Towards Robust Animation of Complex Objects in Interaction

Laks Raghupathi

Institut National Polytechnique de Grenoble

THÈSE

pour l'obtentir le grade de

DOCTEUR DE l' INP Grenoble

Spécialité : Mathématiques et Informatique

Préparée au sein du laboratoire *GRAVIR/IMAG-INRIA. UMR CNRS C5527*,
dans le cadre de l' École Doctorale de Mathémathiques, Sciences et Technologies de l' Information, Informatique
présentée et soutenue publiquement

par

Lakshminarasimhan Raghupathi

le 15 novembre 2006

Towards Robust Animation of Complex Objects in Interaction

Directrice de thèse : Mme. Marie-Paule Cani
Co-encadrant : M. François Faure

JURY

M. Augustin	Lux	Président
M. Philippe	Meseure	Rapporteur
M. Yannick	Remion	Rapporteur
M. Laurent	Grisoni	Examinateur
Mme. Marie-Paule	Cani	Directrice de thèse
M. François	Faure	Co-encadrant

SUMMARY

CHAPTER 1

Introduction

Arise, awake, stop not till the goal is reached

- Swami Vivekananda

1.1 Introduction

In the past years, we have seen rapid strides being made in the field of computer graphics. Over these years, the realism of graphics applications have increased by leaps and bounds. No doubt the ever increasing speed and affordability of personal computers equipped with powerful graphics hardware has played an important part as well. There are two fundamental effects of this. First, it has enabled access to powerful hardware in desktop form to more and more researchers themselves which were erstwhile available only to well-funded university and corporate labs. The second is that the industry folks (such as game developers) are able to transfer these research advances to mass entertainment products thus creating a profitable market for these products which in turn sustains further innovation.

The key to maintain/accelerate this virtuous circle is via the constant efforts of the research community in presenting new solutions to open problems. By fundamentally trying to have a better understanding of the mathematical and physical aspects, we have seen several examples of researchers bringing in bold and complex ideas from basic and applied sciences and adapting it to the problem at hand.

Further, such scientific advances not only benefit entertainment technology but also life-saving domains such as surgical planning, training and virtual prototyping. By giving the surgeons an opportunity to practice on a virtual patient before operating, they can have a more accurate understanding of the problem and plan on tackling possible complications which were erstwhile not available. All this means that we need accurate 3D geometric model which accurately captures the organs and a mechanical model which reacts with realism.

1.2 Motivation

The focus of this thesis is physically-based animation. Since the eighties, there have been several attempts to simulate solid, liquid and gaseous phenomena. By the nineties, a great deal of progress was made in the simulation of rigid bodies through pioneering efforts of researchers like David Baraff . We saw how rigid body models which were designed to respect physical laws and involving collisions and contacts can often be too complex to be solved using the "traditional" math tools available at that time. There the graphics community did not hesitate to look for more sophisticated mathematical tools to get an acceptable solution [Bar94].

There have been lots of efforts to replicate such successes in deformable objects. There again, we saw several examples of researchers in the graphics community introducing new models or those adapted from elsewhere. From the pioneering elastic model by Terzopoulos et al [TPBF87], to Witkin's constraints [WW90] and to adaptive multi-resolution models [DDCB01].

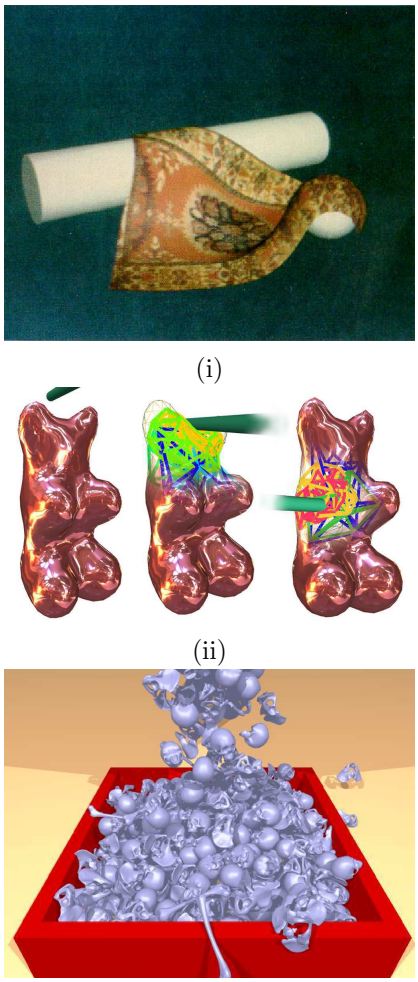
Despite these advances, we believe that the case of multiple collisions and contacts especially for thin deforming objects is a challenging problem at hand. While there have been many earlier models proposed by the graphics research community which led to satisfactory visual results in specific cases - there have been very few models which can boast of a comprehensive approach. In the forthcoming chapters we will convincingly make a case on the need for better solutions for the problems at hand.

1.3 Summary of Contributions

In this thesis, we basically propose a robust model for handling multiple collisions and contacts which is necessary for realistic graphical simulations. We will first examine the state-of-the art methods which exist in literature, then point out the shortcomings when it comes to dealing with the specific problem and then propose our solutions.

1.3.1 Intestine Surgery Simulator

We present a new approach to detect the collisions which occur in highly deformable objects such as the human intestine. Our algorithm which tracks pairs of closest features over



(i) (ii) (iii)

Fig. 1.1: (i) One of the first elastic model of cloth [TPBF87]. (ii) Adaptive multi-resolution deformable model [DDCB01]. (iii) Stacking of many non-convex rigid objects [GBF03].

time-steps is rapid since it does not require expensive bounding volume updates. The collision handling system has been implemented as part of a complete intestinal surgery simulator system which in addition to collisions also consists of modeling, animating, and rendering the intestinal system (rendering and system integration was done by our collaborators at LIFL [1]). The results were published in a peer-reviewed international conference [RCFC03] and subsequently as refereed journal publication [RGF+04].

1.3.2 Robust Mechanical Solver

We present a new way of robustly animating stiff objects such as a mechanical cable colliding with rigid mechanical parts. Our method consists of a mass-spring system integrated using an implicit Euler scheme using an iterative method such as the generic conjugate

[1] GRAPHIX/Alcove, Laboratoire d'Informatique Fondamentale de Lille, 59655 Villeneuve d'Ascq Cédex, FRANCE, `http ://www2.lifl.fr/GRAPHIX/`

gradient solver or a more direct and specific technique such as a banded LU solver. We use an efficient octree-based bounding volume hierarchical system to quickly identify the zone of collision and detect and respond to collisions both at continuous and discrete time intervals. We present our results using practical examples with 3D mechanical parts and cable specifications from an industrial partner Solid Dynamics [2] - a developer of commercial CAD/CAM software.

1.3.3 Continuous-Time Collision Detection

With the help of specific case studies, we illustrate the shortcomings of detecting collisions *only* at discrete time-intervals - it becomes acute while dealing with thin objects. We elucidate the need to detect at continuous time intervals for simulating dynamic thin objects. With this approach, we are able to not only catch all the collisions missed otherwise, but also able to run simulations at relatively large time steps. Though continuous collisions were proposed earlier, our approach is robust since it even handles degenerate cases. We would later combine both these techniques in addition to exploiting temporal coherence for handling "difficult" collision case of simultaneous multiple collisions. Some of the preliminary results were demonstrated in a tutorial at an international conference [ZTK+05]. A state-of-the-art report on collision detection techniques for deformable objects co-written with other researchers interested in this domain was published as a refereed journal publication [TKH+05].

1.3.4 Quadratic Programming Collide

The robust detection of collisions solves only one part of a complex problem. In addition, there are also cases where the need for handling multiple collisions and contacts arise. Here we present two approaches : first we have developed a novel approach which formulates the problem of multiple constraints as a quadratic programming (QP) problem - by considering the collisions as linear constraints and the underlying dynamics as an objective function to be minimized. These constraints directly modify the velocities of the colliding elements thus avoid introducing excessive strain into the system. Finally, we are able to obtain a global solution which is able to satisfy all the collisions. The preliminary results were published in a French conference paper [RF06].

1.3.5 Guaranteed Collision Response

Secondly, we also propose a "fail-safe" method which ensures that no collisions are missed. This method largely relies on exploiting temporal coherence by introducing penalty springs in response to collisions detected *both* at discrete and continuous time-steps. By increasing stiffness value in case of persistent interpenetration we are able to "break-free" of the collision loop which is one of the bane of the iterative methods. Our method also monitors if new

[2] Solid Dynamics S.A., 42300 Roanne, FRANCE, `http ://www.solid-dynamics.fr`

collisions are created while responding to the existing ones, thus making it a truly fail-safe approach.

1.4 Organization of Thesis

After having introduced the problem at hand, we detail the existing techniques in the domain of physically-based animation, collision detection and response in chapter 2. In chapter 3, we present new techniques developed for an intestinal surgery simulator and a stiff mechanical cable system in addition to discussing the shortcomings of the approaches. Then we present our robust collision detection techniques in chapter 4. We present our solutions for handling multiple collisions in chapter 5. Finally, we summarize our contributions and conclude with some perspectives on future work in this research area in chapter 6.

CHAPTER 2

State of the Art

Jim Blinn in his SIGGRAPH '98 keynote address has identified the simulation of spaghetti as one of the ten unsolved problems in computer graphics [Bli98]. The main research issues here are the modeling, detection of multiple collisions and the simulation of a realistic response to that. This chapter reviews some of the prior work in this area.

2.1 Introduction

Not all animation which looks realistic uses a physics-based approach. For example in [Bar97], Barzel describes a method of "fake" dynamics which was used for the full-length CG animated movie *Toy Story* [Las95]. Such applications are intended to animate creatures such as *Slinky Dog* (see Fig. 2.1(i)) in a non-physical and dramatic manner. Since then, we have come a long way in the use of "realistic" looking computer generated characters which drive popular imagination. For example the physically-based cloth animation systems developed by [BFA02] was used to animate the virtual robe of a computer generated *Yoda* (see Fig. 2.1(ii)) in the feature film *Star Wars : The Phantom Menace* [Luc99]. Of course, they still do not provide the perfect results required in a production environment and so artistic tweaks and re-simulations are often required. But what used to be earlier hand-animated is increasing being automatized thanks to the increasing availability of affordable workstations for graphics artists and production engineers powered by sophisticated modeling and anima-

tion software. As we noted in the previous chapter, such advancements have been possible due to the pioneering efforts of computer graphics researchers whose work gets translated into sophisticated tools which aid artists and production engineers.

In this chapter, we briefly trace the developments in the physics-based simulation in computer graphics concentrating on aspects of physically-based animation (cf. § 2.2), collision detection (cf. § 2.3) and collision response (cf. § 2.4).

(i)

(ii)

Fig. 2.1: (i) *Slinky Dog* from the animated feature film *Toy Story* ©Disney/Pixar 1995. (ii) *Yoda* from *Star Wars : Episode I* ©Lucasfilm Ltd. 1999.

2.2 Physically-based Modeling

Early Works : One of the pioneering works in the use of physical laws for simulating deformable objects was by Terzopoulos et al. [TPBF87]. This preliminary model was later extended to take account of visco-elasticity, plasticity and a basic model for fracture in [TF88a, TF88b]. This was followed by the constraint-based approach for solving dynamics problems by Witkin and Welch [WW90] and Baraff and Witkin [BW92] for simulating flexible objects. During this period, several interesting approach were proposed for simulating rigid body dynamics using analytical methods [Hah88, Bar89], to take account of friction [Bar91] and formulating a global computation of non-penetration forces [Bar94].

Deformable Objects : In the later years, more realistic approaches based on finite element analysis [Bat95] began to be applied in computer graphics. Advanced methods which include fast boundary element approaches [JP99], implicit surface formulation [DC95], adaptive multi-resolution techniques [DDCB01], [CGC+02] and [GKS02] provided innovative approaches which were fast-enough to run on standard PCs while giving a realistic solution. Some with special applications such as surgery simulation used finite element preprocessing [CDA99]. More generic approaches exploiting pre-computation were proposed using reduced coordinate models [JF03, BJ05]. Other volumetric approaches such as the 3D Chainmail by Gibson [Gib97], mesh-less method based on finite spheres by De and Bathe [DB00] and point-based methods by Pauly et al. [PKA+05] too have been developed.

Cloth Animation : In deformable object simulation, a popular challenge (driven in part by film and computer games industry) was to realistically simulate cloth for character animation. Initially particle-based approaches were proposed by Breen et al [BHG91]. Provot [Pro95] further advanced this by proposing a mass-spring system using an explicit integration approach. The undesirable "super-elastic" effect occurring due to explicit methods was fixed by a post-integration deformation constraint (with a user-defined value) step. Volino et al. [VCMT95] further extended this approach to take account of collision and self-collisions occurring by checking the colliding particle's orientation - an attractive force is applied if it is "wrong" side as opposed to the usual repulsive force if it is on the "right" side. Finally, it was Baraff and Witkin who pioneered the efficient use of implicit integration [BW98] for a "stiff" material like cloth. The implicit step though more complex than an explicit one nevertheless also permitted the use of large time-steps. Note that the early approach of Terzopoulos [TPBF87] too suggested the use of implicit integration approach - but they relied on a direct solver which is not very efficient for large models. Hence they relied on explicit schemes for such cases. Baraff solved this by proposing an iterative solver such as the conjugate gradient method. A comparison of various explicit and integration was presented in a study by Volino and Magnenat-Thalmann [VMT01]. Recently, a more advanced form of the conjugate gradient algorithm for cloth animation has been presented by Ascher and Boxerman [AB03]. One problem with the approach of Baraff and Witkin is that the implicit scheme "smoothes" out folds and wrinkles. To overcome that, further advances were made in cloth simulation with the treatment of buckling effects by Choi and Ko [CK02]. Another approach to efficiently simulate cloth while preserving the folds and wrinkles was proposed by Bridson et al. [BMF03]. It uses a hybrid explicit/implicit integration scheme originally proposed by Meyer et al. [MDDB00].

Strand and Hair Animation : Pai introduced the mechanics of simulating thin strands such as surgical sutures, hair, ropes, etc. via a static method of Cosserat's rods [Pai02]. The emphasis here is to capture the twisting behavior when one applies a torque along the axis of the strand. The author questioned the approach of using existing techniques such as FEM, mass-spring for modeling such objects since they will require very fine meshes in order to well-represent the curvature. Hence, he proposed to model thin objects by using Cosserat's

theory erstwhile used in solid mechanics. The resulting mathematical formulation followed by the discretization results in an ODE in one independent variable. The results here are 30 Hz with a few hundred points. The present work neither addresses the issue of time-stepping for dynamic simulations nor collisions. Bertails et al. [BAC+06] recently extended the use of Cosserat's rods to dynamic cases and interacting situations through her *Super Helices* used to simulate hair strands.

Yet another approach to hair animation is by using continuum dynamics principle [HMT01]. Their work was inspired by smoothed particle hydrodynamics (SPH) which was first introduced to computer graphics by Desbrun and Cani [DC96] for simulating highly deformable objects. Here they animated hair using a set of articulated rigid bodies to compute the next position of hair strands. To account for hair-hair interaction they viewed the hair as a set of fluid particles based on SPH. The computed fluid forces were then applied to the articulated rigid bodies.

Hair mutual interactions was one of the main focus of [PCP01] where they developed the first model that computed the interactions inside hair, both with a hair wisp and between wisps. Chang et al. [CJY02] too later addressed hair-hair collision by using a set of sparse guide strands with a set of auxiliary triangles. Dense hair is then generated from this sparse model by using an interpolation scheme. For hair-hair interaction, a triangle strip is generated by connecting the hair vertices. Collision is detected when the distance between the hair elements falls below a threshold.

For a detailed treatment of the various techniques for hair simulation, we refer to the survey paper by Ward et al. [WBK+06]. A detailed state-of-the-art report in the general domain of physically-based modeling is presented by Nealen et al. [NMK+05].

2.3 Collision Detection

Collision detection is one of most interesting topics in computer graphics in general and in physically-based animation in particular. As the complexity of graphics applications increase, we have a large amount of interacting objects in the scene. And it is very important to detect and respond to them in order to maintain the realism of the simulation. Overall, collision detection consists of determining when a geometric intersection is going to occur or if it has already occurred. In this section we describe the basic tenets of collision detection and the various approach one can take. Specifically, we describe the following popular approaches to collision detection :

- Bounding Volume Hierarchy
- Spatial Subdivision
- Distance Fields
- Image-Space Techniques

Of these, we describe bounding volume hierarchy in more detail than others. We then detail the narrow phase techniques especially the continuous-time methods. For other good introduction to the various techniques in collision detection, we refer the reader to the survey papers by Lin and Gottschalk [LG98], Jiménez et al. [JTT01] and more recently by Teschner et al. [TKH+05] which specifically addresses deformable object collision detection.

2.3.1 Broad Phase vs Narrow Phase

Broad phase collision detection is an approach to quickly get rid of most non-colliding objects (or regions within an object) using a relatively inexpensive test. It might usually consists of developing data structures (such as trees, distance fields or spatial-partitioning methods) which will quickly identify possible zones of contact. In the broad phase, collision test is accelerated by performing collisions between the BVH rather than the actual polygonal data itself (see Fig. 2.2). Note that such object representations are commonly built in a pre-processing stage and need not be updated over the time for rigid body animations. But they need to updated for deformable cases. There are several update strategies proposed in graphics literature (see §2.3.3.3). We then perform exact tests between the primitives (triangles, vertices, edges) referred to as the *narrow phase* of the detection to find the exact point of collision. This information is then passed to the collision response module.

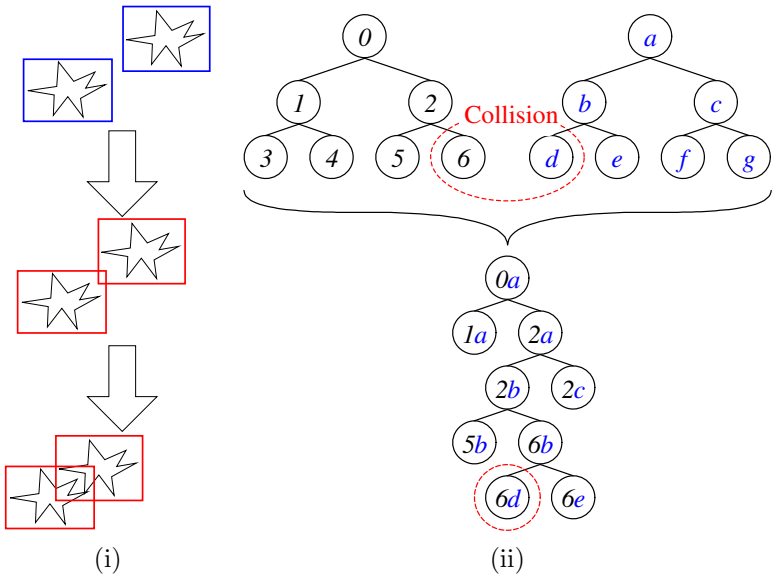


Fig. 2.2: *Broad phase* collision detection illustrated (i) geometrically and (ii) graphically.

2.3.2 Basic Tenets of Primitive Testing

Before we detail the broad phase techniques, we would like to specify the the kind of geometric objects we cover. In this thesis, we generally deal with polygonal objects. Hence the primitives of such objects are vertices, edges, faces (or its special cases - triangles). In one of the earliest work, Boyse [Boy79] discusses his method of interference detection between

objects represented as a polyhedra. Per this, there are three possibilities when two polyhedra A and B intersect (see Fig. 2.3). They are :

- Vertex of A intersects with Face of B
- Vertex of B intersects with Face of A
- Edge of A intersects with Edge of B

Note that the primitive testing comes in the narrow-phase of the testing.

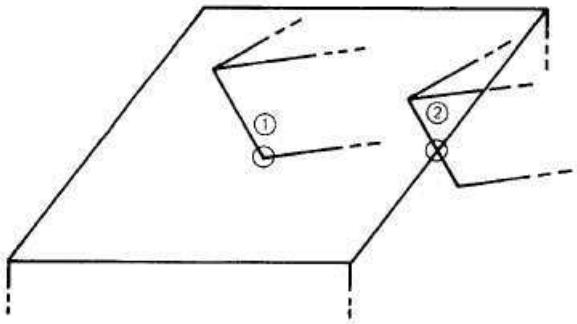


Fig. 2.3: Basic tenets of collision detection [Boy79].

2.3.3 Bounding Volume Hierarchies

Bounding-volume hierarchies (BVHs) have proven to be one of the efficient data structures for collision detection. The characteristics of different hierarchical collision detection algorithms lie in the type of BV used, the algorithm for construction of the BV trees and the overlap test for a pair of nodes. The idea behind a BVH is to partition a set of primitives that constitutes a given object recursively until some leaf criterion is met. Most often, each leaf contains a single primitive, but the leaf criterion could also be met if a node contains less than a fixed number of primitives. Here, primitives are the entities which make up the graphical objects, which can be polygons. As we mentioned earlier, there can be other primitives such as NURBS patches but in this section, we consider mostly polygonal primitives, i.e. vertices, faces and edges. A BVH is commonly constructed for each object in a pre-processing step. In general, BVHs are defined as follows : Each node in the tree is associated with a subset of the primitives of the object, together with a BV that encloses this subset with a smallest containing instance of some specified class of shapes. We refer to [ZL03] for a detailed discussion of BVHs in general.

BVH Types : One of the design choices with BV trees is the type of BV. In the past, a wealth of BV types has been explored, such as spheres [PG95, Hub96] and more recently by Bradshaw and O' Sullivan [BO02], oriented bounding boxes (OBB) [GLM96], discrete oriented polytopes (DOP) [KHM$^+$98], Boxtrees [AdG$^+$02], axis-aligned bounding boxes (AABB) [vdB97, LAM01], spherical shells [KGL$^+$98], and convex hulls [EL01].

