, and conversely, to dispatch pixel forces on the vertices, as illustrated by black dotted arrows in Figure 5.

Our extended LDIs allow us to compute the partial derivatives of the intersection volume only in the viewing direction. Three orthogonal LDIs are thus necessary and sufficient to compute the volume gradient, *i.e.* the derivative of the intersection volume size with respect to the vertex coordinates. The gradient is independent of the directions used to build the LDIs, up to sampling artifacts, provided that the directions are mutually orthogonal.

The computation of the volume size and its gradient is performed using accumulators. The volume accumulator contains one scalar value, while the gradient accumulator is a vector containing three scalar values per vertex, each of them associated with one spatial axis. We first clear the volume and gradient accumulators. We then process the three orthogonal LDIs. Each pixel of each layer is visited, and it is skipped if it does not belong to a contact pair. Otherwise, its contributions are accumulated. Its depth is added to or subtracted from the volume accumulator, depending on its normal orientation. Similarly, its barycentric coordinates are added to or subtracted from the gradient accumulators, based on the same criterion, in the entry associated with the viewing axis of the LDI. While processing the LDI associated with the viewing direction z, a discrete approximation of the intersection volume is accumulated, along with its derivative with respect to the z coordinate of each vertex. Once the three orthogonal LDIs have been processed, the volume has been accumulated three times, while its derivative with respect to each coordinate of each vertex is accumulated in the scalar entries of the gradient vector

$$\frac{\partial \mathcal{V}}{\partial x} = \left(\frac{\partial \mathcal{V}}{\partial x_A}, \frac{\partial \mathcal{V}}{\partial y_A}, \frac{\partial \mathcal{V}}{\partial z_A}, \frac{\partial \mathcal{V}}{\partial x_B} \ldots \right)$$

, expressed in the basis defined by the viewing directions. Finally, the volume is divided by 3.

3.3. Penalty Forces

Our penalty contact forces are a natural extension of the detection algorithm, taking advantage of the already computed intersection volume to accumulate forces. They are derived from the very simple assumption that the repulsion forces try to minimize the intersection volume. We define the potential energy associated with an intersection volume as $E = \frac{1}{2} k \mathcal{V}^2$, where $\mathcal{V}$ is the size of the intersection volume and k is an arbitrary positive constant. This enables us to straightforwardly model repulsion forces by deriving the potential energy with respect to the vertex coordinates. We focus on surfaces defined by triangular meshes. The control variables are the mesh vertices. The forces applied to the vertices are:

$$f = -\frac{\partial E}{\partial x} = -k\mathcal{V}\frac{\partial \mathcal{V}}{\partial x} \quad (1)$$

where x is the vector of vertex coordinates.

The repulsion forces are parallel to the normal of the triangles and proportional to their area, since the volume gradient is the direction of maximum volume variation. The forces

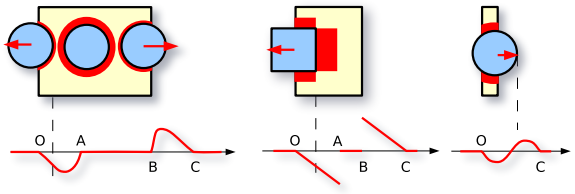

Figure 6: *Pressure force applied to objects traversing fixed obstacles. Top: in red, the pressure applied to the intersecting surface, and the associated net force. Bottom: in red, the repulsion force as a function of the abscissa of the objects.*

can thus be interpreted as the sum of pressure forces integrated over the surface of the intersection volume. The behavior of these forces is illustrated in Figure 6. On the left, a moving sphere traverses a fixed box. A repulsion force appears when the sphere enters the box (point O) and vanishes once the sphere is totally included (point A). No repulsion occurs as long as the sphere is inside the box. As soon as the sphere internally hits the surface of the box (point B), it is progressively repelled outside. The repulsion stops when the intersection vanishes (point C).

Force discontinuities may occur when an object internally touches another one and the surfaces match exactly, as shown by the cube in points A and B in the middle of the same figure. In practice, we generally want to avoid large intersections and we apply an appropriate contact stiffness to stay away from this configuration. Traversing thin objects generates continuous forces, as illustrated in the right of Figure 6.