Although a variety of BVs has been proposed (see Fig. 2.4), two types deserve special

mention : OBBs and k-DOPs. Note, that AABBs are a special case of k-DOPs with $k = 6$. OBBs have the nice property that, under certain assumptions, their tightness increases linearly as the number of polygons decreases [GLM96]. k-DOPs, on the other hand, can be made to approximate the convex hull arbitrarily by increasing k. Further, k-DOPs, especially with $k = 6$, can be computed very efficiently. This is important, since deforming objects require frequent updates of a hierarchy (cf. §2.3.3.3). Also note that the performance of a BVH query depends on both the depth of the hierarchy and the cost of the query performed at a given level of the hierarchy.

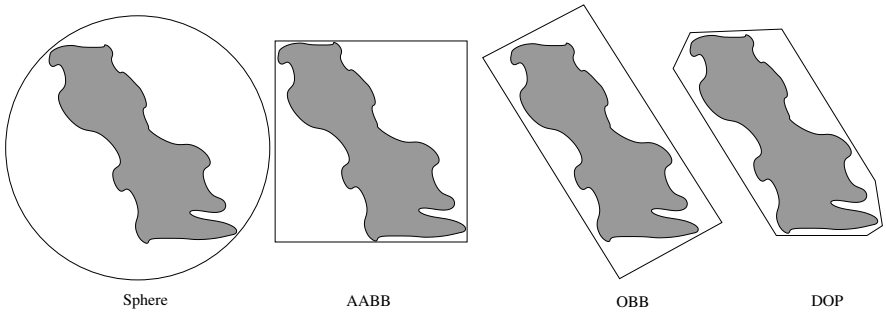


Fig. 2.4: A variety of bounding volumes has been proposed for hierarchy-based collision detection.

2.3.3.1 Hierarchy Construction

As far as collision detection is concerned, the goal is to construct BVHs such that any subsequent collision query can be answered as fast as possible. Such BVHs are called optimal or good in the context of collision detection. The BVH is built by recursively splitting a set of object primitives until a threshold is reached. The splitting is guided by a user-specified criterion or heuristic that will yield good BVHs with respect to the chosen criterion. There exist three different strategies to build BVHs, namely top-down, bottom-up [RL85], and insertion [GS87]. However, the top-down strategy is most commonly used for collision detection. Fig. 2.5 shows two hierarchy levels for the 18-DOP hierarchy of an avatar, that was created top-down [MKE03].

A very simple splitting heuristic is the following. [GLM96] approximated each polygon by its center. Then, for a given set B of such points, they computed its principal components (the eigenvectors of the covariance matrix), chose the largest of them (i.e. the one exhibiting the largest variance) and then placed a plane orthogonal to that principal axis and through the barycenter of all points in B. This effectively split B into two subsets. Alternatively, the splitting plane can be placed through the median of all points. This leads to a balanced tree. However, it is unclear, whether balanced trees provide improved efficiency of collision queries.

With deformable objects, the main goal is to develop algorithms that can quickly update or refit the BVHs after a deformation has taken place. At the beginning of a simulation, a good BVH is constructed for the initially non-deformed object just like for rigid bodies. Then, during the simulation, often times the structure of the tree is kept, and only the extents of

the BVs are updated. Due to the fact that AABB (or in general k-DOPs) are generally faster to update for deformable objects as described in [vdB97], they are preferred to OBBs (cf. §2.3.3.3 for a discussion on update strategies).

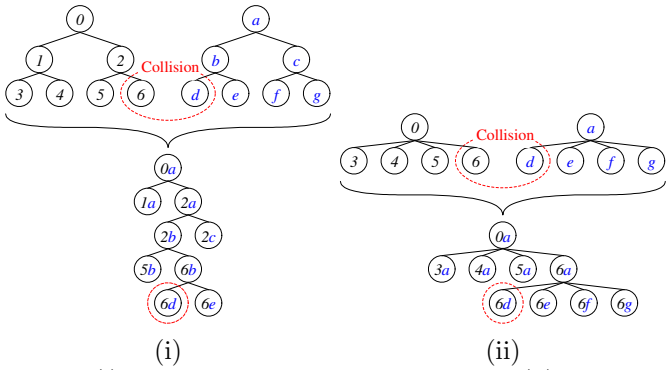
Fig. 2.5: Two levels of an 18-DOP hierarchy [MKE03].

2.3.3.2 Hierarchy Traversal

For testing the collision between two objects or the self collision within a single object, the BVHs are traversed top-down and pairs of tree nodes are recursively tested for overlap. If the overlapping nodes are leaves then the enclosed primitives are tested for intersection. If one node is a leaf while the other one is a internal node, the leaf node is tested against each of the children of the internal node. If, however, both of the nodes are internal nodes, it is tried to minimize the probability of intersection as fast as possible. Therefore, [vdB97] tests the node with the smaller volume against the children of the node with the larger volume (see Fig. 2.6). For two given objects with the BVHs A and B, most collision detection algorithms implement the following general algorithm scheme (see Algo. 1). This algorithm quickly "zooms in" on pairs of nearby polygons.

2.3.3.3 Hierarchy Update

In contrast to hierarchies for rigid objects, hierarchies for deformable objects need to be updated in each time step. Principally, there are two possibilities : *refitting* or *rebuilding*. Refitting is much faster than rebuilding, but for large deformations, the BVs usually are less tight and have larger overlap volumes. Nevertheless, van den Bergen [vdB97] found out that refitting is about ten times faster compared to a complete rebuild of an AABB hierarchy.



(i) (ii)

Fig. 2.6: (i) Recursion using binary trees and (ii) 4-ary trees.

Algorithm 1 BVH Traversal

traverse(A,B)
if A and B do not overlap **then**
  return
end if
if A and B are leaves **then**
  return intersection of primitives
    enclosed by A and B
else
  for all children A_i and B **do**
    traverse(A_i,B)
  end for
end if

Further, as long as the topology of the object is conserved, there is no significant performance loss in the actual collision query compared to rebuilding.

The overall strategy is to update as few nodes as possible. The hierarchies can be updated either by bottom-up, top-down or hybrid strategy. In the top-down approach, we scan all the primitives under a node starting from the root and update the boundaries of the current node. The same is selectively applied for child nodes when needed. This way we only update few nodes as necessary which is relatively faster for "simple" cases when not many deep nodes are reached. In the case of bottom-up, the hierarchies are traversed upwards starting from the leaves merging towards the root. Since we already have the bounding volume of child nodes, finding the bounding volume of parent node is trivial $BVH_{\text{parent}} = \bigcup_i BVH_i, \forall i \in BVH_{\text{child}}$. But the disadvantage is all the nodes need to be traversed. The bottom-up approach is relatively faster for more "difficult" cases when many deep nodes are reached during collision tests.

Other approaches have been proposed by Mezger et al. [MKE03] to further accelerate the hierarchy update by omitting or simplifying the update process for several time steps. For this purpose the bounding volumes can generally be inflated by a certain distance. Then the hierarchy update is not needed as long as the enclosed primitives did not move farther than

that distance.

Hybrid Update : Larsson and Akenine-Möller [LAM01] compared bottom-up and top-down strategies. They found that if many deep nodes are reached the bottom-up strategy performs better, while if only some deep nodes are reached the top-down approach is faster. Therefore they proposed a hybrid method, that updates the top half of the tree bottom-up and only if non-updated nodes are reached these are updated top-down (see Fig. 2.7). Using this method they reduce the number of unnecessarily updated nodes with the drawback of higher memory requirement because they have to store the leaf information about vertices or faces also in the internal nodes.

Another crucial point is the arity of the BVH. For rigid objects, binary trees are commonly chosen. In contrast, 4-ary trees or 8-ary trees have shown better performance for deformable objects [LAM01, MKE03]. This is mainly due to the fact that fewer nodes need to be updated and the total update costs are lower. Additionally, the recursion depth during overlap tests is lower and therefore the memory requirements on the stack are lower.

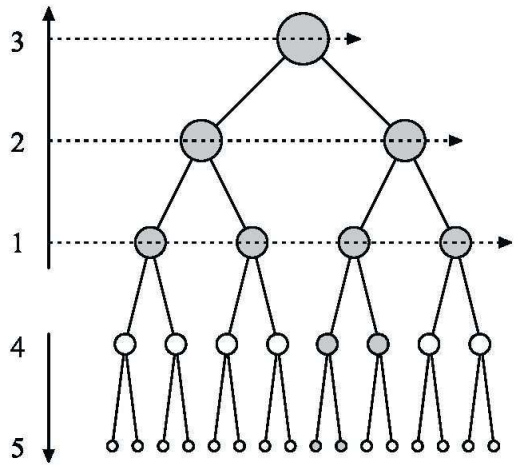


Fig. 2.7: Example of a hybrid update combining the bottom-up and top-down strategy [LAM01].

2.3.3.4 Self-Collision Detection Using BVH

In general, collisions and self-collisions are performed the same way using BVHs. If several objects are tested for collisions, the respective BVHs are checked against each other. Analogously, self-collisions of an object are detected by testing a BVH against itself.

However, it has to be noted, that BVs of neighboring regions can overlap, even when there are no self-collisions. To eliminate such cases efficiently, different heuristics have been presented. Volino and Magnenat-Thalmann [VMT94, VMT95] proposed an exact method to avoid unnecessary self-intersection tests between certain BVs. In a region, if there exists a vector $\mathbf{v}$ such that $\mathbf{v} \cdot \mathbf{n}_i > 0$ for all normal values $\mathbf{n}_i$, then cannot be any self-collisions within the region. If such a vector exists and the projection of the region onto a plane in direction of the vector does not self-intersect, then there can be no self-intersections in the

entire region. Also, if the contour of the region projected on a 2D plane does not self-intersect, then there cannot be any self-collisions. Hence regions which does satisfies these condition are exempt from collision checking. The same philosophy is applied between two adjacent regions (connected by at least one vertex) to check for non-intersection conditions.

A faster approach was proposed by Provot [Pro97] in which normal cones were introduced. The idea is based on the fact, that regions with sufficiently low curvature cannot self-intersect, assuming they are convex. Therefore, a cone is calculated for each region. These cones represents a superset of the normal directions. They are built using the hierarchy and updated during the hierarchy update. The apex angle α of the cone represents the curvature, indicating possible intersections if $\alpha \geq \pi$. The cones are built bottom starting from the leaf node whose $\alpha = 0$. Then moving up the cones of the top node is computed using the normal and cone angle values of the descendants ($\mathbf{n}_1$, $\mathbf{n}_2$, α_1 and α_1 respectively) as follows (see Fig. 2.8) :

$$\beta = \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2) \tag{2.1}$$

$$\alpha = \frac{1}{2}\beta + \max(\alpha_1 + \alpha_2) \tag{2.2}$$

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2}{|\mathbf{n}_1 + \mathbf{n}_2|} \tag{2.3}$$

Of course this assumes that the descendant cones are also adjacent which is the case for convex situations. Self-collisions within a cone can then be pruned if $\alpha < \pi$.

2.3.3.5 Wrapped Hierarchy

Guibas et al. [GNRZ02] proposes a method for the collision detection for deforming necklaces - objects that can be modeled as a chain of connected spheres. Naturally, a sphere-based hierarchical method is used for collision and self-collision detection. The interesting aspect of this work is the use of a wrapped hierarchy. Traditional bounding volumes such as OBB and AABB have been successfully applied for large virtual environments with physical simulations. However, they are slow to update when the object not only moves, but also deforms. This is because they are based on *spatial proximity* which changes with time. The proposed method is based on *topological proximity*, which is preserved even under deformation. Under the new approach, the upper bound for determining interference between two bounding hierarchies is $O(n^{2-d/2})$ for d-dimensions. But this collision model is yet to be integrated with a physical model which can govern the dynamics of the objects. Again, only small deformation have been considered. This cardinal rule applies for *both* rigid and deformable objects.

James and Pai [JP04] too exploited wrapped hierarchy with their BD-Tree which efficiently handles hierarchy update for a reduced coordinate animation model which undergoes limited deformation. The BD-Tree hierarchy is first constructed using any standard technique. It is then "wrapped" by tightening the radius (in case of a sphere tree) while retaining the center (see Fig. 2.9). The father node here usually covers only the primitives under its

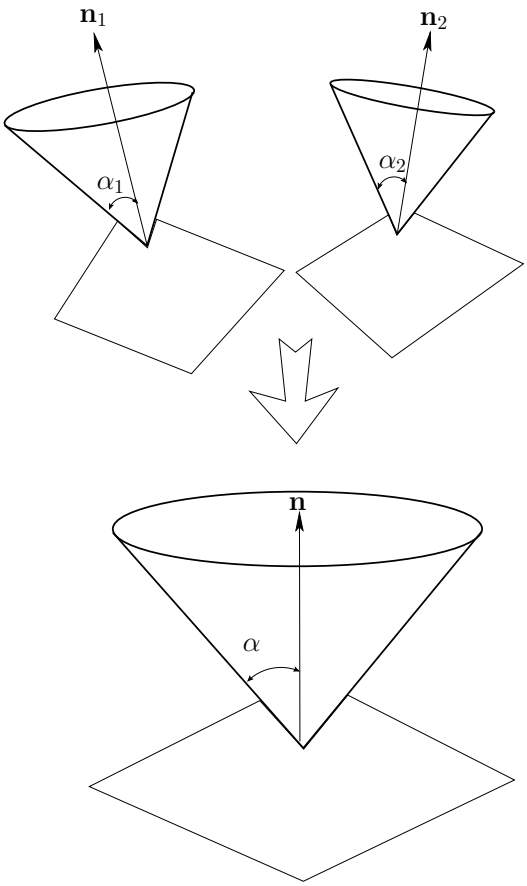


Fig. 2.8: Cone (angle α) enclosing two descendant cones in the hierarchical tree (angles α_1 and α_2) [Pro97].

hierarchy and not the actual child nodes. This avoids the need to update the father node when the child needs to be updated. But they also slightly inflate the coarser nodes in order to wrap the descendant nodes as well. A fast update scheme which takes advantage of the reduced coordinate formulation is presented for the center and the radius. Though conservative, it performs well for limited deformation. But it is not suitable for larger deformation since the calculated radius becomes very conservative and hence very large.

2.3.3.6 BVH Summary

In BVH approaches, the efficiency of the basic BV has to be investigated very carefully. This is due to the fact that deforming objects require frequent updates of the hierarchy. So far, it has been shown that AABBs should be preferred to other BVs, such as OBBs. Although OBBs approximate objects tighter than AABBs, AABBs can be updated or refit very efficiently. Additionally, 4-ary or 8-ary trees have shown a better overall performance compared to binary trees.

Although deformable modeling environments require frequent updates of BVHs, BVHs are nevertheless well-suited for animations or interactive applications, since updating or refitting

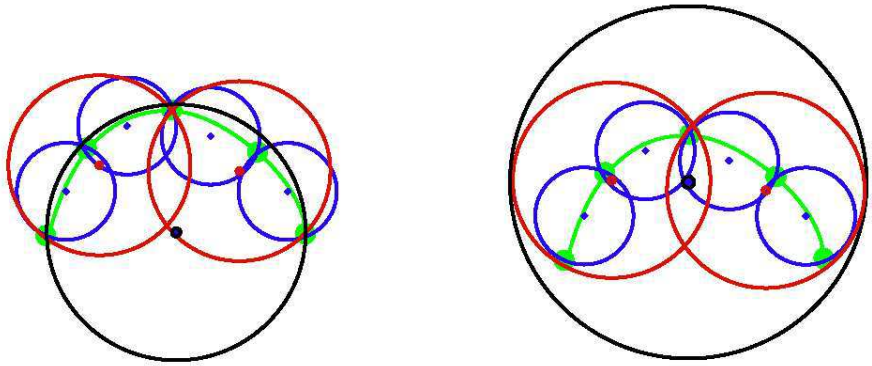


Fig. 2.9: The wrapped hierarchy (left) has smaller spheres than the layered hierarchy (right). Note that the wrapped hierarchy at a certain level need not contain the spheres of its descendants and so can be significantly smaller. But since they do contain the actual geometry (shown in green), it is sufficient for collision detection [JP04].

of these hierarchies can be done very efficiently. Furthermore, BVHs can be employed to detect self-collisions while applying additional heuristics to accelerate this process. Also, BVHs work with triangles and tetrahedrons as object primitives, which allows for a more sophisticated collision response compared to a pure vertex-based response.

2.3.4 Spatial Subdivision

Spatial subdivision is a simple and fast broad phase technique to accelerate collision detection for both rigid and deformable objects. We divide the space into cells and place each object (or bounding volume) in the cells(s) they intersect. We check collisions by examining the cells occupied by each bounding volume to verify if the cells are shared by other objects (see Fig. 2.10). Algorithms based on spatial subdivision are independent of topology changes of objects. They are not restricted to triangles as basic object primitive, but also work with other object primitives if an appropriate intersection test is implemented.

There exist various approaches that propose spatial subdivision for collision detection. These algorithms employ either uniform grids [Tur90, GDO00, ZY00] or binary space partitions (BSP) [Mel00]. In [Tur90], spatial hashing for collision detection is mentioned for the first time. In [Mir97], a hierarchical spatial hashing approach is presented as part of a robot motion planning algorithm, which is restricted to rigid bodies. France et al. [FLMC02] used a grid-based approach for detecting self-collisions of the intestine and collisions with its environment. All objects were first approximated by bounding spheres, whose positions were stored, at each time step, in the 3D grid. Each time a sphere was inserted into a non-empty voxel, new colliding pairs were checked within this voxel. Though this method achieved real-time performances when the intestine alone was used, it failed when a mesentery surface was added. The main difficulty in spatial subdivision is the choice of the data structure that is used to represent the 3D space. This data structure has to be flexible and efficient with respect to computational time and memory.

More recently, in [THM+03], spatial hashing is employed for the detection of collisions and self-collisions for deformable tetrahedral meshes. Tetrahedral meshes are commonly used in medical simulations, but can also be employed in any physically-based environment for deformable objects that are based on FEM, mass-spring, or similar mesh-based approaches. This algorithm implicitly subdivides $\mathbf{R}^3$ into small grid cells. Instead of using complex 3D data structures, such as octrees or BSPs, the approach employs a hash function to map 3D grid cells to a hash table. This is not only memory efficient, but also provides flexibility, since this allows for handling potentially infinite regular spatial grids. Information about the global bounding box of the environment is not required and 3D data structures are avoided.

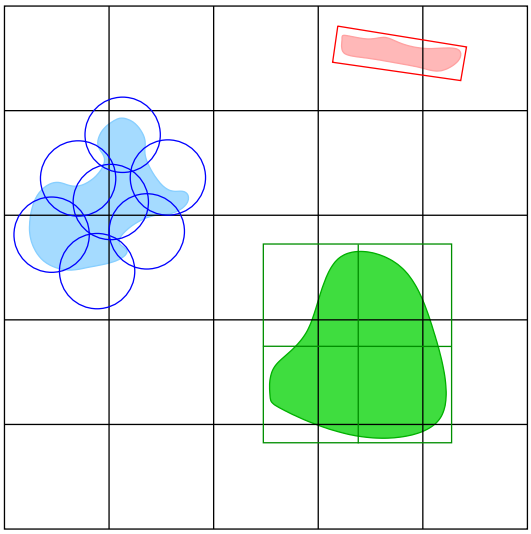


Fig. 2.10: Spatial subdivision technique showing objects in a scene and their bounding volumes in an uniform grid.

2.3.5 Distance Fields

Distance fields specify the minimum distance to a surface for all points in the field. The distance may be signed in order to distinguish between inside and outside. Representing a closed surface by a distance field is advantageous because there is no restriction on the topology. Further more, the evaluation of distances and normals needed for collision detection and response is extremely fast and independent of the complexity of the object. Besides collision detection, distance fields have a wide range of applications. They have been used for morphing [BMWM01, COSL98], volumetric modeling [FPRJ00, BPK+02], motion planning [HKL+99] and recently for the animation of fire [ZWF+03]. Distance fields are sometimes called distance volumes [BMWM01] or distance functions [BMF03].

For collision detection, distance fields are particularly well-suited in virtual garments applications. Here a static mannequin can be represented by a distance fields both inside and outside the body (see Fig. 2.11). Fuhrmann et al. [FSG03] tackled the problem of rapid distance computation between rigid and deformable objects. Rigid objects are represented by

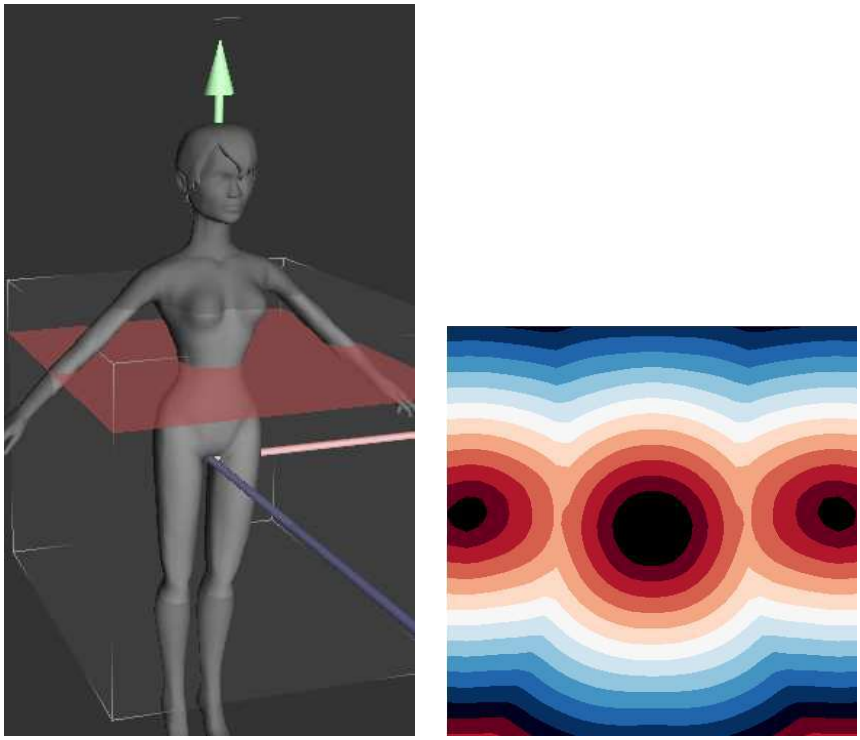


Fig. 2.11: Model of a mannequin with a cutting plane (left). 2D Distance fields generated along this plane viewed from top shown in different colors (right). Image Courtesy : James Withers, EVASION/GRAVIR.

distance fields, which are stored in a uniform grid to maximize query performance. Vertices of a deformable object, which penetrate an object, are quickly determined by evaluating the distance field. Additionally, the center of each edge in the deforming mesh is tested in order to improve the precision of collision detection (see Fig. 2.12). Here, collisions with a complex non-convex objects and self-collisions were not handled using distance fields. Bridson et al. [BMF03] too used an adaptive distance function and a fast local update algorithm for detecting cloth-object collisions.

Collision detection between two deformable objects is carried out by comparing the vertices of one object to the distance field of the other and vice versa. Of course, as the objects deform updates of distance fields are particularly expensive. Fisher and Lin [FL01] estimated the penetration depth for deformable volumetric objects simulated using FEM. They used a fast level set method which internally propagated an initially pre-computed distance field and then partially updated it after each time step as the object deformed. The update is done partially in the sense that only those lying in the colliding regions are updated (returned by a hierarchical sweep and prune method in combination with an AABB-based BVH). This will avoid the complete update or re-computation of the distance field which are expensive. However, even this fast method is not applicable for detecting self-collisions within thin objects such as the cloth or hair since the approximation cannot faithfully represent the change in object shape.

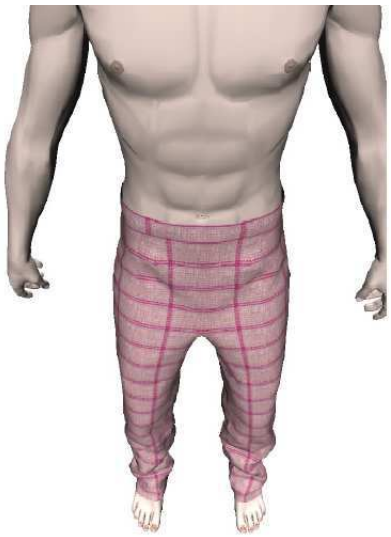


Fig. 2.12: Collision between a trouser and a mannequin using distance fields [FSG03].

2.3.6 Image-Space Techniques

Recently, several image-space techniques have been proposed for collision detection [MOK95, BWS99, LCN99, IZLM01, BW02, KOLM02, HTG03, KP03, GRLM03]. These approaches commonly process projections of objects to accelerate collision queries. Since they do not require any pre-processing, they are especially appropriate for environments with dynamically deforming objects. Furthermore, image-space techniques can commonly be implemented using graphics hardware. However, due to buffer read-back delays and the limited flexibility of programmable graphics hardware, it is not always guaranteed that implementations on graphics hardware are faster than software solutions in all cases (see [HTG04]). As a rule of thumb, graphics hardware should only be used for geometrically complex environments.

An early approach to image-space collision detection of convex objects has been outlined in [SF91]. In this method, the two depth layers of convex objects are rendered into two depth buffers. Now, the interval from the smaller depth value to the larger depth value at each pixel approximately represents the object and is efficiently used for interference checking. A similar approach has been presented in [BWS99]. Both methods are restricted to convex objects, do not consider self-collisions, and have not explicitly been applied to deforming objects.

In [MOK95], an image-space technique is presented which detects collisions for arbitrarily-shaped objects. In contrast to [SF91] and [BWS99], this approach can also process concave objects. However, the maximum depth complexity is still limited. Additionally, object primitives have to be pre-sorted. Due to the required pre-processing, this method cannot efficiently work with deforming objects. Self-collisions are not detected.

Lombardo et al. [LCN99] proposed one of the first image-space approach to collision detection in surgery simulation applications. Here they developed a method to address the detection of collisions between a rigid surgical tool and a deformable organ model in a surgical simulation environment. It exploited GPU-based computation by using the OpenGL ®

clipping process for collision detection. A surgical tool such as a grasper or cautery tool is simple enough to be modeled as a orthographic or perspective viewing volume with clipping planes. Thus, by rendering in feedback mode, they identified the triangles of the organ that are "visible" to this volume as the colliding ones. This simply provided a static collision test (see Fig. 2.13 (i)) when the tool is stationary. A dynamic collision detection taking into account the volume covered by the tool between consecutive time steps too was proposed (see Fig. 2.13 (ii)). Though, the results of this method (see Fig. 2.13 (iii) and (iv)) are extremely fast, it is very specific applicable only to simple object shapes such as cylinders.

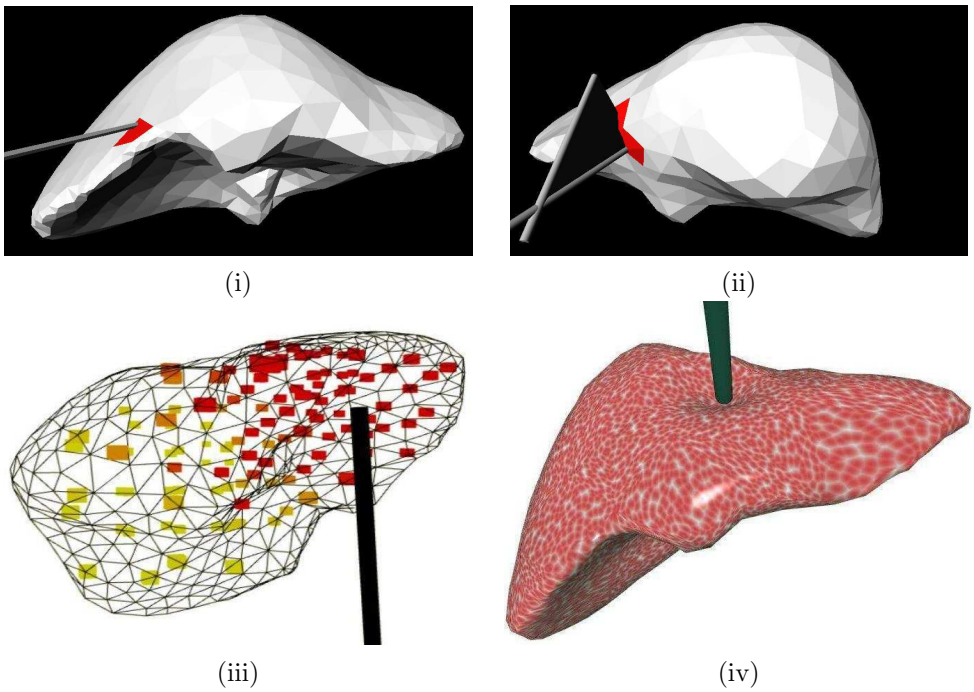


Fig. 2.13: Illustration of a fast, OpenGL clipping-based collision detection method in virtual liver surgery using a standard PC [LCN99]. (i) Collision detected (the dark patch) at a given time step when the tool is static. (ii) Dynamic collision detection by sweeping the viewing volume over subsequent time steps as the tool probes the organ. (iii) Dynamic simulation with input from the collision response driving a physically-based model. (iv) Final textured image.

A first application of image-space collision detection to dynamic cloth simulation has been presented in [VSC01]. In this approach, an avatar is rendered from a front and a back view to generate an approximate representation of its volume. This volume is used to detect the penetrating cloth particles. In [IZLM01], an image-space method is not only employed for collision detection, but also for proximity tests. However, this method is restricted to 2D objects. In [KOLM02] and [KmLM02], closest-point queries are performed using bounding-volume hierarchies with a multi-pass rendering approach. Baciu and Wong's work [BW02] is on the lines of Provot [Pro97] for modeling self-collisions in deformable surfaces. It uses the pixel buffer of the graphics hardware to accelerate and collision and self-collision detection tests for deformable meshes represented by triangular surfaces using Provot's $(\pi, \beta)-$surfaces to find out regions of high-collision (discussed in §2.3.3.4).

In [KP03], edge intersections with surfaces were detected in multi-body environments. This approach is very efficient though it is not robust in case of occluded edges. In [GRLM03], several image-space methods are combined for object and sub-object pruning in collision detection. The approach can handle objects with changing topology. The setup is comparatively complex and self-collisions are not considered.

In [HTG03], an image-space technique is used for collision detection of arbitrarily-shaped, deformable objects. This approach computes a Layered Depth Image (LDI) [SGHS98] of an object to approximately represent its volume. This approach is similar to [SF91], but not restricted to convex objects. Still, a closed surface is required in order to have a defined object volume. It also does not handle self-collisions. In [HTG04], Heidelberger et al. presented an improved algorithm which treats self-collisions combining the image-space object representation with information on face orientation to overcome this limitation. They provided a comparison of three different implementations for LDI generation, two based on graphics hardware and one software-based. Results suggest that the graphics hardware accelerates image-space collision detection in geometrically complex environments, while CPU-based implementations provide more flexibility and better performance in case of small environments (see Fig. 2.14).

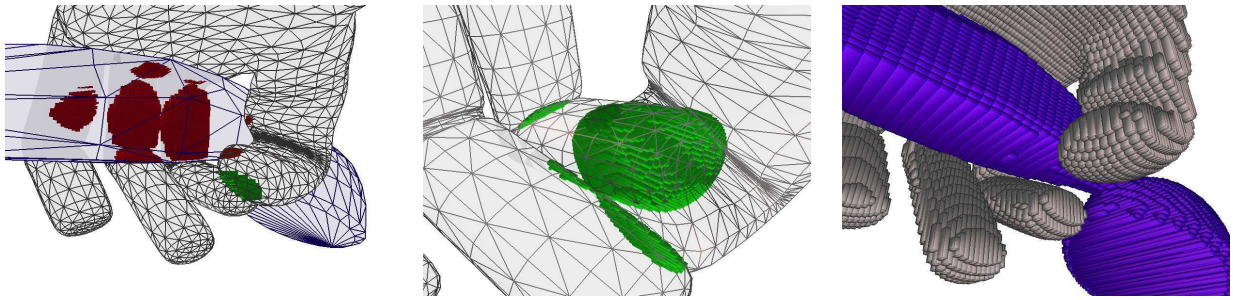


Fig. 2.14: Left : Collisions (red) and self-collisions (green) of the hand are detected. Middle : Self-collisions (green) are detected. Right : LDI representation with a resolution of 64x64. Collisions and self-collisions are detected in 8-11 ms using a standard PC [HTG04].

Image-Space Techniques Summary : In contrast to other collision detection methods, image-space techniques do not require time-consuming pre-processing. This makes them especially appropriate for dynamically deforming objects. Topology changes of objects too can be handled with ease. They can be used to detect collisions and self-collisions. Image-space techniques usually work with triangulated surfaces. However, they could also be used for other object primitives as long as these primitives can be rendered.

Since image-space techniques work with discretized representations of objects, they do not provide exact collision information. The accuracy of the collision detection depends on the discretization error. Thus, accuracy and performance can be balanced in a certain range by changing the resolution of the rendering process. While image-space techniques efficiently detect collisions, they are limited in providing information that can be used for collision response in physically-based simulation environments. Even in the light of next generation

bus technologies such as the PCI Express ® [PS], the read-back speeds are not fast enough for getting all the information for an appropriate collision response (unless the response too is performed in the GPU as in [JP02]). In many approaches, further post-processing of the provided result is required to compute or to approximate information such as the penetration depth of colliding objects.

2.3.7 Continuous Collision Detection

In most cases it is sufficient to check for collisions at the discrete instants of simulations $t_0, t_1, ...$ However in some cases it is possible that some objects cross each other *during* the time interval. This happens when the maximum relative velocity times the time-step is greater than the size, i.e. $\mathbf{v}_{relMax} \times \Delta t \geq$ ObjectSize. In other words, collisions are missed when the objects are too small (say thin) or move very fast. Meseure and Chaillou [MC00] lucidly highlight the need for continuous collision (see Fig. 2.15). Continuous techniques in addition to handling fast moving, thin objects also allows for large time-step simulations. This is especially handy when seen in the context of implicit integration techniques for cloth simulation proposed by Baraff and Witkin [BW98]. We first present the continuous approaches applied to broad phase scenarios in the following section before describing the narrow phase techniques (cf. § 2.3.7.2).

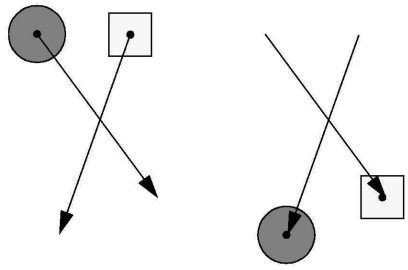


Fig. 2.15: Positions at time t_0 (left) and $t_0 + \Delta t$ (right). Case of collision missed if detected only at discrete instants [MC00].

2.3.7.1 Continuous Collision Detection - Broad Phase

BVHs can also be used to accelerate continuous collision detection, i. e. to detect the exact contact of dynamically simulated objects within two successive time steps. Therefore, BVs do not only cover object primitives at a certain time step. Instead, they enclose the volume described by the linear movement of a primitive within two successive time steps [BFA02, RKC02].

A simple way to augment traditional, static BVH traversals was proposed in [ES99]. During the traversals, for each node a new BV is computed that encloses the static BV of the node at times t_0 and t_1 (and possibly several t_i in-between). Other approaches utilize quaternion calculus to formulate the equations of motion [SSW95, Can86]. Finding the first point

of contact basically corresponds to finding roots of polynomials that describe the distance between the basic geometric entities, i. e. all face/vertex and all edge/edge pairs.

For solids, these polynomials are easier to process if the motion of objects is a screw motion. Thus, [KR03, RKC02] approximate the general motion by a sequence of screw motions. In multi-body systems, the number of collisions at a single time step can increase significantly, causing simple sign checking methods to fail. Therefore, [GK03] developed a reliable method that adjusts the step size of the integration by including the event functions in the system of differential equations, and a robust root detection. In order to quickly eliminate possible collisions of groups of polygons that are part of deformable objects, [MKE03] constructed a so-called velocity cones throughout their BVHs. Another technique sorts the vertices radially and checks the outer ones first [FW99].

We already introduced OBBs proposed by Gottschalk et al. [GLM96] in §2.3.3. Redon et al. [RKC02] adapted the separating axis theorem used for detecting collisions between OBBs to the continuous case. Let the first OBB be described by three axes $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, a center $\mathbf{T}_A$, and its half-sizes along its axes a_1, a_2 and a_3 respectively. Similarly, let the second OBB be described by its axes $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$, its center $\mathbf{T}_B$, and its half-sizes along its axes b_1, b_2 and b_3 respectively. The separating axis theorem states that two static OBBs overlap if and only if all of fifteen separating axis tests fail. A separating test is simple : an axis a separates the OBBs if and only if :

$$|\mathbf{a} \cdot \mathbf{T}_A\mathbf{T}_A| > \sum_{i=1}^{3} a_i|\mathbf{a} \cdot \mathbf{e}_i| + \sum_{i=1}^{3} b_i|\mathbf{a} \cdot \mathbf{f}_i| \tag{2.4}$$

This test is performed for 15 axes at the most (see Fig. 2.16). Here, rather than directly performing the 15 tests in continuous which is computationally very inefficient, Redon et al. used interval arithmetic to narrow down if the OBBs intersected *during* a time step. Since each member of the inequality (2.4) is a function of time, they used interval arithmetic to bound them within a specific interval I. When the lower bound of the left member is larger than the upper bound of the right member, the axis $\mathbf{a}$ separates the boxes during the entire interval I, which means that the boxes will not overlap during the time interval and can be ignored for narrow phase testing.

2.3.7.2 Continuous Collision Detection - Narrow Phase for Rigid Bodies

Having introduced the need for continuous-time collision detection in § 2.3.2, we now describe the methods proposed to do continuous primitive tests. Let us first describe the notational conventions which will use to denote the dynamic state (positions, velocities, etc.) of the primitives. Let a point with position $\mathbf{p}$ (or $\mathbf{p}(t_0)$ implying $\mathbf{p}$ at time t_0) in three-dimensional space moving at a velocity $\mathbf{v}$ intersect with a triangle represented by the points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ each moving with velocities $\mathbf{v}_a, \mathbf{v}_b, \mathbf{v}_c$ as shown in Fig. 2.17. Using similar notations, let an edge with end-points $\mathbf{a}$ and $\mathbf{b}$ moving at velocities $\mathbf{v}_a$ and $\mathbf{v}_b$ respectively collide with

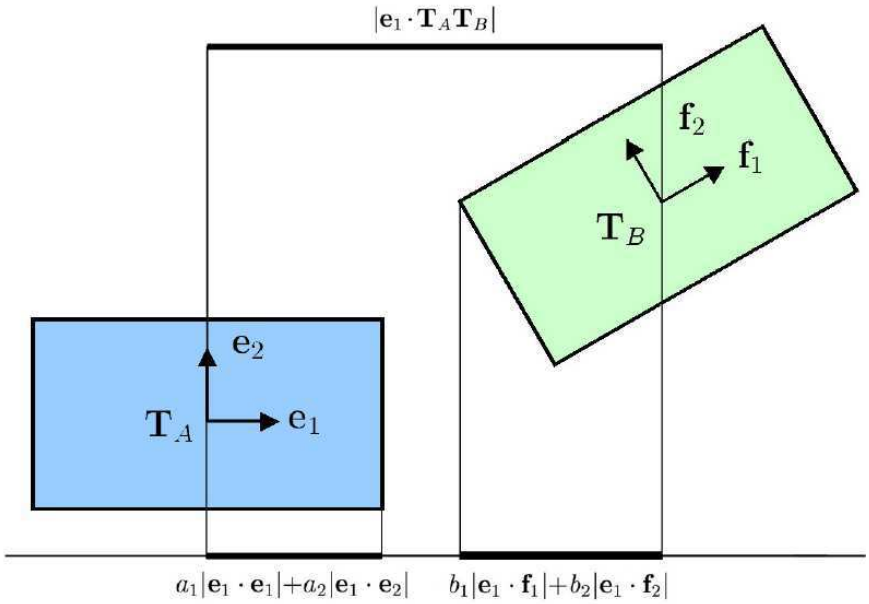


Fig. 2.16: Continuous broad phase collision detection using OBBs. The axis $\mathbf{e}_1$ separates the two oriented bounding boxes since, in the axis direction, the projected distance between the centers of the boxes $|\mathbf{e}_1 \Delta \mathbf{T}_A\mathbf{T}_B|$ is larger than the sum of the projected radii of the boxes, $(a_1|\mathbf{e}_1 \cdot \mathbf{e}_1| + a_2|\mathbf{e}_1 \cdot \mathbf{e}_2|) + (b_1|\mathbf{e}_1 \cdot \mathbf{f}_1| + b_2|\mathbf{e}_1 \cdot \mathbf{f}_2|)$ [Red04].

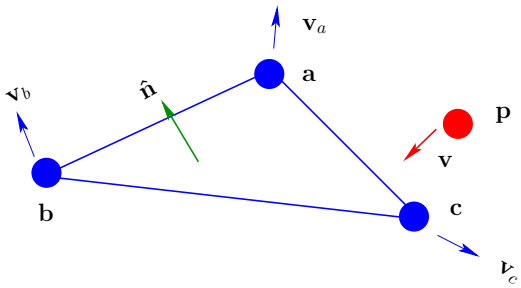


Fig. 2.17: Continuous collision detection if vertex intersecting with a triangle.

another edge with end-points **c** and **d** moving at velocities $\mathbf{v}_c$ and $\mathbf{v}_d$ as shown in Fig. 2.18.

Fifth Order Continuity Tests : In the case of rigid body collisions, Moore and Wilhems [MW88] proposed one of the first work which addressed the collision detection for moving polyhedra for computer animation. Here the intersection of a point with a triangle was described by :

$$\mathbf{p}(t) = \mathbf{a}(t) + u(\mathbf{ab}(t)) + v(\mathbf{ac}(t)) \tag{2.5}$$

where $\mathbf{p}(t) = \mathbf{p}(t_0) + t\mathbf{v}(t_0)$.

This results in a *fifth* order polynomial in t. The actual intersection point is then determined by performing a binary search to first determine the approximate value of t, i.e. the interval $t \in [t_0, t_0 + \Delta t]$ is divided into a number of sub-intervals. The polynomial is evaluated at the two end points. If the sign of the polynomial is different for the two end points of a particular, then solution t should lie within that interval. Using the value of t, the barycentric

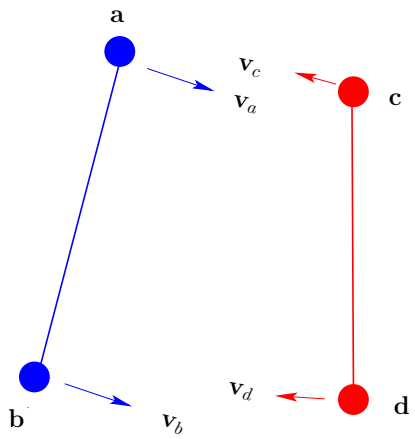


Fig. 2.18: Continuous collision detection edge colliding with an edge.

coordinates are determined and checked if they lie within the valid interval $u \geq 0$, $v \geq 0$ and $0 \leq u + v \leq 1$. As we will quickly see, later methods have reduced the computation from 5^{th} to a 3^{rd} degree polynomial.

For an edge **a** and **b** intersecting with a face at a point **p** with normal **n**, the perpendicular distances are calculated as :

$$d_a = (\mathbf{a} - \mathbf{p}) \cdot \mathbf{n} \tag{2.6}$$

$$d_b = (\mathbf{b} - \mathbf{p}) \cdot \mathbf{n} \tag{2.7}$$

$$t = \frac{|d_i|}{|d_i| + |d_j|} \tag{2.8}$$

and the actual point of intersection is $\mathbf{a} + t(\mathbf{b} - \mathbf{a})$.