If one wants to apply implicit time integration, a stiffness matrix is required. This matrix encodes the variation of forces in response to a variation of positions. This variation is due to two factors. The first is the change of size of the intersection volume, and the corresponding stiffness matrix is obtained by differentiating the volume $\mathcal{V}$ in equation 1 :

$$\frac{\partial f}{\partial x} = -\frac{\partial \mathcal{V}}{\partial x} k \frac{\partial \mathcal{V}}{\partial x}^T \tag{2}$$

The second factor is the variation of the area enclosed by the intersection of the surfaces. This corresponds to the length of the blue rectangle in the top left of Figure 4. It depends only on the vertices of the triangles of one object which intersect triangles of the other object. It is more difficult to compute and for simplicity, we drop this term and use equation 2 to approximate the stiffness matrix. It fits well with the conjugate gradient algorithm used in implicit integration, since it is symmetric and its product with a vector can be computed efficiently without determining the entries of the matrix.

To the best of our knowledge, the integration of pressure forces on the surfaces of the volume intersection has not been used in previous work, probably because the analytical computation of the volume of the intersection of two polyhedra is generally complex, not to mention its derivative. In order to extend the method to the computation of viscous friction, which is opposed to the relative tangential velocity, we render the velocities in additional images in the LDIs. At each contact pixel, the relative velocity is computed and accumulated at the triangle vertices using the Gouraud shading coefficients as in Section 3.2. In contrast with the gradient computation, the pixels accumulate three-dimensional values rather than scalars. Since we consider tangential velocities, the entry associated with the viewing direction is discarded. Once multiplied by the pixel area and a viscosity coefficient, the result is the integral of viscous forces in the tangential planes, distributed on the vertices.

4. Implementation

This section presents a brief overview of our algorithm and technical implementations on the current graphics hardware. At each time step, our algorithm performs the following phases :

- Find pairs of overlapping bounding boxes (broad phase);
- For each pair of overlapping bounding boxes :
  1. Set the rendered volume to the intersection of the bounding boxes and upload meshes to GPU;
  2. Render the LDIs of the objects;
  3. Read back the LDIs to the CPU;
  4. Compute the intersection volume and its gradient;
  5. Perform the mechanical integration, considering the penalty forces.

We detail hereinafter more precisely each phase and discuss a more powerful GPU implementation for Phase 4.

The LDIs are built in Phase 2 using a GPU-friendly *depth-peeling* algorithm based on [Eve01]. We exploit the functionalities of the new graphics hardware to enhance the efficiency of the approach. During this stage, two colliding objects are simultaneously rendered to sort their texels, freeing us from applying Boolean intersections on the CPU afterwards as in [HTG04]. Each mesh is represented as a soup of triangles and is sent to the GPU. The fragment shader outputs into a RGBA texture for each pass the barycentric coordinates of the corresponding pixel, along with the index of the rasterized triangle and the index of the object it belongs to.

Occlusion queries are exploited to determine when the scene becomes empty. Consequently, for each pair of potentially colliding objects, the total number of rendering passes in each direction is equal to the maximum number surface layers in the viewing direction, plus one, as in standard LDIs.

In our first implementation, the rendered layers are transferred to the CPU for further processing in Phase 3. This step is often the most prohibitive. To remove this bottleneck, we implemented a GPU-based method to accumulate

the volume contributions from the rendered layers in Phase 4 that discards the read back (Phase 3). Our implementation is based on the NVIDIA CUDA API, but other choices are available depending on the brand of GPU. This allows us to transfer much less information between the GPU and CPU, as the results of Phase 4 are only the intersection volume (a scalar) and its gradient (a sparse vector with up to $3n$ scalar entries, n being the number of surface vertices). This provides a good speedup, as it improves performances by 10 to 20 percents. Also while the GPU handles this computation, an interesting advantage is that the CPU is free to complete other computations in parallel.

5. Results and Discussion

We have implemented our method within the SOFA framework [ACF*07] and applied it to various deformable, rigid and skinned models, with geometries ranging from simple cubes to highly detailed meshes, using stiff penalty forces and implicit time integration. The time steps typically range from 0.01 to 0.04s. All our tests are performed on a computer with 2GB of memory, an Intel Core 2 Duo 2.66GHz processor and a GeForce 8600 GTS graphics card. Only one core was used for our evaluation.

Since no assumption about the constitutive laws of the objects is used, we are able to animate arbitrary physical models, provided that they are embedded in a watertight polygonal mesh as shown in Figure 1. For instance, CAD models can be directly animated from their detailed polygonal models. Furthermore, the ability of our method to handle collisions and self-collisions without geometry preprocessing makes it especially useful for simulation of deformable objects with detailed geometry and intricated contacts, as illustrated in Figure 7. The octopus is modeled using articulated rigid bodies embedded within a deformable skin. Due to penalty-based repulsion which applies a force rather than a non-penetration constraint, it sometimes happens that some collisions are not totally resolved. The robustness of the detection and of the volume-based response allows recovery in the subsequent time steps. This enables us to run the simulation with a fixed time step and to interact in real time with one octopus.

Table 1 details the measured performance on a scene containing a rigid bunny represented by either a fine (69674 triangles) or coarse (21553 triangles) mesh and colliding with a cube using two LDI resolutions (16×16 or 128×128).

Comparing with other methods is difficult, since most of them consider collision detection independently of reaction, while in our method the force is a natural extension of the detection algorithm. In contrast with distance maps, edge contact is automatically handled without edge supersampling. Sharp edge contact is possible without visible intersection, as illustrated in the left of Figure 8. The contact area between the two rigid cubes is theoretically reduced to a single point

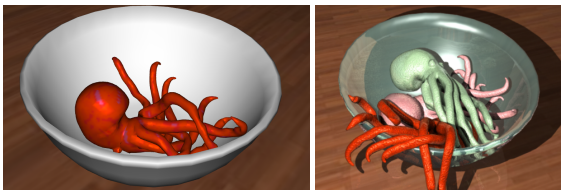

Figure 7: *Highly deformable models. Left: skinned octopus in a rigid bowl, 20000 triangles, 25 fps. Right: three instances of the same with offline rendering.*

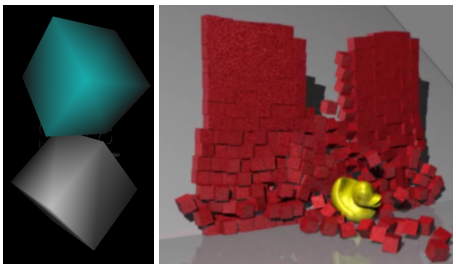

Figure 8: *Sharp geometry. Left: 256×256 contact resolution, implicit Euler integration, dt=0.001, $k = 10^{10}$. Right: 64×64, dt=0.01, $k = 10^5$.*

and an infinite stiffness would be necessary to avoid an intersection. Using unit masses and dimensions, and applying earth gravity to the upper cube while the other is fixed, we are able to maintain the unstable equilibrium with no visible intersection during a few seconds. This shows that even if our general-purpose model cannot compete with specialized approaches, its ability to sample contacts at arbitrary resolution allows a broad range of applications. Lower resolutions and stiffnesses can be applied when small intersections are visually acceptable, such as the scene including a large number of objects in Figure 8.

In contrast with surface-based approaches, our method can safely handle scenes including objects which are already intersecting at initialization time, such as medical simulations based on non-perfect geometric models illustrated in Figure 9.

The time step can however be limited by the discrete nature of our detection. Well-known artifacts can appear, such as fast objects traversing thin volumes without being detected, or biased repulsion directions due to late detection. Given a mesh thickness at the thinnest point and the maximum object velocity, one can deduce a safe time step size. If necessary, the method can also be used in conjunction with a continuous-time collision processing method. Another limitation is due to our penalty-based reaction model combined with discrete detection. Some potential energy is created each time a new intersection is detected. The amount of induced energy increases along with the penetration depth and thus depends on the time step. When deep intersections

Mesh, Resolution	BP	Upload	LDI	Vol.	Sim.	Render.	Total
coarse, 16×16	0.077 (1.5%)	0.478 (9.3%)	2.101 (41%)	0.039 (0.8%)	0.315 (6.1%)	1.951 (38%)	5.13
coarse, 128×128	0.074 (0.6%)	0.492 (3.8%)	7.874 (60.2%)	1.905 (14.6%)	0.4 (3.1%)	1.908 (14.6%)	13.09
fine, 16 × 16	0.508 (1.3%)	2.964 (7.5%)	15.20 (38.3%)	0.046 (0.1%)	0.23 (0.6%)	20.01 (50.5%)	39.63
fine, 128 × 128	0.494 (0.8%)	2.842 (4.6%)	36.97 (59.3%)	1.949 (3.1%)	0.258 (0.4%)	19.12 (30.7%)	62.35

Table 1: *Times in milliseconds of each phase: - broadphase (BP), computing pairs of potentially colliding objects; - upload of meshes to the GPU; - LDI generation; - collision volumes computation; - physical simulation; - and rendering.*