Backtracking : Hahn [Hah88] in his work on rigid body simulation proposed *backtracking* in order to rectify the object position after they are found interpenetrating. For point (moving) - polygon (stationary) case, upon collision at time $t_0 + \Delta t$, (the next time step) a ray is originated from the colliding point in a direction given by the relative velocity of the two objects. Assuming that the velocity remains constant during the time step, the ray represents the path that the point took from its position at time t_0. The time step is *backtracked* to generate the new velocities and positions at time $t_0 + \Delta t$ (see Fig. 2.19). Similarly, for the edge-polygon case, the penetration point is calculated by intersecting the polygon swept by the edge during the time step with the edges of the pierced polygon. The collision point is calculated by finding the intersection between the penetrating edge and a ray originating from the penetration point in negative direction of the relative velocity (the bodies having interpenetrated already). The assumption here is that the time step Δt is small enough and/or the velocity of the polygons of the objects are small enough such that the distance covered during Δt is much smaller than the dimensions of the polygons. Obviously, this approach has serious drawbacks when there are multiple collisions. Then, for every collision occurring, we have to backtrack one time step. Also, this approach is not suitable for interactive applications, where we would like to advance in time as the simulation progresses (also cf. § 2.4.1).

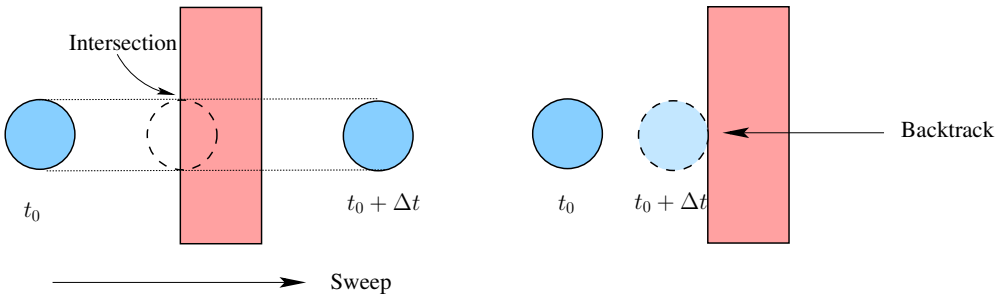


Fig. 2.19: Collision detection by *backtracking* the time step to generate the new positions and velocities.

Linear Interpolation : Schömer et al. [ES99, LSW99] addressed building a collision processing scheme for virtual assembly planning application. The scheme uses several bounding volume representations (spheres, OBBs, k-DOPs, AABBs) to quickly get rid-off non intersecting regions. Otherwise, the classical case follows. We only describe the elementary tests as follows :

Vertex-Face : Given a vertex $\mathbf{p}$ and a fixed face F with equation $\mathbf{n} \cdot \mathbf{x} = n_0$, the distance between them is defined as $d = \mathbf{n} \cdot \mathbf{p} - n_0$. Given the vertex coordinates at time t_0 and $t_0 + \Delta t$, the corresponding distances are d^{t_0} and $d^{t_0+\Delta t}$. The distance is linearly interpolated as $d(t) = d^{t_0} + t/\Delta t \cdot (d^{t_0+\Delta t} - d^{t_0})$. If $d^{t_0} > d^{t_0+\Delta t}$, (meaning the objects are coming towards each other), then the time of collision is computed as $t_c = \Delta t d^{t_0}/(d^{t_0+\Delta t} - d^{t_0})$. If $d^{t_0} \leq d^{t_0+\Delta t}$, the vertex is moving parallel or away from the face. If $t_c < 0$ or $t_c > \Delta t$, there is no collision during the time step. Otherwise, the coordinates of vertex and face at time t_c are computed by interpolation.

Edge-Edge : Here the end-points $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}, \mathbf{d}$, the distances d_i are computed for times t_i, $i = 1, 2$.

$$d_i = \frac{det(\mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}, \mathbf{a} - \mathbf{c})}{|(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})|} \tag{2.9}$$

If $t_c < t_1$ or $t_c > t_2$, then there is no collision, otherwise we need to determine if the point of intersection lies on the edges or not.

Using Interval Arithmetic : Several papers from Redon et al. [RKC00, RKC01, RKC02] address the approach of using a fast continuous collision detection for rigid bodies in a virtual environment. Here, they use a modified version of the OBB hierarchy (adapted to continuous case) combined with interval arithmetic [SWF+93] to achieve interactive collision detection. The relative motion between objects is represented by using a screwing matrix (consisting of translation along one axis and one rotation around the same axis). This motion when expressed in time results in a polynomial equation, whose solution is obtained through interval arithmetic. After doing rejection tests using a modified hierarchical subdivision, the algorithm concentrates on primitives.

Vertex-Face : A collision is detected between a vertex $\mathbf{p}(t)$ and a triangle $\mathbf{abc}(t)$ when :

$$\mathbf{ap}(t) \cdot (\mathbf{ab}(t) \times \mathbf{ac}(t)) = 0 \tag{2.10}$$

Again, a solution $t \in [t_0, t_0 + \Delta t]$ is kept only when $\mathbf{p}(t_c)$ is inside the triangle $\mathbf{abc}(t_c)$.

Edge-Edge : The condition for collision between edges $\mathbf{ab}(t)$ and $\mathbf{cd}(t)$ is :

$$\mathbf{ac}(t) \cdot (\mathbf{ab}(t) \times \mathbf{cd}(t)) = 0 \tag{2.11}$$

A solution $t \in [t_0, t_0 + \Delta t]$ is kept only when the corresponding contact points belong to the edges.

Parametric Curves and Surfaces : Von Herzen et al. [HBZ90] proposed an algorithm to detect collisions between pairs of time-dependent parametric surfaces. The surfaces described in this paper are bounded in terms of the rate of change described by their *Lipschitz* values. The paper discourages the use of determining the time of the collision (say, using binary subdivision given that a collision occurred in the interval (t_1, t_2)). The approach here is to get an upper bound on the velocity of the moving surfaces, so that we can estimate its position at the time of collision. Given two parametric surfaces $f(u_f, v_f, t)$ and $g(u_g, v_g, t)$, would like the earliest time t_{min} such that :

$$||f(u_f, v_f, t_{min}) - g(u_g, v_g, t_{min})|| < \gamma \tag{2.12}$$

The *Lipschitz condition* states that :

$$||f(u_2) - f(u_1)|| \leq L||u_2 - u_1|| \tag{2.13}$$

for some L in some region R of f. The Lipschitz value L is a generalization of the derivative of the surface. Given two parametric surfaces and their Lipschitz values, they were able to determine the earliest collision time. Of course the limitation is that this approach works only with parametric surfaces with bounded derivatives. Later Snyder et al. [Sny92] used interval arithmetic to detect collisions between time-dependent parametric surfaces. This work accounts for both the collisions generated due to new contacts in addition to those generated when bodies already in contact undergo rolling or sliding motion.

2.3.8 Continuous Collision Detection - Narrow Phase for Deformable Bodies

In [Pro97], Provot proposed a mass-spring model for cloth animation with a continuous collision detection scheme used for collision and self-collision. In addition to Moore and Wilhem's basic condition in (2.5), we have another condition that at the time of collision the triangle normal $\mathbf{n}(t) = \mathbf{ab}(t) \times \mathbf{ac}(t)$ should be perpendicular to the vector $\mathbf{ap}(t)$ (see Fig. 2.20). This gives :

$$(\mathbf{ab}(t_c) \times \mathbf{ac}(t_c)) \cdot \mathbf{ap}(t_c) = 0 \tag{2.14}$$

This yields a *third* (in contrast to the fifth degree equations in [MW88]) degree equation in t. A solution t_c should satisfy $t_c \in [t_0, t_0 + \Delta t]$. The value is plugged back in (2.5) to find the valid barycentric coordinates of the triangle.

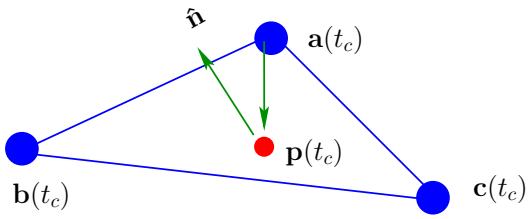


Fig. 2.20: Condition at the time of collision between a vertex and a triangle.

An edge-edge case is considered as follows : During the time interval $[t_0, t_0 + \Delta t]$, there will be a collision for $u, v \in [0, 1]$, when $u\mathbf{ab}(t) = v\mathbf{cd}(t)$. Similar to the vertex-triangle case, Provot introduced another condition that the normal at the time of collision $\mathbf{n}(t) = \mathbf{ab}(t) \times \mathbf{cd}(t)$ should be perpendicular to the vector $\mathbf{ac}(t)$ giving :

$$(\mathbf{ab}(t_c) \times \mathbf{cd}(t_c)) \cdot \mathbf{ac}(t_c) = 0 \tag{2.15}$$

This again yields a third degree equation in t and is solved as before. For handling multiple collisions, whenever collision is detected, they again check to see if handling of this collision has created any further collision scenario. This is continued until no new collision is detected. Naturally, the simulations take a few hours to compute collisions for a few thousand polygon cloth model in a SGI Indigo 2. The self-collision detection is accelerated by pruning low-curvature regions as described in §2.3.3.4.

Bridson et al. [BFA02] proposed a generalized approach for handling collisions, contacts and friction for cloth animation. We will revisit the overall collision framework in detail in § 2.4.3, but let us first detail the narrow-phase detection technique. Here, in addition to doing dynamic collisions on the lines of Provot [Pro97], they also perform static proximity tests. The idea behind this is to treat different types of collisions in a different manner as we will see in §2.4.3. For a vertex $\mathbf{p}$ and triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ with normal $\hat{\mathbf{n}}$ tests, they first check if $|\mathbf{pc} \cdot \hat{\mathbf{n}}| < h$, where $\mathbf{pc} = \mathbf{c} - \mathbf{p}$ and h is the thickness. If yes, they find the intersecting barycentric coordinates w_1, w_2, w_3 where $w_3 = 1 - w_1 - w_2$, by solving :

$$\begin{bmatrix} \mathbf{ac} \cdot \mathbf{ac} & \mathbf{ac} \cdot \mathbf{bc} \\ \mathbf{ac} \cdot \mathbf{bc} & \mathbf{bc} \cdot \mathbf{bc} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \mathbf{ac} \cdot \mathbf{pc} \\ \mathbf{bc} \cdot \mathbf{pc} \end{bmatrix} \tag{2.16}$$

The actual intersection point is then determined as $w_1\mathbf{a} + w_2\mathbf{b} + w_3\mathbf{c}$.

Similarly the edges $\mathbf{ab}$ and $\mathbf{cd}$ are checked for parallel case by evaluating $|\mathbf{ab} \times \mathbf{cd}|$. For non-parallel cases the barycentric values u, v are found as :

$$\begin{bmatrix} \mathbf{ab} \cdot \mathbf{ab} & -\mathbf{ab} \cdot \mathbf{cd} \\ -\mathbf{ab} \cdot \mathbf{cd} & \mathbf{cd} \cdot \mathbf{cd} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \mathbf{ab} \cdot \mathbf{ac} \\ -\mathbf{cd} \cdot \mathbf{ac} \end{bmatrix} \tag{2.17}$$

The intersecting point is then computed $\mathbf{a} + u\mathbf{ab}$ and $\mathbf{c} + v\mathbf{cd}$. Note that none of the the methods presented in this section handles the detection of degenerate edges (when they become co-planar or collinear at the time of collision). In chapter 4, we present new techniques to handle such cases in addition to comprehensively summarizing previous work with experimental validation.

2.4 Collision Response

So far we have seen only one part of the problem, that of detecting collisions. It makes no difference to the simulation behavior if we only detect and do not act upon to prevent the interpenetration from happening. There are several possible approaches here. We discuss the relative merits of various approaches for both rigid and deformable objects.

2.4.1 Rigid Body Collision Response - Local Correction

We first describe the impulse based method to deal with rigid body collisions. As we previously mentioned in §2.3.7, Hahn proposed [Hah88] *backtracking* in order to rectify the object position after they are found interpenetrating. For two colliding rigid objects with masses m_1 and m_2, with linear velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ and angular velocities ω_1 and ω_2 (see Fig. 2.21), let it collide with an impulse $\mathbf{P}_1$ and $\mathbf{P}_2$. Hahn applied the conservation of linear

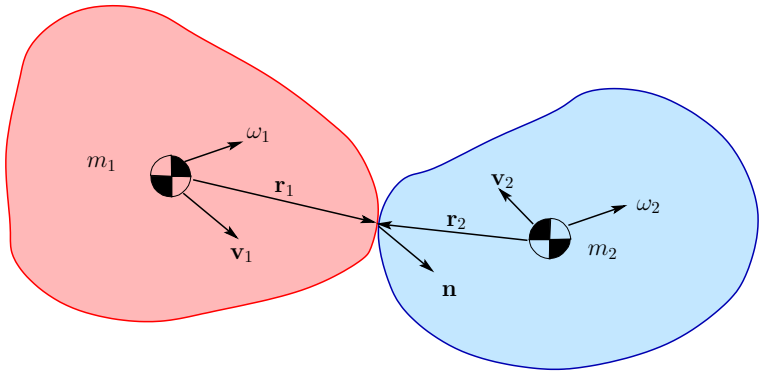


Fig. 2.21: Impulse collision response between two rigid bodies [Hah88].

and angular momentum as :

$$\begin{aligned} m_1(\mathbf{v}_1^{'} - \mathbf{v}_1) &= -\mathbf{P}_1 \\ m_2(\mathbf{v}_2^{'} - \mathbf{v}_2) &= \mathbf{P}_2 \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} \mathbf{I}_1 \cdot (\omega_1^{'} - \omega_1) &= \mathbf{r}_1 \times (-\mathbf{P}_1) \\ \mathbf{I}_2 \cdot (\omega_2^{'} - \omega_2) &= \mathbf{r}_2 \times (\mathbf{P}_2) \end{aligned} \tag{2.19}$$

Hahn further used the generalized Newton's rule to compute the new velocities $\mathbf{v}_1^{'}$, $\mathbf{v}_2^{'}$, $\omega_1^{'}$ and $\omega_2^{'}$ as :

$$(\mathbf{v}_1^{'} + \omega_1^{'} \times \mathbf{r}_1) \cdot \mathbf{n} - (\mathbf{v}_2^{'} + \omega_2^{'} \times \mathbf{r}_2) \cdot \mathbf{n} = -\epsilon((\mathbf{v}_1 + \omega_1 \times \mathbf{r}_1) \cdot \mathbf{n} - (\mathbf{v}_2 + \omega_2 \times \mathbf{r}_2) \cdot \mathbf{n}) \quad (2.20)$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ are the vectors from the center of gravity of each body to the colliding point, $\mathbf{n}$ is the normal at that point and ϵ is the coefficient of restitution. ϵ depends of the elasticity of the material and has a value of 0 for a perfectly inelastic collision to a value of 1 for a perfectly elastic collision. Assuming a frictionless interaction, he expressed (2.20) in the orthogonal directions to $\mathbf{n}$:

$$\begin{aligned} [(\mathbf{v}_1^{'} + \omega_1^{'} \times \mathbf{r}_1) - (\mathbf{v}_2^{'} + \omega_2^{'} \times \mathbf{r}_2)]_t = 0 \\ [(\mathbf{v}_1^{'} + \omega_1^{'} \times \mathbf{r}_1) - (\mathbf{v}_2^{'} + \omega_2^{'} \times \mathbf{r}_2)]_r = 0 \end{aligned} \quad (2.21)$$

where t and r are the orthogonal components of the velocity vector perpendicular to $\mathbf{n}$. (2.18), (2.19) and (2.21) gives us 15 independent equations in 15 unknowns ($\mathbf{P}$, $\mathbf{v}_1$, $\mathbf{v}_2$, ω_1 and ω_2). The problem with this approach is in case of multiple collision, this will result in the algorithm getting stuck in time advancing very slowly or never advancing at all. As shown in Fig. 2.19, Hahn used backtracking to apply the above response one by one.

2.4.2 Global Collision Resolution in Rigid Bodies

If there are m collisions in a system, one can handle them one by a local correction method discussed above. But what happens if the correction of one colliding pair *creates* a new collision. Obviously this can be confirmed only by performing another collision detection pass. Alternatively, we can handle these collisions by treating them *simultaneously*. In most cases this consists of formulating it as an *optimization problem*.

Baraff [Bar89] proposed an analytical method for calculating non-penetration forces between polygonal rigid objects by obtaining an NP-hard quadratic programming solved using a heuristic approach. Baraff [Bar90] further extended this to curved objects. In [Bar94] he proposed a better approach for the rigid body contact problem by computing a global solution to compute the non-penetration forces. In this approach, Baraff considered the problem of non-penetrating constraints as a Linear Complementarity Problem (LCP) by defining either the force $f_i \geq 0$ when the acceleration $a_i = 0$ and viceversa. Here the force for the first contact f_1 was computed by setting the rest of the forces $\{f_2, f_3, ..., f_m\} = 0$. Similarly f_1 and f_2 was computed by setting $\{f_3, f_4, ..., f_m\} = 0$, and so on. Hence he solved a quadratic programming problem (QP) each time thus making this approach as some sort of sequential quadratic programming [GMS02].

Mirtich [Mir00] tried to alleviate the problem of multiple collisions by proposing a time-warp algorithm which "holds" back colliding objects while moving forward non-colliding states - this approach works better but is still problematic in the case of large number of

collisions. Another approach is the treatment of rigid body stacking by plausible simulation methods using optimization methods in Milenkovic and Schmidl [MS01]. Here the collision processing is handled as follows : from time t_0 to $t_0 + \Delta t$, they computed the final position to see if there is a collision. If yes, they re-computed the position such that there is no interpenetration. This work follows the principle of *physically-plausible motion* which means that it may "appear" to be a realistic motion but is not true. For each pair of bodies, a set of *critical vertices* are identified and the above constraint minimization is applied for those vertices. However, if a non-critical vertex happens to collide, this vertex is added to the critical vertex list and the simulation goes on. Also, if the constraint cannot be solved, it is declared as "infeasible", and the positions and orientations are rolled back to the previous positions. In this method, it is possible to miss out some collisions, when a remote vertex (non-critical) moves quickly and collides. The algorithm will not be able to prevent such a scenario, though it can correct itself in subsequent time steps. Again, we reiterate that the prediction method will adjust the positions in a non-physical way.

We usually differentiate between a *collision* and a *contact* by checking the relative velocity of the colliding objects - if its magnitude is within a particular threshold - it is considered as a resting contact else it is a collision [BW01]. The problem occurs when we consider a resting contact as a series of elastic collisions : this may result in vibrations if the threshold parameters are not set right. Gundelman et al. [GBF03] proposed a better approach with a time integration scheme which eliminated the need for determining *ad hoc* threshold velocities as in [Mir96]. This coupled with a shock propagation scheme provided visually pleasing results for collision cases with thousands of polygonal objects.

2.4.3 Deformable Body Collision Response

Penalty vs Impulse Corrections : Collision Response in the case of deformable objects such as cloth, we can either correct by displacement [GC94] and velocity corrections [Pro97] or by penalty forces by inserting a stiff spring [BW98].

If we try to extend Baraff's non-penetrating force approach [Bar94] described in §2.4.2 to deformable bodies, we would need an explicit computation of the $\mathbf{M}^{-1}$ - in this case the inverse of the stiffness matrix $\mathbf{K}^{-1}$. This is a very difficult thing to compute sometimes even impossible due to ill-conditioned or near-singular matrices. In contrast, in deformable body simulations, we do not necessarily have an explicit representation of the $\mathbf{K}$ matrix. For solving an equation such as $\mathbf{Kx} = \mathbf{b}$, we only need to create methods which computes a the matrix-vector product $\mathbf{K}$ to some vector $\mathbf{y}$ using an iterative method such as the conjugate gradient. Most of the methods we describe below has only an implicit matrix representation which is memory efficient and more inexpensive to compute than inverting $\mathbf{K}$ at every time step (cf. §5.2.2 for a detailed discussion on this).

Once the collisions are detected as described in § 2.3.8, Provot [Pro97] used only impulse-based collision response by instantaneous correcting the displacements and velocities correc-

tions. This method while being precise nevertheless adds undesirable extra energy into the system and may also provoke new collisions thus warranting additional treatment. In such cases, penalty methods might offer a good solution to efficiently introduce a response. The structure of the stiffness matrix is not altered assuming that we include the collision spring forces in the matrix - though we see a degradation in the conditioning of the matrix. This becomes especially obvious when the mechanical equations are formulated as an explicit form. But implicit formulation first proposed in graphics by Baraff and Witkin [BW98] handles such "stiff" equations with ease.

Volino and Magnenat-Thalmann [VMT00] proposed a global solution to the problem of multiple collisions which occurs in the case of self-collisions in deformable objects (such as a folding ribbon). They computed forces needed to correct the accelerations, velocities and positions of the present, next and next-to-next time steps. By directly manipulating the positions, this method may introduce large amounts of strain and energy into the system.

Bridson et al. [BFA02] realized this issue and hence handled collision response via a combination of velocity and spring-based repulsive impulses. Their approach intelligently combined both the impulse and penalty methods thus taking care of the different scenarios while overall not adding a significant amount strain into the system. This explained the rationale of detecting collisions *both* at discrete and continuous time instants (cf. §2.3.8). However, as will show their algorithm while being complex might fail to converge in one of the steps. They do not guarantee that all the collisions will be treated.

Let us describe this method in detail. In order to dramatically reduce the number of collisions, they used *repulsive forces* for those collisions lying in the "proximity region" (detected using (2.16) for vertex-triangles and (2.17) for edge-edge). Among the pairs which are in close proximity, they differentiate between pairs which are to be handled (moving towards each other) and those that were already handled (probably moving apart due to repulsion or impulse). An inelastic collision (zero restitution coefficient) was applied if the relative velocity in the normal direction, $v_N < 0$ (means that they are approaching each other). Hence an inelastic impulse of magnitude $I_c = \frac{mv_N}{2}$ was applied in the normal direction.

To account of the thickness h of the cloth, the cloth pieces should be well separated. Here they used a repulsive spring force to separate the colliding cloth pieces. Problems with stiffness were avoided by limiting the maximum repulsion that can be applied. This allows the cloth segments to stay close enough together to feel repulsion forces in subsequent time steps.

$$I_r = -\min\left(\Delta t k d, m\left(\frac{0.1d}{\Delta t} - v_N\right)\right) \tag{2.22}$$

Here $d = h - (\mathbf{p} - w_1\mathbf{a} - w_2\mathbf{b} - w_3\mathbf{c}) \cdot \hat{\mathbf{n}}$ is the overlap (using the notations of (2.16)). The repulsion force is not applied if the normal component of the relative velocity satisfies the following condition $v_N \geq \frac{0.1d}{\Delta t}$. This ensures that the repulsive forces does not exceed the overlap region in one time step.

The impulses calculated above were then applied in the opposite direction to each of the

colliding primitives proportionate to their barycentric coordinates. For a vertex **p** colliding with the triangle **abc** (using notations in (2.16)), they normalized the impulse using the barycentric coordinates w_1, w_2, w_3 as follows :

$$\tilde{I} = \frac{I}{1 + w_1^2 + w_2^2 + w_3^2} \tag{2.23}$$

where I is I_c for velocity impulse or I_r for spring-based impulse. The adjusted value $\tilde{I}$ is then redistributed among a vertex-triangle pair as follows. For a colliding vertex, the pre-impulse velocity $\mathbf{v}$ is altered to :

$$\mathbf{v}^{\text{new}} = \mathbf{v} - \frac{\tilde{I}}{m}\hat{\mathbf{n}} \tag{2.24}$$

For the triangle the pre-impulse velocities $\mathbf{v}_a$, $\mathbf{v}_b$ and $\mathbf{v}_c$ are altered as :

$$\begin{aligned} \mathbf{v}_a^{\text{new}} &= \mathbf{v}_a + w_1\frac{\tilde{I}}{m}\hat{\mathbf{n}} \\ \mathbf{v}_b^{\text{new}} &= \mathbf{v}_b + w_2\frac{\tilde{I}}{m}\hat{\mathbf{n}} \\ \mathbf{v}_c^{\text{new}} &= \mathbf{v}_c + w_3\frac{\tilde{I}}{m}\hat{\mathbf{n}} \end{aligned} \tag{2.25}$$

where we assume that all the particles have the same mass m.

For an edge-edge pair, the impulses are redistributed as follows. Referring to the notation in Fig. 2.18, the calculated impulse (I_c or I_r) is first normalized with the colliding barycentric coordinates u, v as :

$$\tilde{I} = \frac{2I}{u^2 + (1-u)^2 + v^2 + (1-v)^2} \tag{2.26}$$

The recalculated impulse is then redistributed among the four end-points to find the new velocities as :

$$\begin{aligned} \mathbf{v}_a^{\text{new}} &= \mathbf{v}_a + (1-u)\frac{\tilde{I}}{m}\hat{\mathbf{n}} \\ \mathbf{v}_b^{\text{new}} &= \mathbf{v}_b + u\frac{\tilde{I}}{m}\hat{\mathbf{n}} \\ \mathbf{v}_c^{\text{new}} &= \mathbf{v}_c - (1-v)\frac{\tilde{I}}{m}\hat{\mathbf{n}} \\ \mathbf{v}_d^{\text{new}} &= \mathbf{v}_d - v\frac{\tilde{I}}{m}\hat{\mathbf{n}} \end{aligned} \tag{2.27}$$

where we assume that all the particles have the same mass m.

The collision impulses were applied sequentially or all at once. While taking implicit measures (such as (2.22)) to limit additional strain due to collision response, they also have explicit control algorithm to limit both the strain and the strain rate. A parallel Jacobi step is used for the former whereas a sequential Gauss-Siedel step is performed for the latter. The authors also consider applying the impulses parallel in some cases of multiple collisions when the sequential application fails to converge. While being the state-of-the-art giving beautiful

visuals (see Fig. 2.22), this approach is also complicated since it requires several complex steps (Jacobi and Gauss-Seidel updates for strain/strain rate control) and iterations. Note that in all the above methods, the solution is iterated over a few times before which collisions are expected to be resolved. But they nevertheless do not guarantee that all the collisions will be resolved.

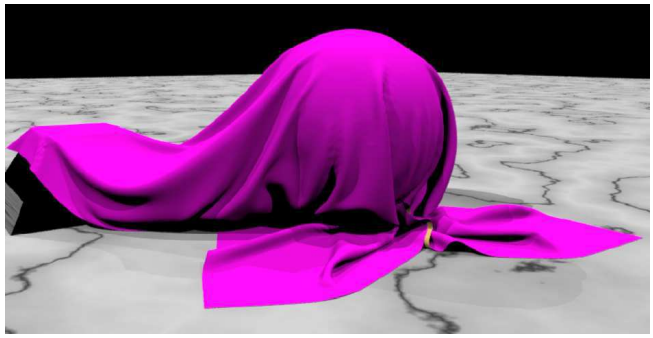


Fig. 2.22: Treatment of collisions and contact in cloth animation [BFA02].

Huh et al. [SHB01] proposed to solve the problem of multiple collisions in cloth by dividing the collisions into several *impact zones* (IZ) instead of addressing one by one. The collisions *within* each IZ is treated simultaneously by a linear equation to find the new velocity impulses which also conserves momentum. Then the collisions *between* the IZ are then treated. This way they strive not to provoke new collisions while treating the existing ones. However, there is no guarantee that collision resolutions within an IZ don't create new collisions - in such cases the algorithm needs to be looped over. The examples they use too are relatively unconstrained in terms of collisions with simulations at very low time step values (5 ms at the most).

In contrast to methods which attempt to "guarantee" that no collisions are missed, Baraff et al. [BWK03] propose an approach which untangles cloth if external constraints actually result in an entanglement - i.e., there are in fact intersecting simulation states. This way the simulator is not obliged to assume that past states are "collision free" (and continue to apply an erroneous collision response). This often happens in a CG film production environment when motion capture data driving the character animation may force such situations (see Fig. 2.23). To this effect, a *global intersection analysis* (GIA) proposed by the authors which resolves these tangles occurring due to pinching bringing back to the normal state in addition to resolving any artifacts. The GIA finds the set of intersecting contours and notes this information. It is subsequently used to decide whether to apply attractive forces (for tangled pairs) or apply the regular repulsive forces. We reiterate that this approach was conceived with a production environment in mind. Real-time applications may not have such a luxury of time.

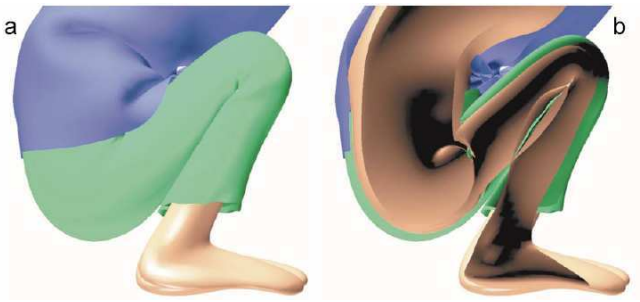


Fig. 2.23: Pinching in production environments handled using global intersection analysis [BWK03].

2.5 Chapter Summary

In this chapter, we described the various techniques for physically-based animation, collision detection and response. This chapter has shown the difficulty of processing collision detection and response for deformable objects in particular when they are thin. None of the approach we described provides the robust solution we are looking for in this case. In the forthcoming chapters, we describe the problem in detail and propose new ideas to tackle them.

Since the emphasis of this thesis work is on robust handling of collisions, we would like to use a suitable existing approach for mechanical modeling. Given the available choices described earlier, a mass-spring system with a bending model and a robust integration such as implicit can be used to a simulate a variety of objects (linear, cloth, volumetric). For collisions, we described a variety of broad phase techniques which will quickly identify the zones of collisions. In addition, we saw the importance of continuous collision detection. In the coming chapters, we describe how we adapted them for the problem at hand. After the detection, it is important that we treat them correctly. The collisions can either be attended one by one or a global approach is possible. Each has its own merits and demerits. Proper attention has to paid so that we don't create unnecessary strain or provoke more collisions. If it is the latter, the collisions have to rechecked. In our proposed approaches, we take up these problems and present a set of new solutions.

CHAPTER 3

Problem at Hand

In this chapter, we illustrate the problem with the existing approaches to collision handling with the help of two case studies : first, that of a highly deformable, self-colliding object such as the human intestine. Second, that of a stiff cable in contact with a rigid object. In each case, we highlight the shortcomings of the current approaches with indications on how to solve them.

3.1 Thin Objects

The simulation of thin deformable objects pose a particularly difficult problem. Most of the approaches described in §2.3.3 were developed for volumetric models and are not applicable here. From the point of view of collision detection, traditional approaches such as bounding volume hierarchies (cf. §2.2) are very ineffective. This is evident since these objects tend to collide with themselves at multiple points (see Fig. 3.1). Hence the update required at each time step tends to be expensive thus negating the idea of broad phase collision detection. Thus there is a need to come up with a new collision detection approach. In addition, once the collisions are detected, responding to them also is problematic. The collision problem becomes even more complex for those objects which are extremely rigid (such as a mechanical cable). Here, traditional collision responses such as displacement correction will cause instabilities due to the high stiffness. Hence, this too requires a more sophisticated treatment. In this chapter, we describe an initial approach for collision response which was simple and effective.

Nevertheless there are problems with this approach. We shall highlight them and propose means of solving them in the next chapters.

Broadly, this chapter is organized as follows. We illustrate the need for new approaches through two case studies.

- Modeling, animating and collision handling in a surgery simulator : case of an intestine and a thin mesentery tissue
- Mechanics and collision handling in a rigid cable

Note that the problems concerning the thin objects presented here will occur in both lineic (such as intestine or cable) and surfacic objects (such as mesentery or cloth). Though our case studies comprise of only lineic objects, one can very well extrapolate it to thin surfaces as well.

The remainder of this chapter is organized as follows. For the intestine, we describe the mechanical and collision model we developed and also justify the need for this specific new approach. We present the results of our method and highlight its shortcomings for more complicated cases. Then for the second case of a rigid cable, we show how a specialized mathematical solver and collision routines can be used to accelerate the computation time for simulation. Finally, while dealing with collision response, we will show that traditional approaches such as penalty or impulses for the cable with very high stiffness are not very helpful.

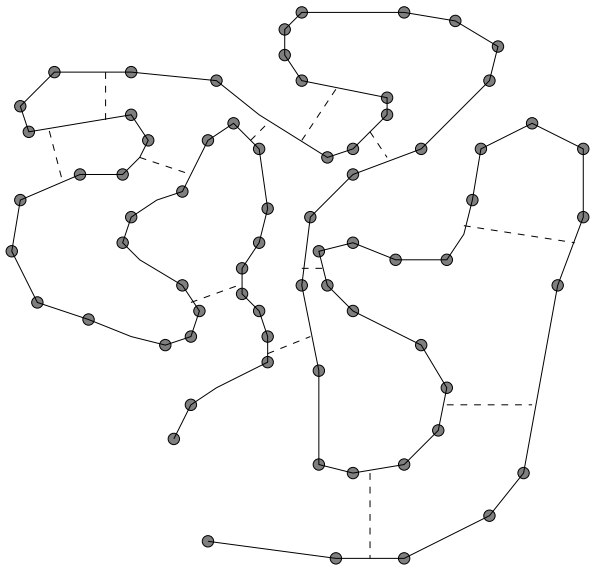


Fig. 3.1: Multiple self-collisions in deformable thin objects.

3.2 Case Study #1 : Intestine

This research work described here was performed in collaboration with our research parters LIFL [1] and IRCAD[2] in the context of an INRIA ARC SCI [3] research project during the beginning part of my thesis published in [RCFC03, RGF+04]. It is aimed towards the development of a VR-based trainer for colon cancer removal. Such a trainer will enable the surgeons to interactively view and manipulate the concerned virtual organs as during a real surgery. First, we present a method for animating the small intestine and the mesentery (the tissue that connects it to the main vessels) in real-time, thus enabling user-interaction through virtual surgical tools during the simulation. We then present a stochastic approach that exploits temporal coherence for fast collision detection in highly deformable, self-colliding objects. A simple and efficient response to collisions is also introduced in order to reduce the overall animation complexity.

3.2.1 Introduction

Minimally invasive surgical (MIS) procedures are gaining popularity over open procedures among surgeons and patients. This is mainly due to lesser post-operative pain, fewer infections and an overall faster recovery. The tremendous success of laparoscopic cholecystectomy (gall bladder removal) has prompted surgical practitioners and educators to apply such techniques to other gastrointestinal procedures. In this research project, we focus on *laparoscopic colectomy* (colon cancer removal). Studies show that many patients undergoing this procedure benefit from the advantages of MIS procedures listed above while sharing the same risks of the corresponding open procedure [Fin00]. Yet, as with most laparoscopic procedures, it is difficult to master with a very flat learning curve [SB95].

As part of the current training procedure, surgeons practice on pigs to get a feel of the organ's behavior. However, this technique is prohibitively expensive and also raises numerous ethical issues. We believe that a VR-based simulator platform can significantly help non-specialist surgeons and medical residents to acquire the necessary surgical skills in a cost-effective way. This may well result in popularizing the use of the laparoscopic technique for this procedure thus benefiting more patients. Thus, our aim is to simulate the behavior of the intestine in real time when the surgeon is practicing in the virtual environment. Note that the current scope of this research work does not include the simulation of the cancer removal itself. The simulator focuses on two important pedagogical problems : (1) *Camera positioning* by allowing the trainee to visualize the relevant organs in 3D, (2) *Manual dexterity* by letting them interactively manipulate these organs. For many surgeons who are trained primarily in open techniques, this may help to overcome the perceptual and motor challenges associated

[1] GRAPHIX/Alcove, Laboratoire d'Informatique Fondamentale de Lille, 59655 Villeneuve d'Ascq Cédex, FRANCE, `http ://www2.lifl.fr/GRAPHIX/`

[2] Institut de Recherche contre les Cancers de l'Appareil Digestif, 67091 Strasbourg Cédex, FRANCE, `http ://www.ircad.org/`

[3] Action de Recherche Coopérative - Simulateur de Chirurgie Intestinale

with MIS procedures.

We will first review the background of the problem and highlight the challenges they pose. During this surgical procedure, the patient is lying on his back. As a result, the small intestine (henceforth simply referred to as *intestine*) is positioned just above the colon region, hiding the colon beneath (see Fig. 3.2). The intestinal region of a human body is characterized by a very complex anatomy. The intestine is a tubular structure, about 4 m long, constrained within a small space of the abdominal cavity, resulting in the creation of numerous *folds*. This is further complicated by a tissue known as the *mesentery* that connects the intestine to the blood vessels [ABR02] (see Fig. 3.3). An important surgical task of this procedure is

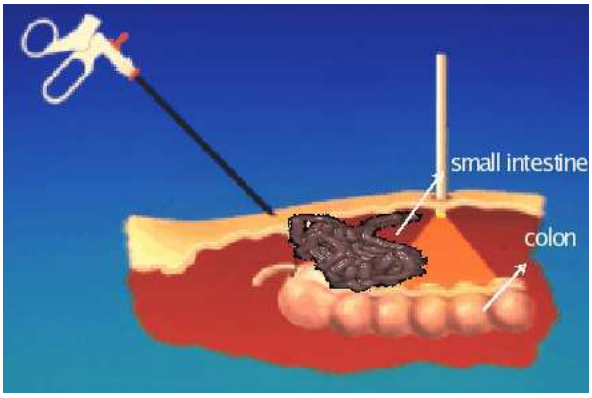


Fig. 3.2: Position of the intestine during the surgery.

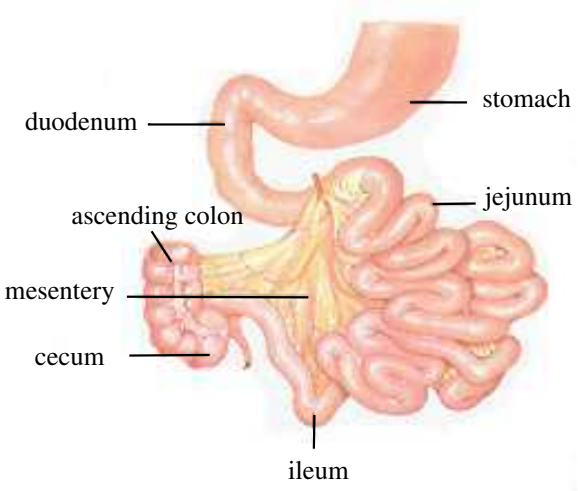


Fig. 3.3: Anatomy of the intestinal region.

to displace the tissues and organs by pulling and folding them from the site of the operation [fVS98]. As the surgeon manipulates these organs, they deform and collide with each other. Thus the broad challenges here are the real-time animation and visualization of these organs at an acceptable frame rate (a minimum of 25 frames a second). Our overall approach to solving this problem consists of a layered model : a skeletal axis deformed using physically-based animation, rendered with a generalized cylinder-based skinning. Thus in order to animate

these organs in real-time, we propose to :

- efficiently model the intestine and the mesentery taking into account its geometry.
- detect the collisions and self-collisions occurring in the intestinal region during animation.
- provide a fast and stable collision response.

The skeletal model used for animation should be covered with a triangulated mesh which can be realistically shaded or textured. However, a naïve skinning approach will create tessellation problems in high-curvature regions. We use a hardware-accelerated skinning based on generalized cylinders proposed by Grisoni et al. [GM03]. All these contributions have been implemented within a surgical simulator platform [MDH+03] that can be easily evaluated by the surgeons.

3.2.2 Real-time Animation of Deformable Models

In the previous chapter, we covered several models developed for deformable object simulation (cf. § 2.2) and the treatment of collisions between deformable objects (cf. § 2.3 and § 2.4). The problem we have to solve here is different. As will be shown in the next section, no volumetric deformable model will be needed since the intestine and the mesentery can be represented as a 1D and a 2D structure respectively. Moreover, the main issue is to detect and process the collisions and self-collisions of the intestinal system in real-time. Accordingly, a simple spline model animated by mass-spring dynamics was used by France et al. [FAM+02] for simulating the intestine.

For collision detection, we covered several "traditional" approaches such as the BVH in §2.3. However, they are not suitable for intestine-mesentery interaction where, even a small local deformation can potentially cause a large movement of the folds. This creates a large scale global deformation, which prevents the BVH from being efficiently updated. France et al. [FLMC02] used a grid-based approach (cf. §2.3.4) for pre-detecting the self-collisions of the intestine and the collisions with its environment. All objects were first approximated by bounding spheres, whose positions were stored, at each time step, in the 3D grid. Each time a sphere was inserted into a non-empty voxel, new collisions were checked within this voxel. Though this method achieved real-time performances when the intestine alone was used, it did not handle the simulation of the mesentery.

An alternate multi-resolution method, based on layered shells, was recently presented by Guy and Debunne [DG04]. It is well-suited for collision detection between deformable objects since the shells themselves are deformable structures extracted from a multi-resolution representation of these objects. Though suitable for volumetric deformable bodies, this method is not appropriate for intestine and mesentery modeling, since the time-varying folds cannot easily be approximated at a coarse scale. They also exploited temporal coherence following Lin and Canny's [LC92] idea of detecting collisions between convex polyhedra by tracking pairs of closest vertices. These pairs were efficiently updated at each time-step by propagating closest distance tests from a vertex to its neighbors. Guy and Debunne adapted this

technique for detecting collisions between his volumetric layered shells very efficiently. Since these shells were neither convex nor rigid, a stochastic approach was used at each time step to generate new pairs of points anywhere on the two approaching objects. These pairs were made to converge to the local minima of the distance, disappearing when they reached an already detected minimum. Our work is inspired by this idea of *stochastic* collision detection exploiting temporal coherence. It has been adapted here to the specific processing of multiple collisions and contacts between the intestine and the mesentery folds.

3.2.3 Modeling and Animating the System

Our overall aim is not to develop an accurate, patient-specific trainer but rather a generic simulator which can help the surgeons to practice the gestures used to manipulate the organs. The first problem we have to solve is to create a virtual model of the intestine and the mesentery which will serve as the basis for the animation and rendering stages. In this section, we describe the actual anatomy in brief, provide the basis of our approach and present the details of our model. We further present the mechanical model needed for animating the intestinal system.

3.2.3.1 Anatomical Model

In order to extract the anatomy of the intestine and the mesentery, we sought the help of our medical collaborator IRCAD [4] in Strasbourg (a digestive cancer research institute). With the current imagery techniques, they found it impossible to extract the complex anatomy of organs such as the mesentery. Hence we decided to come up with something simpler than the actual geometry, but which can still capture the overall behavior.

The mesentery is a folded membrane-like surface, approximately 15 cm wide, which links the intestine to the main vessels of 10 cm length (see Fig. 3.3). Since the mesentery is a non-developable surface (which cannot be unfolded onto a plane), setting up its initial geometry free of self-intersections is quite difficult. We solved the problem by modeling a possible rest position for the intestine as a folded sine curve lying on the inner surface of a cylinder of radius 15 cm. With the axis of the cylinder representing the main vessel, the folds are placed on the cylinder such that their total length is 4 m (see Fig. 3.4). Then the mesentery can be defined as the surface generated by the set of non-intersecting line segments linking the cylinder axis to the curve. Though this initial geometry is too symmetric to be realistic, this model gives adequate local geometric properties to the mesentery membrane (see Fig. 3.5). When animated under the effect of gravity, this collision-free initial positions will automatically move to their correct positions. The geometry of the intestine is defined by creating a piecewise tubular surface of radius 2 cm along its skeleton curve. The thickness of the mesentery surface, parameterized based on patient-specific data, was set to 1 cm.