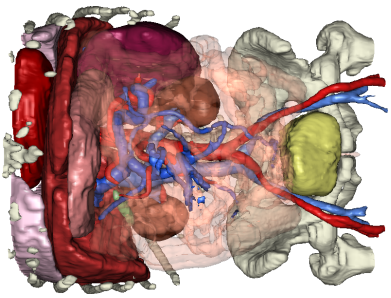

Figure 9: *Patient-specific anatomic model where some organs are already intersecting at initialization time.*

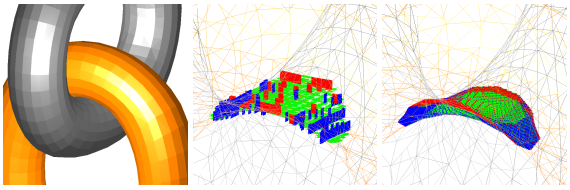

Figure 10: *Intersection volume at different resolutions. Left: the collision surfaces. Middle: the intersection volume at resolution 16×16. Red, green and blue pixel show the bounds in x,y and z directions, respectively. Right: resolution 256×256.*

suddenly appear due to large time steps, large amounts of potential energy are added and can generate instabilities. In our experiments, explicit integration schemes require small time steps to avoid divergence. Fortunately, the approximate stiffness matrix discussed in Section 3.3 allows us to use an implicit Euler time integration which efficiently dissipates this energy. However, when a fast object such as the tip of the tentacle of the octopus deeply penetrates another object, it sometimes undergoes an unrealistically strong repulsion. More generally, the control of bouncing is not easy using penalty-based repulsion forces. The use of methods derived from constraint-based physics would be an interesting extension. The intersection volume could be seen as a scalar constraint and its gradient as the associated row in the Jacobian matrix. Constraint-based methods would allow us to precisely monitor the intersection, as well as to control bouncing and to model accurate Coulomb friction.

The need for watertight meshes is the only limitation specific to our method. A straightforward extension to two-dimensional objects is possible when all the contacts occur on the same side of the surface, *e.g.* if the surface models the ground, or a bag containing other simulated objects. In this case, the surface can be seen as the boundary of a half-infinite volume.

A nice feature of image-based methods is their ability to provide levels of detail using various resolutions. In our experiments, the optimal LDI resolution depends on the smoothness of the geometry and on the desired stability of the contacts. The discontinuities between the contact forces evaluated at each time step prevent the objects from reaching a perfect rest state. This artifact is not specific to our approach, and the resulting "noise" can be unnoticeable when its amplitude is less than the size of a pixel on the screen. Tuning the resolution allows us to obtain the desired stability. In most of the presented examples, we applied resolution 32×32 to contacts between smooth objects, while 64×64 was used for contacts with sharp objects. Higher resolutions are necessary to model very precise contacts, as shown in the left of Figure 8. The size of the intersection of the bounding boxes of the objects has also an influence on the necessary resolution, because the viewing frustum used to generate the contact images is adjusted on it. For self-collision, the volume includes the whole object, and we apply resolution 256×256 for the octopus in Figure 7. More detailed bounding volumes could allow us to reduce the image resolution. We have measured the stability of our simulator at various resolutions in the case of the intricated rings illustrated in Figure 10. One ring is fixed, while the other swings due to gravity. We have recorded the kinetic energy of the ring at various resolutions, up to less than one pixel per polygon, and verified that the stability increases along with the resolution of the contacts. However, while 32×32 is necessary for a close-up view, it is even possible to model the non-convex contact at resolution 2×2 without divergence. This allows us to modulate the contact resolution based on the distance to the camera, as illustrated in the accompanying video.

6. Conclusion

We have presented a novel volume-based approach to contact detection and reaction. In contrast with most existing methods in the graphics community, it is based on the minimization of the intersection volume between objects. The

intersection volume is defined by pixels modeling its boundaries. The graphics hardware is used to model the intersection volume at desired resolution, thereby freeing the CPU from complex geometrical computations. Self-collisions are easily detected, and the level of detail can be adjusted using image resolution, independently of the geometry. Applied to deformable volume bodies, our new geometrical model of contact combines the robustness of distance maps with the speed of surface-based methods.

We have implemented penalty-based repulsion and viscous friction, and we plan to apply constraint-based repulsion methods to allow the simulation of more complex contact phenomena such as bouncing and Coulomb friction.