[4] Institut de Recherche contre les Cancers de l'Appareil Digestif, 67091 Strasbourg Cédex, FRANCE, `http ://www.ircad.org/`

3.2.3.2 Mechanical Considerations for Animation

For animation, our motivation to use a mass-spring approach was derived from the following arguments :

- The mass-spring method is more efficient and suitable over FEM for deforming bodies with large displacements and local elastic deformations.
- In addition, an organ such as the intestine can have an infinite number of rest states, whereas FEM is based on the notion of displacements with respect to a unique rest state thus making it unsuitable for our case.
- Mass-spring can provide stable simulation of deformable objects at moderate time-steps.
- We can adjust the behavior of the system intuitively by adjusting the damping, stiffness, etc.
- Collision detection here is a much more complex task requiring more CPU-time over animation. So, a very complex mechanical model might slow down the simulation.
- Finally, the *perceived quality* of most interactive 3D applications does not depend so much on exact simulation but rather on real-time response to collisions [US97].

Accordingly, we designed a model consisting of mass points connected by damped springs. Since the mesentery has a much larger length (4 m near the intestine) than width (15 cm near the vessel), we sampled our model by four sets of 100 masses each (see Fig. 3.5). The last set of masses requires no computation since they are attached to the main vessels, requiring only 300 masses to be integrated at each time step. In addition, no specific model is needed for the intestine since it can be simulated by adjusting the mass and stiffness values along the first bordering curve of the mesentery surface.

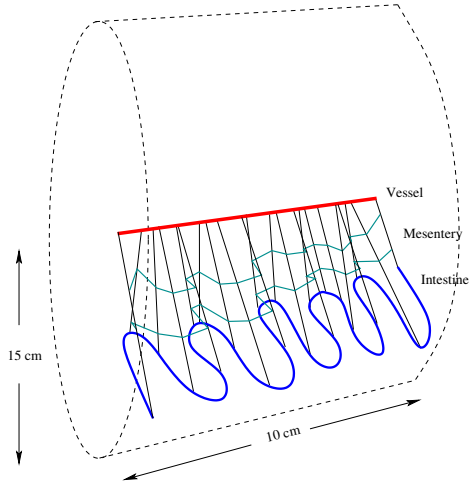


Fig. 3.4: Initialization of the geometry of the intestine and mesentery. The intestine is drawn on the inner surface of a small cylinder. Figure greatly simplified for clarity.

3.2.4 Real-Time Collision Processing

A major computational bottleneck in many animation/simulation systems is the handling of collisions between the objects under the influence of external forces (gravity, user-input,

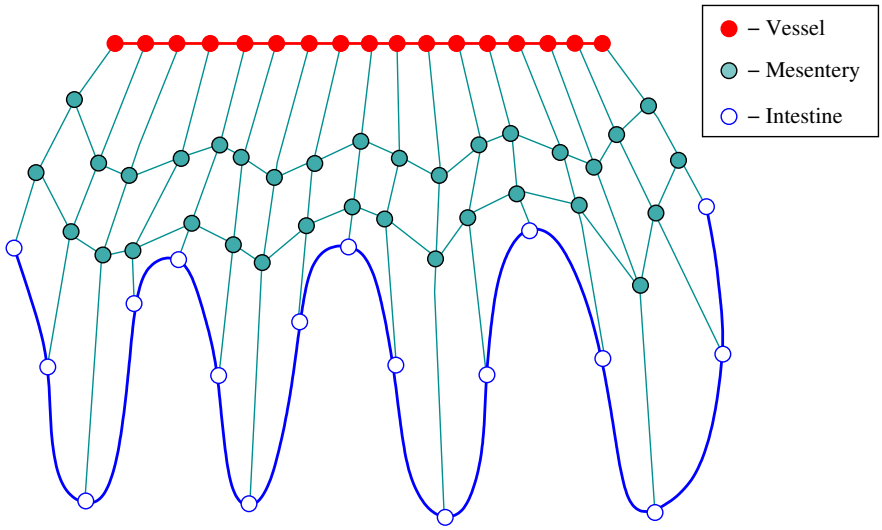


Fig. 3.5: Mechanical model of the organs (shown unfolded).

etc.). In our case of real-time simulation, we need to perform this as quickly as possible. Failure to detect the collisions will result in interpenetration between the intestine's folds or the mesentery's folds - an unrealistic behavior. In this section, we describe our approach, which efficiently detects the colliding regions and a response algorithm which prevents/corrects the interpenetrations.

3.2.4.1 Collision Detection

Our method for real-time collision detection exploits *temporal coherence*. Though there are different interpretations of using temporal coherence, our approach is inspired from [LC92, DG04], to track pairs of closest points between colliding bodies are tracked over time. The main differences here are : (1) the interacting objects have a tubular (intestine) or a membrane-like structure (mesentery), and (2) most collisions will be self-collisions between different folds of the same body. We first explain the collision detection method for the intestine alone, and then explain the mesentery case.

Collision detection between cylinders can be processed by computing the closest distance between their axes [Ebe00], and comparing them to the sum of their radii. For the intestine, computing the distance between two segments is done by considering the distance between their principal axes (a segment here refers to a simple line segment and it's end-points are parameterized by $(s,t) \in [0,1]$). We then store the (s,t) value corresponding to the closest point within the segments and the actual minimum distance $d_{\min}$. Recall that the rendering model uses an adaptive spline interpolation [GM03] of these skeletal segments with fixed initial length giving it a smooth appearance.

Adapting the notion of "closest element pairs" to this skeleton curve means that we track the local minima of the distance between non-neighboring segments along the curve (see Fig. 3.6). Of course, only the local minima satisfying a given distance threshold are

considered relevant. We refer to these pairs of segments as *active pairs*. Each active pair is locally updated at each time step in order to track the local minima when the intestine folds move. This is done by checking whether it is the current segment pair or a pair formed using one of their neighbors now corresponds to the smallest distance. This update requires *nine* distance tests (see Fig. 3.7), and the pair of segments associated with the closest distance becomes the new active pair. When two initially distant active pairs converge to the same local minimum, one of them is suppressed. A pair is also suppressed if the associated distance is greater than a given threshold. The above process tracks the existing regions of interest

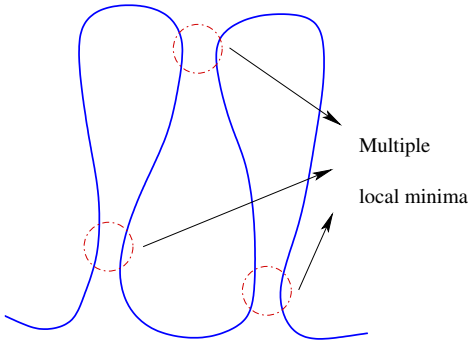


Fig. 3.6: Tracking the local minima of distance between non-neighboring segments.

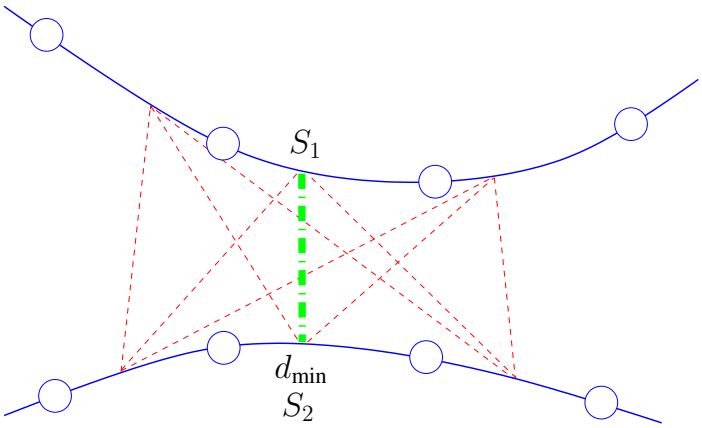


Fig. 3.7: Distance computation between two intestine segments.

but does not detect new ones. Since the animation of the intestine may create new collisions, a method for creating new active pairs of segments is needed. Our method is inspired from the stochastic approach of [DG04]. At each time step, in addition to the update of the currently active pairs, n additional random pairs of segments, uniformly distributed between the end-points, but under the distance threshold are generated. The update of these extra active pairs is similar to the update of the existing local minima. The complexity of the detection process thus linearly varies with the user-defined parameter n. At each time step, collision detection consists of selecting among the currently active pairs, the pairs of segments which are closer than the sum of their radii. Collision response will then be applied between these segments.

For the mesentery, the total number of segments to be considered during each time-step

is too large for real-time computation. We use the following approximations to reduce the complexity of the problem. Firstly, since the mesentery is very thin and soft compared to the intestine, its self-collisions will almost have no effect on the overall behavior of the system. Hence, we ignore the testing of these and only consider the non-trivial cases of intestine-intestine and intestine-mesentery interactions.

Secondly, we use a two-step convergence algorithm to reduce the number of distance computations required for the mesentery. Accordingly, given a segment pair (S_1, S_2), we first find if there exists another segment $S_1^{'}$ that is the closest to S_2 ($S_1^{'}$ is S_1 if all other neighbors are further from S_2). Then, we find a segment $S_2^{'}$ that is the closest to $S_1^{'}$. This update requires *thirteen* distance computations at most (i.e. between an intestine segment and a non-neighboring mesentery segment). When a collision is detected, we apply a response force not only to the deepest penetrating segment-pair but also to the entire collision area (both for intestine and mesentery). A recursive algorithm searches the neighbors to find all the colliding pairs in the area.

3.2.4.2 Collision Response

We initiate the response whenever the distance between the two segments is less than the sum of their radii. The earlier approaches such as the penalty method [BW98] and the reaction constraint method [PB88] implemented collision response by altering the force matrix in the mass-spring method. This force has to be of large magnitude in order to be effective in large time-step scenarios. However, this may cause segment displacements several times larger than their thickness thus creating new collisions and instabilities. Instead, our new method alters the displacements and velocities such that it instantaneously cancels the interpenetration while keeping a resting contact between the two colliding bodies with no bouncing effects.

Let the end-point velocities of segment S_1 be $\mathbf{v}_1$ and $\mathbf{v}_1^{'}$ and that of segment S_2 be $\mathbf{v}_2$ and $\mathbf{v}_2^{'}$ respectively. Let $\mathbf{x}_1$, $\mathbf{x}_1^{'}$, $\mathbf{x}_2$ and $\mathbf{x}_2^{'}$ be the corresponding end-point positions. Let $\mathbf{v}_{c1}$ and $\mathbf{v}_{c2}$ be the velocities of the closest approaching points within each segment and $\mathbf{x}_{c1}$ and $\mathbf{x}_{c2}$ be the positions of the closest points (see Fig. 3.8). Let $\overline{s} = 1 - s$ and $\overline{t} = 1 - t$. We have :

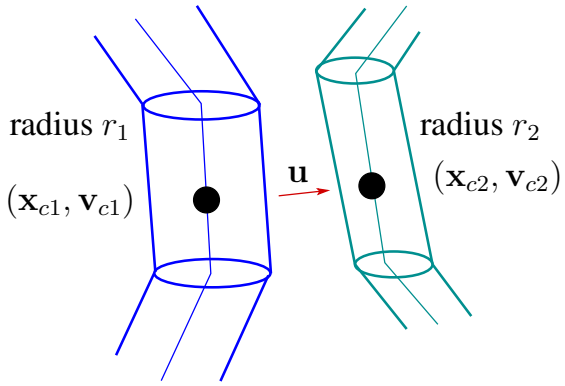


Fig. 3.8: Collision response by displacement-velocity correction.

$$\begin{aligned} \mathbf{v}_{c1} &= \overline{s}\mathbf{v}_1 + s\mathbf{v}_1^{'} \\ \mathbf{v}_{c2} &= \overline{t}\mathbf{v}_2 + t\mathbf{v}_2^{'} \end{aligned} \tag{3.1}$$

Let two impulses per time-step, f and $f'(=-f)$ (one for each segment), be applied along the direction of collision $\mathbf{u}$ to cause a velocity change such that the relative velocities along $\mathbf{u}$ is zero. These impulses should set the new velocities $\mathbf{v}_{newc1}$ and $\mathbf{v}_{newc2}$ such that :

$$(\mathbf{v}_{newc1} - \mathbf{v}_{newc2})\cdot\mathbf{u} = \mathbf{0}^5 \tag{3.2}$$

This cancels the penetration velocity and avoids any bouncing. The impulse f acting on the point of collision can be split between the end-points according to their barycentric coordinates. Then we have :

$$\begin{aligned} \mathbf{v}_{new1} &= \mathbf{v}_1 + \frac{\overline{s}f}{m_1}\mathbf{u} \\ \mathbf{v}_{new1}^{'} &= \mathbf{v}_1^{'} + \frac{sf}{m_1^{'}}\mathbf{u} \\ \mathbf{v}_{new2} &= \mathbf{v}_2 + \frac{\overline{t}f}{m_2}\mathbf{u} \\ \mathbf{v}_{new2}^{'} &= \mathbf{v}_2^{'} + \frac{tf}{m_2^{'}}\mathbf{u} \end{aligned} \tag{3.3}$$

where m_1, $m_1^{'}$, m_2 and $m_2^{'}$ are the masses of the end-points. Again, expressing the new velocity of the colliding point $\mathbf{v}_{newc1}$ in terms of $\mathbf{v}_{new1}$ and $\mathbf{v}_{new1}^{'}$:

$$\begin{aligned} \mathbf{v}_{newc1} &= \overline{s}\mathbf{v}_{new1} + s\mathbf{v}_{new1}{}' \\ &= \mathbf{v}_{c1} + (\frac{\overline{s}^2}{m_1} + \frac{s^2}{m_1^{'}})f\mathbf{u} \end{aligned} \tag{3.4}$$

Similarly for segment S_2 :

$$\mathbf{v}_{newc2} = \mathbf{v}_{c2} - (\frac{\overline{t}^2}{m_2} + \frac{t^2}{m_2^{'}})f\mathbf{u} \tag{3.5}$$

Substituting (3.4) and (3.5) into (3.2), we have :

$$f = \frac{(\mathbf{v}_{c2} - \mathbf{v}_{c1})\cdot\mathbf{u}}{\frac{\overline{s}^2}{m_1} + \frac{s^2}{m_1^{'}} + \frac{\overline{t}^2}{m_2} + \frac{t^2}{m_2^{'}}} \tag{3.6}$$

Using this value of f, we compute the new velocities of the end-points from (3.3). We use a similar formulation for correcting the positions of the colliding segments. The only difference is in the condition for avoiding interpenetration, which considers the segment radii r_1 and r_2 :

$$(\mathbf{x}_{newc1} - \mathbf{x}_{newc2})\cdot\mathbf{u} = r_1 + r_2 \tag{3.7}$$

[5] where $\cdot$ denotes a vector dot product.

The integrated impulse value g that changes the positions in accordance with the above condition is then :

$$g = \frac{(\mathbf{x}_{c1} - \mathbf{x}_{c2})\cdot\mathbf{u} + r_1 + r_2}{\frac{\overline{s}^2}{m_1} + \frac{s^2}{m_1'} + \frac{\overline{t}^2}{m_2} + \frac{t^2}{m_2'}} \tag{3.8}$$

g is used to modify the end-point positions as in (3.3).

We note that updating the position and the velocity of any of the edge may still create or cancel collisions among other segments. A possible solution is to repeat the collision check for all the pairs in our list in order to identify such pairs. But this may result in an endless loop. So in our implementation, we prefer using a fixed number of iterations even if we miss some collisions during the current time-step. Handling multiple collisions in general is a difficult problem even for rigid bodies with no straightforward solution as we noted in the previous chapter (also cf. § 3.2.6).

3.2.5 Results

Snapshots from the real-time animation of the simulator are shown in Fig. 3.9. In all cases, the organs were subject to forces due to gravity, user-input and collisions. Fig. 3.9(i) shows the case of an isolated intestine and mesentery manipulated by a virtual probe (the tiny sphere). Note that our displacement-velocity method for collision response produces fairly stable simulations (see Fig. 3.9(ii)) with some undesired vibrations. Figures 3.9(iii) and (iv) show the simulations inside a simulated abdominal cavity. All the snapshots were captured from our simulator in real-time on a standard PC with Bi-Athlon ® 1.2 GHz, 512 MB RAM and nVIDIA ® GeForce ® 3 graphics card.

Quantitative Validation : The mechanical and collision detection model were implemented and tested independently prior to the integration. The results were fast enough to run at 30 Hz on a standard PC and the overall behavior was good. However, due to the stochastic nature of our algorithm, we do not have any theoretical proof that all the collisions are detected. However, in Fig. 3.10, we present one of the several evaluations which we performed on the collision processing algorithm.

Here, as the object is deformed, we plotted the colliding regions as detected by our method and a naïve $O(n^2)$ method. Results show that our method is able to identify all the *active* colliding regions by evaluating a far fewer number of points (69 active points as opposed to 237 $O(n^2)$ points in Fig. 3.10). Though not the same as finding the *exact* collisions, this is good enough for our case, since we can quickly narrow down to these points using temporal coherence. Nevertheless, we miss collisions in some cases between intestine-mesentery due to an entirely different reason (Fig. 3.9(iv)). The interpenetrations occurred because we performed only segment-segment collision detection. Note that this technique worked well for intestine-intestine collisions and we do not see any collisions missed here. But the mesentery was modeled as a triangulated mesh and when there are cases of segment-triangle or point-triangle collisions, the above approach in its present form was not effective. In addi-

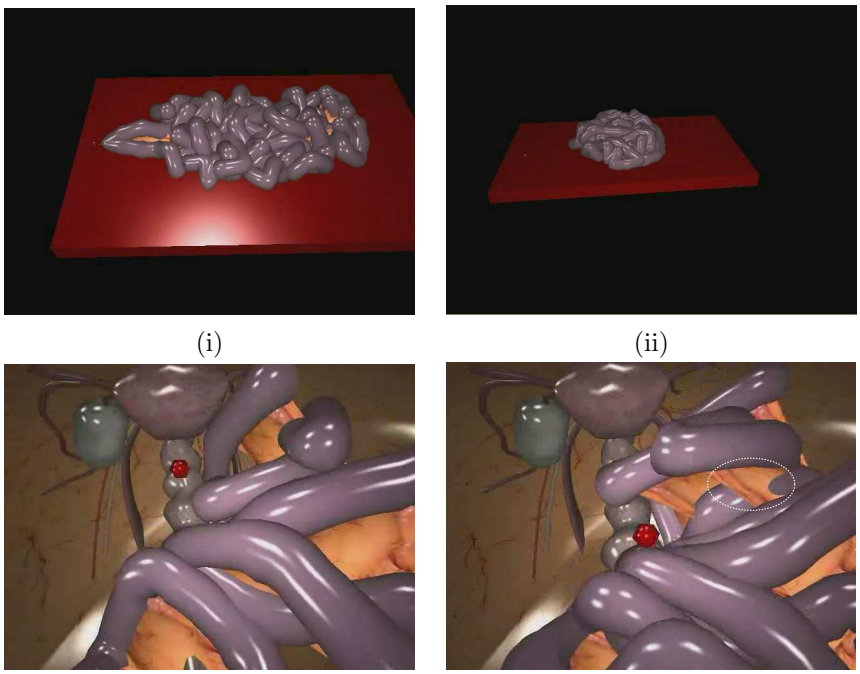


Fig. 3.9: Snapshots from our intestinal surgery simulator. (i) Intestine and mesentery on a plane pulled by a probe. (ii) Stable rest position. (iii) Inside the virtual abdominal cavity. (iv) Case when collision detection fails (see encircled region).

tion, the mesentery being a thin object would have required that the collisions be treated in a time-continuous manner using methods that will be described in chapter 4. The self-collisions within the mesentery too were not accounted for.

Qualitative Validation : The prototype simulator was demonstrated to the surgeons at IRCAD. The surgical educators practiced on our simulator and explained to us the good points as well as the drawbacks of the current model. The overall intestine behavior and contact modeling was observed to be very good. Some of the instabilities observed in certain cases of the intestine's motion actually turned out to simulate a patient who is spasmodic (suffering from intestinal convulsions). Although this happens quite frequently in a real patient, this problem can be fixed by increasing the damping of the system. They also suggested that the mesentery should be less elastic. With our mass-spring model, the characteristics demanded by the surgeons can be obtained simply by tuning the simulation parameters. Though these parameters can be varied intuitively, it is difficult to incorporate those parameters obtained from biometric studies. Finally, a small error was detected in the geometric design - the mesentery should have a zero width at the two extremities, where the intestine is directly attached to the main vessels.

3.2.6 Lessons from the intestine case study

Need for continuous collision : First, we note that for realistic simulation it is important

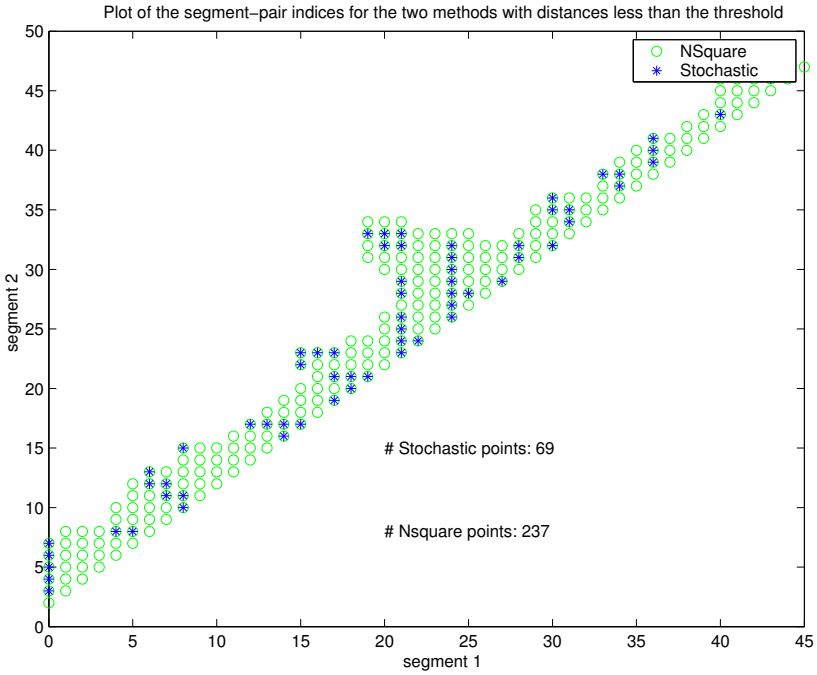


Fig. 3.10: Quantitative evaluation of the collision detection method. Points in green (light circles) detected by the $O(n^2)$ method and blue (dark points) by our approach.

to perform robust collision detection tests handling all the the primitive cases, i.e. vertex-triangle, edge-triangle and edge-edge. In addition, for thin objects we need to perform collision tests in a continuous manner - this will very well take care of the interpenetrations (such of intestine colliding with the thin mesentery) which occurred in this present model. To take care of this shortcoming, Chapter 4 will comprehensively describe the approaches for robust detection of collision between primitives both at discrete and continuous time instants.

Handling multiple collisions : Second, note that in this approach we had avoided the treatment of self-collisions within the mesentery. We were justified in this case since it is thin and light compared to the intestine, the collisions will not have much impact on to the overall simulation. Moreover, being located behind the intestine, most of the mesentery is hidden. Hence the visual impact too is less apparent. But there might be cases where we will have to treat the self-collisions in other thin tissues. For that we need a solution that can take care of the simultaneous multiple collisions. Chapter 5 will present new solutions to this problem which hopefully fulfill our needs.

3.3 Case Study #2 : Rigid Cable

We now present the case of a very stiff cable colliding with a rigid object such as a pulley. This problem was raised to us by a maker of commercial CAD/CAM software Solid Dynamics [6] which was interested in the simulation to demonstrate their software. We worked on this

[6]Solid Dynamics S.A., 42300 Roanne, FRANCE, `http ://www.solid-dynamics.fr`

problem between May 2005 to October 2005. Here the aim is to show :

- The possibility of developing specialized solvers for better efficiency
- The need for robust collision detection
- The shortcomings of the standard collision response methods

As in the intestine case, we first present the approach we took followed by experimental results and analysis.

3.3.1 Dynamics and Numerical Solver

Typical dynamics problems formulated as an ordinary differential equation (ODE) can be solved either by explicit integration techniques such as the forward Euler, Runge-Kutta, etc. or by implicit techniques like the backward Euler. The latter is especially useful for solving "stiff" set of equations and provides a stable solution even while simulating at large time-steps [BW98]. In our system we chose a mass-spring system with a set of masses m_1, m_2, ... , m_n connected by linear springs with stiffness constant $k_{1,2}$, $k_{2,3}$, ... , $k_{n-1,n}$ (see Fig. 3.11).

Typically we write the implicit Euler equation to solve for $\Delta\mathbf{v}$ as [BW98] :

$$\left(\mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}} - \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right) \Delta\mathbf{v} = \Delta t \left(\mathbf{f} + \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{v}} \mathbf{v}\right) \tag{3.9}$$

The above equation can be written in short form as :

$$\mathbf{K}\Delta\mathbf{v} = \mathbf{b} \tag{3.10}$$

We integrate the new velocities $\mathbf{v} + \Delta\mathbf{v}$ to get the new positions $\mathbf{x} + \Delta t(\mathbf{v} + \Delta\mathbf{v})$.

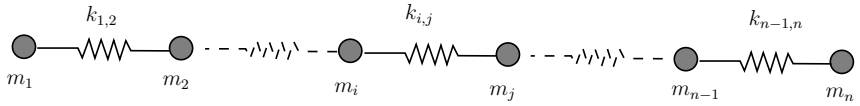


Fig. 3.11: Cable represented by a mass-spring system.

Though the above can be a large matrix system, the matrix can often be sparse. In addition it is also positive semi definite (PSD) and symmetric. Hence, Baraff used an iterative conjugate gradient approach for solving this for cloth. However, we found that such an iterative approach though very stable takes the worst case approach in the case of a 1 dimensional cable. For a mass-spring system with n particles, the CG solver usually iterates for n steps to find an acceptable solution (i.e. the residue value is less than the preset tolerance value). Hence it is worthwhile to examine other solvers.

We found that a banded LU solver represents well the structure of the cable. There are fast inversion algorithm which does the matrix inversion of a banded LU matrix. One disadvantage with this approach is that we would require an explicit representation of the stiffness matrix. This is in contrast with iterative approaches where we only need to provide the CG solver with a method to compute the product of the matrix $\mathbf{K}$ with a vector $\mathbf{x}$. But

fortunately the stiffness matrix for a cable tends to be sparse with non-zero elements in and around the diagonal. In fact, we found out that the matrix tended to be a block tridiagonal matrix. Further, since we know the size of the diagonal band in advance, we can use this information to create memory efficient matrices in our data structure (also cf. §5.2.4 for a discussion on exploiting sparse matrices).

3.3.1.1 Expressing K matrix in block tridiagonal system

For a cable, the structure of the **K** matrix looks as follows :

$$\begin{pmatrix} \mathbf{M}_1 + \mathbf{k}_{1,2} & -\mathbf{k}_{1,2} & & & & \\ -\mathbf{k}_{1,2} & \mathbf{M}_2 + \mathbf{k}_{1,2} + \mathbf{k}_{2,3} & -\mathbf{k}_{2,3} & & & \\ & -\mathbf{k}_{2,3} & \mathbf{M}_3 + \mathbf{k}_{2,3} + \mathbf{k}_{3,4} & & & \\ & & & \ddots & -\mathbf{k}_{n-1,n} & \\ & & & -\mathbf{k}_{n-1,n} & \mathbf{M}_n + \mathbf{k}_{n-1,n} & \end{pmatrix} \tag{3.11}$$

where each $\mathbf{M}_i$ is 3x3 mass matrix of the i^{th} particle :

$$\begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{pmatrix} \tag{3.12}$$

and $\mathbf{k}_{i,j}$ is the stiffness matrix representing the spring between particles i and j with stiffness value $k_{i,j}$, computed force $\mathbf{f}_{i,j}$, present length $l_{i,j}$ and gradient direction $(\mathbf{u}_{i,j})_{1\times 3}$ written as :

$$(\mathbf{k}_{i,j})_{3\times 3} = (k_{ij} - \frac{\parallel \mathbf{f}_{i,j} \parallel}{l_{i,j}})((\mathbf{u}_{i,j})_{3\times 1} \times (\mathbf{u}_{i,j}^T)_{1\times 3} + \frac{\parallel \mathbf{f}_{i,j} \parallel}{l_{i,j}}\mathbf{I}_{3\times 3}) \tag{3.13}$$

Note that the addition of two 3x3 matrices is also a 3x3 matrix. Hence the stiffness matrix of (3.11) is a *block tridiagonal matrix* which can be solved by LU forward-backward substitution as follows. Assuming that the matrix can be factorized into **L** and **U** using techniques described in [PTVF92] :

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{K} \tag{3.14}$$

such that (3.10) can be rewritten in LU form :

$$\mathbf{K} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b} \tag{3.15}$$

The solution can be found by first solving for a vector : **y** such that

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \tag{3.16}$$

and then solving for **x** :

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \tag{3.17}$$

We now present a *block tridiagonal* algorithm to find the solution $\mathbf{x}$. Let the $\mathbf{L}$ and $\mathbf{U}$ be written as :

$$\mathbf{L} = \begin{pmatrix} \mathbf{I} & & & \\ \mathbf{L}_2 & \mathbf{I} & & \\ & \ddots & & \\ & & \mathbf{L}_n & \mathbf{I} \end{pmatrix} \tag{3.18}$$

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}_1 & \mathbf{B}_1 & & \\ & \mathbf{U}_2 & \mathbf{B}_2 & \\ & & \ddots & \\ & & & \mathbf{U}_n \end{pmatrix} \tag{3.19}$$

The solution is found out by forward and backward substitution :

$$\mathbf{y}_1 = \mathbf{b}_1 \tag{3.20}$$

$$\mathbf{y}_i = \mathbf{b}_i - \mathbf{L}_i \mathbf{y}_{i-1}, i = 2, ..., n \tag{3.21}$$

$$\mathbf{U}_n \mathbf{x}_n = \mathbf{y}_n \tag{3.22}$$

$$\mathbf{U}_i \mathbf{x}_i = \mathbf{y}_i - \mathbf{b}_i \mathbf{x}_{i+1}, i = n-1, ..., 1 \tag{3.23}$$

We used the `C++` library TBCI [7] which provides tools for LU inversion and other linear algebra operations.

3.3.2 Fast Cable-Object Collision Detection

In the case of a cable (geometrically represented by a set of points connected with line segments) colliding a rigid object, we can take advantage of localized collision areas (unlike the intestine case) which can be quickly identified via using bounding volume hierarchical structure. Note that in this simulation we *do not* take into account of the self-collisions within the cable. Hence, we used a octree based BVH (discussed earlier in §2.3.3) for fast collision detection. The octree is subdivided using a pre-determined depth level `depthMax` (see Fig. 3.12).

We use a top-down traversal technique to nail down the final leaf node (until we reach `depthMax`) starting from the root. For discrete collision detection, we do a point-box test to check if the point $\mathbf{p} = (p_x, p_y, p_z)$ belonging to the cable lies inside the box bounded by $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max})$ or not. This can be done by doing a simple 6-plane tests to see if the point lies within the plane bounds in each direction, i.e. check if $x_{\min} \leq p_x \leq x_{\max}$ and similarly in y and z directions. If yes, we further descend to the child nodes of the box until we reach the leaf node.

Since we are also interested in continuous collisions, we have to do a ray-box test (in the

[7] Available from `http ://plasimo.phys.tue.nl/TBCI/`

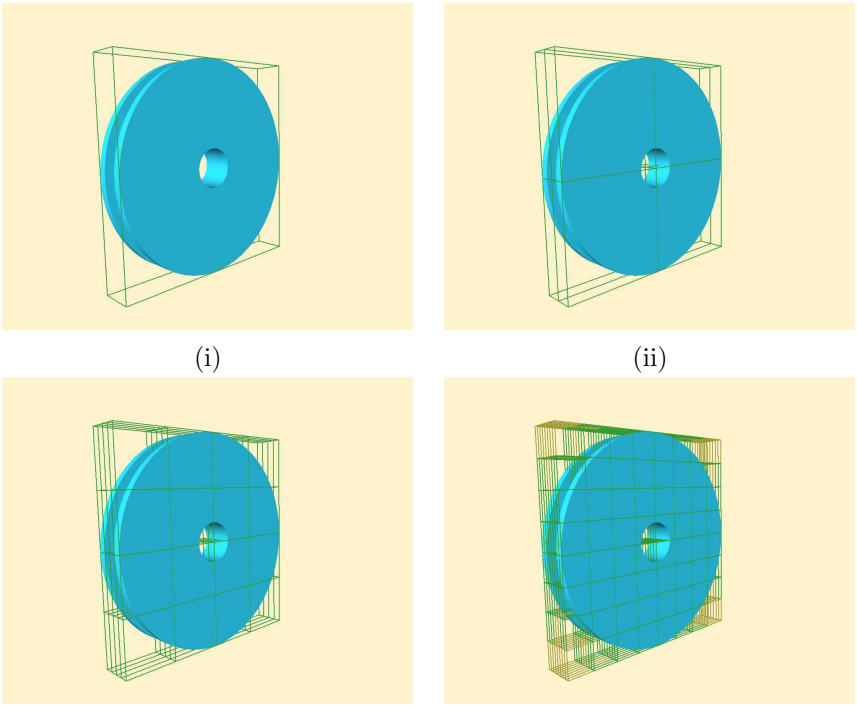


Fig. 3.12: (i) A rigid pulley with no octree subdivisions. (ii) Pulley with 1-level octree subdivision. (iii) Pulley with 2-level octree subdivision. (iv) Pulley with 3-level octree subdivision. 3D model of the pulley courtesy of Solid Dynamics S.A.

place of point-box test described above) to determine if a cable point crossed the box *during* the time-step. For ray-box tests, we perform three ray-plane tests described in [Gla89] to check if a ray defined by $\mathbf{o} + t\mathbf{d}$ lies within the plane bounds, where the origin is $\mathbf{o}$ is the position of the point $\mathbf{x}$ at time t_0 and the direction is $\mathbf{d} = \mathbf{x}(t_0 + \Delta t) - \mathbf{x}(t_0)$ (see Fig. 3.13). Note that for a generic case, we have to test the collision between a moving segment and a box as well. For that we have to consider the plane swept by the segment in a time step and test for its intersection with the fixed box.

Once we do the broad phase collision detection, we get to the narrow phase of the test where we examine the primitives concerned with the selected ray-box pairs. Accordingly, each ray is tested with all the triangles inside the box (see Fig. 3.14). We used the ray-triangle tests proposed by Moeller and Trumbore [MT97] for efficient computation. Here we get a cubic equation in time t which once solved is tested for a valid interval $t_c \in [t_0, t_0 + \Delta t]$. If yes, we then test if the point lies within the triangle at the time of collision. We refer to chapter 4 where we comprehensively treat the various possible collision cases.

3.3.3 Collision Response

We discussed several techniques in §2.4 for performing collision response. Here, we used an impulse based method for collision response. Once collision is detected the velocity of the colliding particle $\mathbf{v}_i$ is altered by deflecting it away towards the normal $\hat{\mathbf{n}}$ of the intersec-

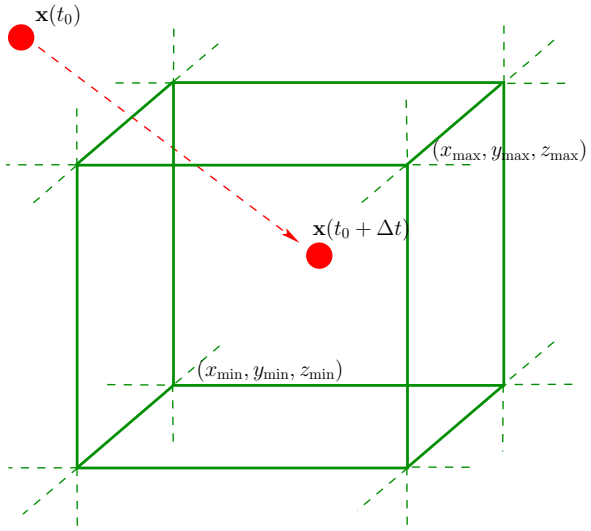


Fig. 3.13: Collision test between a ray and a box.

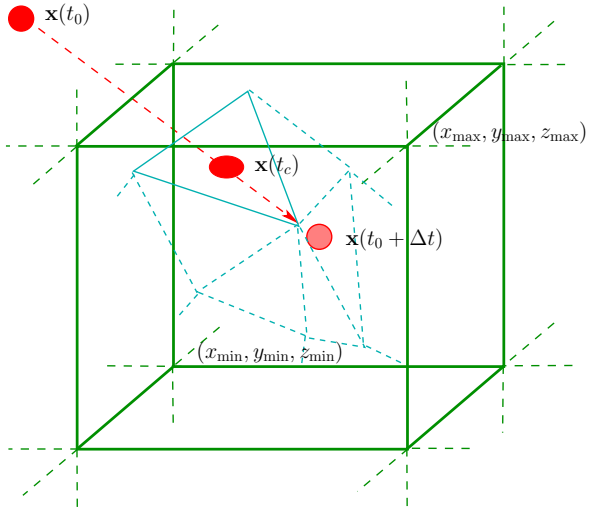


Fig. 3.14: Collision test between a ray and the triangles inside a box.

ting triangle. The magnitude of the impulse is the relative velocity projected in the normal direction. The new velocity is calculated as :

$$v_{\text{rel}} = \mathbf{v}_i \cdot \hat{\mathbf{n}} \tag{3.24}$$

$$\mathbf{v}_i + = v_{\text{rel}} \hat{\mathbf{n}} \tag{3.25}$$

3.3.4 Results

Results of Implicit + Conjugate : Table 3.1 shows the parameters of the dynamic simulation of the cable sliding over the fixed pulley. Fig. 3.15 shows the snap shots of the simulation as the cable with an initial velocity on one end falls under the influence of gravity and gradually slides. The simulations ran at 40 Hz in a Intel ® Pentium ® 4 3.0 GHz PC with 1GB

Tab. 3.1: Parameters used in the simulation shown in Fig. 3.15.

Parameter	Value
Number of pulley triangles	13722
Number of cable particles	100
Mass	0.7 kg
Stiffness	$1.38 \times 10^6 Nm^{-1}$
Gravity	$-10ms^{-2}$
Initial velocity	$-2.0ms^{-1}$

of RAM and GeForce ® 3 graphics card.

3.3.5 Problems

We now describe the problems with this collision response approach. Here, knowing well the high rigidity of the cable we did not attempt to do displacement correction as we did for intestine in § 3.2.4.2 since it would have led to instabilities. We only corrected the velocity using impulses. But still we encountered problems due the high stiffness. Fig. 3.16 illustrates this situation very well when we attempt to correct the interpenetration which occurred when the i^{th} particle interpenetrates the surface. While we attempt to correct, we also introduce additional strain rate and even interpenetration of the neighboring particles $i-1$ and $i+1$. This is because of large deformation forces generated by the instantaneous velocity correction and a high stiffness spring. Note the difference between strain and strain rate. Strain represents the change in the spring length due to deformation with respect to the initial rest length. Whereas strain rate represents the change in the deformation due to the new velocities with respect to the initial spring rest length.

Bridson et al. [BFA02] too noted this problem and suggested an explicit strain and strain-rate control mechanism. Accordingly, they used a Jacobi-based solution which simultaneously updated the velocities to limit the strain to a fixed limit relative to the initial spring rest length. For strain-rate, they used a Gauss-Seidel based solution which sequentially readjusted the particle velocities to limit the strain rate. This, while adding extra steps to the calculation also carries the risk of non-convergence. Though they attempt to iterate over a few times, they do not guarantee that such a mechanism actually converges. This again made us think of a global solution, which in addition to handling collision constrains can also handle additional constraints such as strain and strain-rate control.

3.3.6 Lessons from the rigid cable case

We note that the implicit Euler method with a banded LU formulation provides a stable solution to the difficult problem of rigid cables. The use of continuous collision detection accelerated by a fast octree-based BVH tests provides a rapid and accurate collision detection.

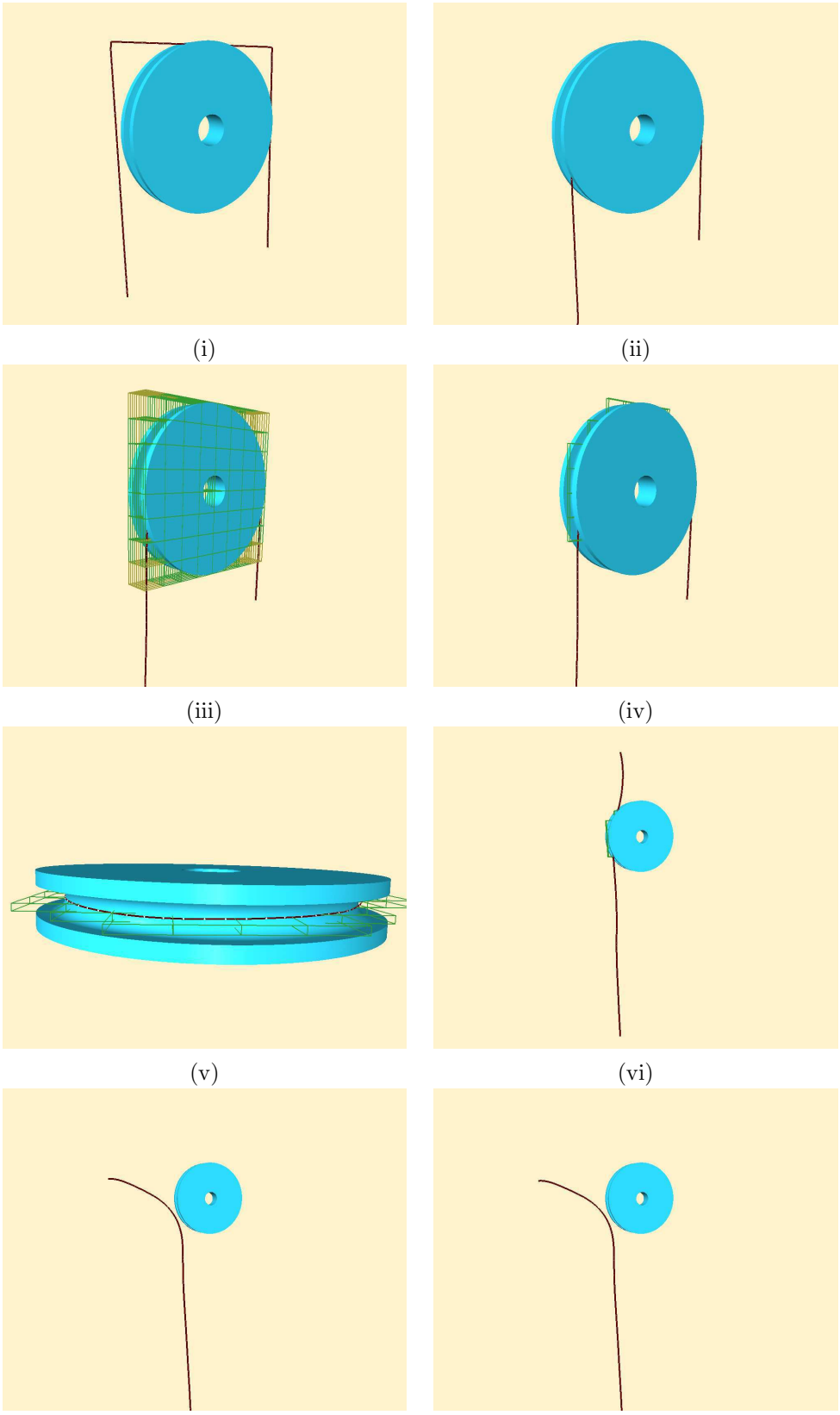


Fig. 3.15: A cable falling over a pulley with an initial velocity and and sliding over. (i) Initial non-colliding state. (ii) Cable falling over the pulley. (iii) Showing the bounding volume for broad phase collision detection. (iv) Showing the bounding volume nodes in collision with the cable. (v) View from top showing the BVH nodes in collision. (vi) Cable sliding over the pulley. (vii) & (viii) Cable sliding off the pulley and freely falling.