Currently, the most important bottleneck of our method is due to the LDI rendering step. We have shown that it is possible to reduce the amount of transferred data from GPU to CPU by performing the detection and computing the forces entirely in the GPU. In the near future, the advances in available power and architecture on GPU as well as more efficient algorithms [BM08] may allow considerable performance improvements.

Acknowledgments

We would like to thank Michael Adam, as well as the rest of the SOFA team for their support, and Laurence Boissieux for graphics models.

References

[ACF*07] ALLARD J., COTIN S., FAURE F., BENSOUSSAN P.-J., POYER F., DURIEZ C., DELINGETTE H., GRISONI L.: SOFA – an open source framework for medical simulation. In *Medicine Meets Virtual Reality (MMVR)* (2007). http://www.sofa-framework.org.

[Bar94] BARAFF D.: Fast contact force computation for nonpenetrating rigid bodies. In *Proc. of ACM SIGGRAPH 94* (1994).

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics 21* (2002), 594–603.

[BJ07] BARBIČ J., JAMES D.: Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. In *Proc. of ACM/Eurographics SCA* (2007).

[BM08] BAVOIL L., MYERS K.: Order independent transparency with dual depth peeling. NVIDIA OpenGL SDK, 2008.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 43–54.

[CW05] CIRAK F., WEST M.: Decomposition-based contact response (dcr) for explicit finite element dynamics. *Int. Journal for Numerical Methods in Engineering 64*, 8 (2005), 1078–1110.

[Eve01] EVERITT C.: *Interactive order-independent transparency*. Tech. rep., NVIDIA Corporation, 2001.

[FL01] FISHER S., LIN M. C.: Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. of IEEE/RSJ IROS* (2001).

[GKLM07] GOVINDARAJU N. K., KABUL I., LIN M. C., MANOCHA D.: Fast continuous collision detection among deformable models using graphics processors. *Computers and Graphics 31*, 1 (2007), 5–14.

[GLM05] GOVINDARAJU N. K., LIN M. C., MANOCHA D.: Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware. In *IEEE Conf. on Virtual Reality* (2005).

[HFS*01] HIROTA G., FISHER S., STATE A., FUCHS H., LEE. C.: An implicit finite element method for elastic solids in contact. In *Proc. Computer Animation* (2001).

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M.: Real-time volumetric intersections of deforming objects. In *Proc. of Vision, Modeling, Visualization (VMV)* (2003).

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. In *Proc. of WSCG'04* (2004), pp. 145–152.

[HTK*04] HEIDELBERGER B., TESCHNER M., KEISER R., MULLER M., GROSS M.: Consistent penetration depth estimation for deformable collision response. In *Proc. of VMV'04* (2004), pp. 339–346.

[JTT01] JIMENEZ P., THOMAS F., TORRAS C.: 3d collision detection: A survey. *Computer and Graphics 25*, 2 (2001), 269–285.

[LM03] LIN M., MANOCHA D.: Collision and proximity queries. In *Handbook of Discrete and Computational Geometry* (2003).

[MC95] MIRTICH B., CANNY J.: Impulse-based simulation of rigid bodies. In *Interactive 3D graphics* (1995).

[OJSL04] OTADUY M. A., JAIN N., SUD A., LIN M. C.: Haptic display of interaction between textured models. In *Proc. of IEEE Visualization Conference* (2004).

[Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garments. In *Proc. of 8th Eurographics Workshop on Animation and Simulation* (1997).

[SGwHS98] SHADE J., GORTLER S., WEI HE L., SZELISKI R.: Layered depth images. In *Proc. of ACM SIGGRAPH 98* (1998).

[SOM04] SUD A., OTADUY M. A., MANOCHA. D.: Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum 23*, 3 (2004), 557–566.

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M.-P., FAURE F., MAGNETAT-THALMANN N., STRASSER W., VOLINO P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61–81.

[VMT00] VOLINO P., MAGNENAT-THALMANN N.: Implementing fast cloth simulation with collision response. In *Computer Graphics International* (2000).

[VMT06] VOLINO P., MAGNENAT-THALMANN N.: Resolving surface collisions through intersection contour minimization. *ACM Transactions on Graphics 25*, 3 (2006), 1154–1159.

[VSC01] VASSILEV T., SPANLANG B., CHRYSANTHOU Y.: Fast cloth animation on walking avatars. In *Proc. Eurographics* (2001), vol. 20.