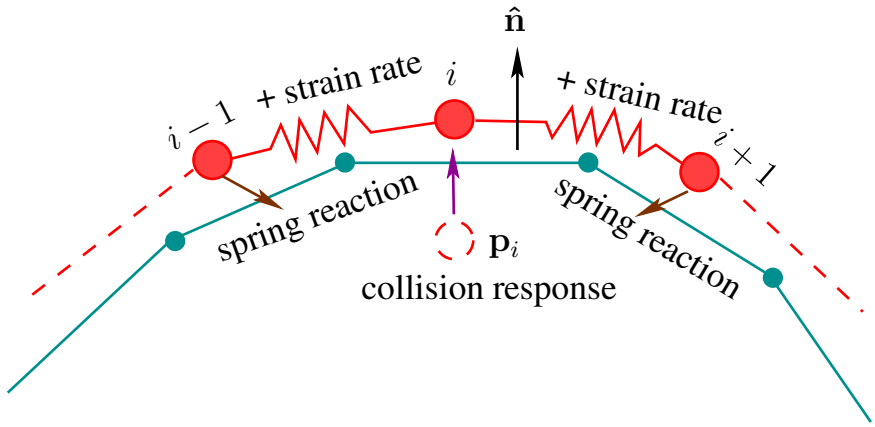


Fig. 3.16: Problems with collision response in case of a rigid cable (shown in red) colliding with a fixed object (shown in green).

While a local impulse method gave satisfactory results, it also introduces additional strain rate and possibly even new collisions. Even though explicit strain control techniques can be applied they also mean additional calculation. In addition there is no guarantee that such steps can converge. Thus, we are setting the stage for a global approach which could treat the collisions all at once while neither provoking new collisions and nor adding additional energy into the system. Such a method will be investigated in Chapter 5.

3.4 Chapter Summary

In this chapter, we illustrated two specific problems involving thin lineic or surfacic objects and came up with possible solutions. This study shows that there is probably no single best solution that works well for all the cases. In the case of the intestine, our geometric model while being simple was also fast enough for real-time simulation. This is a much better option than trying to extract the geometry via medical image segmentation - a very hard task given the complex geometry. Our mechanical model based on the mass spring model was fast enough to run on normal PCs.

For collision detection, the stochastic model gave efficient performance while not compromising on accuracy thus giving a satisfactory result when compared with the slower BVH updates. We also saw the need for robust continuous collision detection. For collision response, we saw how an impulse-based displacement and velocity impulse applied locally can give a fast and stable response. However for simulations which require collision handling for thin tissues, this will not give a satisfactory result and a more robust solution is needed.

In the case of the rigid cable, our approach was based on implicit Euler, octree-based broad phase test and continuous collision detection for narrow phase collision tests gave a fast and robust solution. Further, we illustrated how a different numerical solver can be applied for a more efficient calculation. Finally, for collision response, we saw how a local approach which

while being efficient also introduces additional strain rate into the system. Thus there is a case for a more robust collision handling scheme.

In the forthcoming chapters, we will propose solutions to the issues raised here.

CHAPTER 4

Robust Collision Detection for Thin Objects

4.1 Introduction

In the previous chapter we saw cases of two difficult collision scenarios. First that of an intestine colliding with a thin tissue and that of a stiff cable sliding over a rigid object. In both cases, we applied new approaches for modeling, animating and handling the collisions. However we also saw the short comings of the same. In particular, we saw how the robust detection of collision is important to maintain the realism of the simulation. In this chapter, we present and summarize robust methods based on detecting both at discrete and continuous instances. With the help of simple examples, we illustrate the methods which we will use in the next chapter as the basis for treating more complex, real-word examples.

In this chapter, we shall cover the following cases of collisions between primitives :

- *Point-Triangle :* Discrete collision between a vertex and a triangle (§4.2.1).
- *Moving Point - Fixed Triangle :* Continuous collision between a moving point and a fixed triangle (§4.2.2).
- *Moving Point - Moving Triangle :* Continuous collision between a moving point and a moving triangle (§4.2.3).
- *Edge-Edge :* Discrete collision between two edges (§4.2.5).
- *Moving Edge - Fixed Edge :* Continuous collision between a moving edge and a fixed edge (§4.2.6).
- *Moving Edge - Moving Edge :* Continuous collision between two moving edges (§4.2.7).

– *Coplanar Edges :* Continuous collision between two moving edges which are co-planar at the time of collision (§4.2.9).
– *Collinear Edges :* Continuous collision between two moving edges which are collinear at the time of collision (§4.2.10).

For real-world examples, we will use a combination of these utilities. For example for the cable over a fixed pulley example, we will use a combination of a moving point-triangle (the time sweep of a cable point at two successive time steps) to detect the fast-moving dynamic collisions and also the point-triangle to detect the proximity collisions as the cable slowly slides over the pulley.

4.2 Different Cases of Collision Detection

4.2.1 Point-Triangle

For a vertex $\mathbf{p}$ and triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ with normal $\hat{\mathbf{n}} = \frac{\mathbf{ab}\times\mathbf{ac}}{\|\mathbf{ab}\times\mathbf{ac}\|}$, let us elaborate on the method proposed in [BFA02] which we introduced in §2.3.8. Accordingly, we first check if $|\mathbf{pc}\cdot\hat{\mathbf{n}}| < h$, where $\mathbf{pc} = \mathbf{c}-\mathbf{p}$ and h is the thickness. If yes, we find the intersecting barycentric coordinates w_1, w_2, w_3 by first projecting $\mathbf{p}$ on to the plane containing the triangle. The equation of the point $\mathbf{p}$ lying on the plane defined by a point $\mathbf{x}$ and normal $\hat{\mathbf{n}}$ is then :

$$\hat{\mathbf{n}} \cdot (\mathbf{x} - \mathbf{p}) = 0 \tag{4.1}$$

Further, if the point $\mathbf{x}$ has to lie inside the triangle it should satisfy :

$$\mathbf{x} = w_1\mathbf{a} + w_2\mathbf{b} + w_3\mathbf{c} \tag{4.2}$$

where $w_3 = 1 - w_1 - w_2$. Thus they can be rearranged as :

$$w_1\mathbf{ac} + w_2\mathbf{bc} = \mathbf{pc} \tag{4.3}$$

Or,

$$\begin{bmatrix} \mathbf{ac}\cdot\mathbf{ac} & \mathbf{ac}\cdot\mathbf{bc} \\ \mathbf{ac}\cdot\mathbf{bc} & \mathbf{bc}\cdot\mathbf{bc} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \mathbf{ac}\cdot\mathbf{pc} \\ \mathbf{bc}\cdot\mathbf{pc} \end{bmatrix} \tag{4.4}$$

The actual intersection point $\mathbf{p}_i$ is then determined as $w_1\mathbf{a} + w_2\mathbf{b} + w_3\mathbf{c}$. The signed distance $d_{\min}$ can then be computed as $(\mathbf{p} - \mathbf{p}_i)\cdot\hat{\mathbf{n}}$ (see Fig. 4.1).

Experimental Validation : We now present the results of the discrete collision detection between a point and a triangle (see Fig. 4.2). A point with position $\mathbf{p} = (0.1, -0.0076, -0.1)$ collides with a triangle with coordinates $\mathbf{a} = (-0.2, -0.05, -0.2)$, $\mathbf{b} = (0.2, 0.0, 0.2)$ and $\mathbf{c} = (0.2, 0.0, -0.2)$ with $\hat{\mathbf{n}} = (0, 1, 0)$ and thickness $h = 0.01$. Using the above algorithm, the barycentric coordinates of the point of collision is found as $w_1 = .25$, $w_2 = .25$ and $w_3 = .5$. Then the point of collision $w_1\mathbf{a} + w_2\mathbf{b} + w_3\mathbf{c}$ is $\mathbf{p}_i = (0.1, -0.01246, -0.1)$. The minimum

distance $d_{\min}$ is verified as 0.0048 which is less than the thickness 0.01 thus registering a proximity collision.

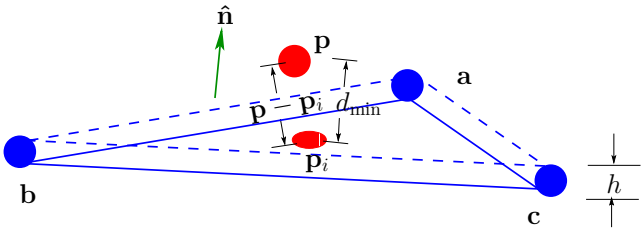


Fig. 4.1: Computing the signed distance between a colliding point and a triangle.

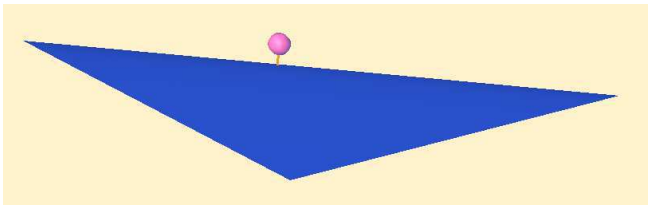


Fig. 4.2: Experiment for computing the signed distance between a colliding point and a triangle.

4.2.2 Moving Point - Fixed Triangle

We introduced the need for continuous collision detection in §2.3.7. For detecting the continuous collision between a moving point and a fixed triangle, we use the memory efficient method proposed by Möller and Trumbore [MT97]. The aim is to determine if there is a *crossing* between the vertex and the triangle *during* the time step. Here point $\mathbf{p}(t_0)$ at time t_0 displaces to $\mathbf{p}(t_0 + \Delta t)$. This can be described as a ray $\mathbf{r}(t)$ with origin $\mathbf{o}$ and a normalized direction $\mathbf{d}$:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}. \tag{4.5}$$

Where the origin $\mathbf{o}$ is the position at time t, $\mathbf{p}(t)$ and the direction $\mathbf{d}$ is the vector between the positions at time $t_0 + \Delta t$ and t_0, i.e. $\mathbf{d} = \mathbf{p}(t_0 + \Delta t) - \mathbf{p}(t_0)$. They then transform the origin of the ray yielding a vector containing the distance t to the intersection coordinates (u, v) within the triangle. If we describe the triangle with three vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, then the point $\mathbf{i}(u, v)$ on a triangle is given by :

$$\mathbf{i}(u, v) = (1 - u - v)\mathbf{a} + u\mathbf{b} + v\mathbf{c} \tag{4.6}$$

where (u, v) are the barycentric coordinates which must satisfy $u \geq 0$, $v \geq 0$ and $u + v \leq 1$. Now computing the intersection between the ray $\mathbf{r}(t)$ and the point on the triangle $\mathbf{i}(u, v)$ gives :

$$\begin{bmatrix} -\mathbf{d} & \mathbf{b} - \mathbf{c} & \mathbf{c} - \mathbf{a} \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \mathbf{o} - \mathbf{a} \tag{4.7}$$

We can now solve for the barycentric coordinates (u, v) and the distance t from the ray origin to the intersection by finding a solution to this linear system of equation. Geometrically, this can be conceived as the translation of the triangle to the origin and transforming t to a unit triangle y and z with the ray direction projected in the x direction. Fig. 4.3 illustrates this transformation where $\mathbf{M} = [-\mathbf{d}, \mathbf{b} - \mathbf{a}, \mathbf{c} - \mathbf{a}]$.

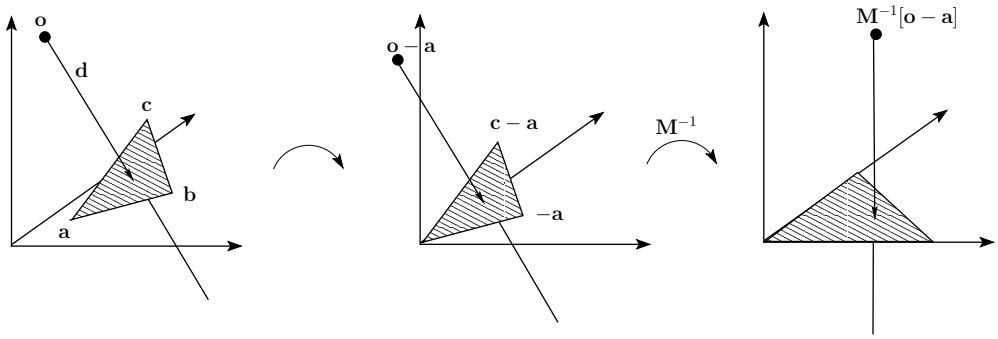


Fig. 4.3: Efficient ray-triangle intersection by translation and changing the base of the ray origin [MT97].

The solution to (4.7) is :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-\mathbf{d}, \mathbf{e}_1, \mathbf{e}_2|} \begin{bmatrix} |\mathbf{e}_3, \mathbf{e}_1, \mathbf{e}_2| \\ |-\mathbf{d}, \mathbf{e}_3, \mathbf{e}_2| \\ |-\mathbf{d}, \mathbf{e}_1, \mathbf{e}_3| \end{bmatrix} \tag{4.8}$$

where $\mathbf{e}_1 = \mathbf{b} - \mathbf{a}$, $\mathbf{e}_2 = \mathbf{c} - \mathbf{a}$ and $\mathbf{e}_3 = \mathbf{o} - \mathbf{a}$. Since determinant of three vectors $|\mathbf{x}, \mathbf{y}, \mathbf{z}| = -(\mathbf{x} \times \mathbf{z}) \cdot \mathbf{y} = -(\mathbf{z} \times \mathbf{y}) \cdot \mathbf{x}$, (4.8) can be rewritten as :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{e}_1} \begin{bmatrix} (\mathbf{e}_3 \times \mathbf{e}_1) \cdot \mathbf{e}_2 \\ (\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{e}_3 \\ (\mathbf{e}_3 \times \mathbf{e}_1) \cdot \mathbf{d} \end{bmatrix} \tag{4.9}$$

This notation will help in reusing the factors $\mathbf{e}_3 \times \mathbf{e}_1$ and $\mathbf{d} \times \mathbf{e}_2$ which will speed-up computation and decrease memory usage.

Experimental Validation : We now present the results of the continuous collision detection between a ray and a triangle. A point with position $\mathbf{p}(t_0) = (0.1, -0.0076 - 0.1)$ and $\mathbf{p}(t_0 + \Delta t) = (0.1, -0.022, -0.1)$ collides with a triangle during the time step with $\Delta t = 40ms$. The triangle coordinates are $\mathbf{a} = (-0.2, -0.05, -0.2)$, $\mathbf{b} = (0.2, 0.0, 0.2)$ and $\mathbf{c} = (0.2, 0.0, -0.2)$. Using the above algorithm, the time of collision is found as $t_c = .136ms \in [0, \Delta t]$ $(\Delta t = 40ms)$ and the barycentric coordinates as $u = .25$ and $v = .5$. Then the point of collision on the triangle $\mathbf{a} + u\mathbf{ab} + v\mathbf{ac}$ was verified as $(0.1, -0.0125, -0.1)$. This can again be confirmed to be the same by computing the position of the point at time t_c, i.e. $\mathbf{p}(t_0) + t_c\mathbf{v}(t_0)$ given $\mathbf{v}(t_0) = (0.0, -0.36, 0.0)$.

4.2.3 Moving Point - Moving Triangle

For a point $\mathbf{p}$ moving at a velocity $\mathbf{v}$ intersecting with a triangle represented by the points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ each moving with velocities $\mathbf{v}_a, \mathbf{v}_b, \mathbf{v}_c$ as shown in Fig. 4.4, the basic condition of intersection is :

$$\mathbf{p}(t) = \mathbf{a}(t) + u(\mathbf{ab}(t)) + v(\mathbf{ac}(t)) \tag{4.10}$$

where $\mathbf{p}(t) = \mathbf{p}(t_0) + t\mathbf{v}(t_0)$. We used the simplification by [Pro97] who introduced another condition that at the time of collision the triangle normal $\mathbf{n}(t) = \mathbf{ab}(t) \times \mathbf{ac}(t)$ should be perpendicular to the vector $\mathbf{ap}(t)$ (or the four points are co-planar) giving :

$$(\mathbf{ab}(t_c) \times \mathbf{ac}(t_c)) \cdot \mathbf{ap}(t_c) = 0 \tag{4.11}$$

This yields a *third* degree equation in t. A solution t_c should satisfy $t \in [t_0, t_0 + \Delta t]$. The value is plugged back in (4.10) to find the valid barycentric coordinates $(u, v) \in [0, 1]$.

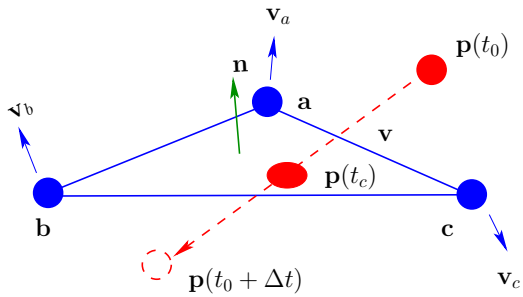
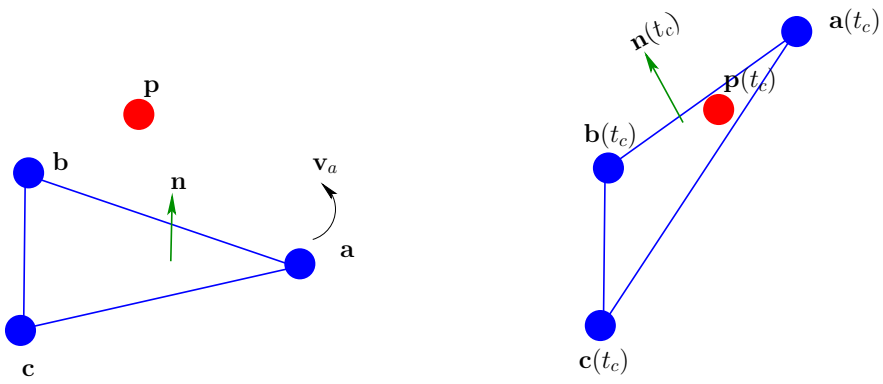


Fig. 4.4: Continuous collision detection of a vertex intersecting with a triangle.

Experimental Validation : We now present the results of the continuous collision detection between moving point and a moving triangle. A point with position $\mathbf{p} = (0.03, 0.248, 0.15)$ moving at a velocity $\mathbf{v} = (0.0, 0.418, 0.0)$ collides with a triangle during the time step with $\Delta t = 40ms$. The triangle coordinates are $\mathbf{a} = (0.0, 0.238, 0.1)$, $\mathbf{b} = (0.0, 0.238, 0.2)$ and $\mathbf{c} = (0.1, 0.238, 0.1)$ moving at velocities $\mathbf{v}_a = (0.0, 0.82, 0.0)$, $\mathbf{v}_b = (0.0, 0.67, 0.0)$ and $\mathbf{v}_c = (0.0, 0.77, 0.0)$. Using the above algorithm, the time of collision is found as $t_c = 28.8ms$, and the barycentric coordinates $u = .2$ and $v = .5$. Then the point of collision $\mathbf{a}(t_c) + u\mathbf{ab}(t_c) + v\mathbf{ac}(t_c)$ was found to be $(0.03, 0.26, 0.15)$. This can again be confirmed to be the same by computing the position of the point at time t_c, i.e. $\mathbf{p} + t_c\mathbf{v}$.

4.2.4 A Note on Normal Computation

Note that in some cases, we might have to recompute the value of the normal at time t_c. This is illustrated in Fig. 4.5 where the triangle intersects with the point at time t_c. If we need to apply a collision response, it is important to apply it in the correct normal direction. Here, we cannot use the normal at time t_0 (see Fig. 4.5(i)). The correct value of the normal should be $\mathbf{n}(t_c) = \mathbf{ab}(t_c) \times \mathbf{ac}(t_c)$ (see Fig. 4.5(ii)). Also §5.3.3.3 where we apply the re-computed normal for correct collision response.



(i) At time t (ii) At time t_c

Fig. 4.5: Normal computation during continuous collision detection between a vertex and a triangle.

4.2.5 Edge - Edge

The edges **ab** and **cd** are checked for parallel case by checking if $|\mathbf{ab} \times \mathbf{cd}| \neq 0$ (we deal with parallel edges in §4.2.9 and §4.2.10). For such non-parallel cases the barycentric values u, v are found as :

$$\begin{bmatrix} \mathbf{ab}\cdot\mathbf{ab} & -\mathbf{ab}\cdot\mathbf{cd} \\ -\mathbf{ab}\cdot\mathbf{cd} & \mathbf{cd}\cdot\mathbf{cd} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \mathbf{ab}\cdot\mathbf{ac} \\ -\mathbf{cd}\cdot\mathbf{ac} \end{bmatrix} \tag{4.12}$$

The closest point on the edge **ab** is $\mathbf{p}_1 = \mathbf{a} + u\mathbf{ab}$ and that on the edge **cd** is $\mathbf{p}_2 = \mathbf{c} + v\mathbf{cd}$. The minimum distance between them is then computed as $d_{\min} = |\mathbf{p}_1 - \mathbf{p}_2|$ (see Fig. 4.6). A collision is registered if the distance is less than the sum of the radii, i.e. $d_{\min} < r_1 + r_2$.

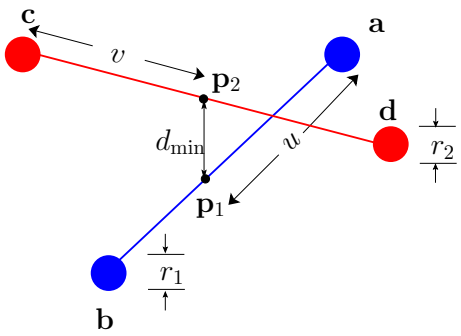


Fig. 4.6: Discrete collision detection between two edges.

Experimental Validation : We now present the results of the discrete collision detection between two edges (see Fig. 4.7). An edge with end-point coordinates $\mathbf{a} = (0, 1, 0)$ and $\mathbf{b} = (1, 1, 0)$ is in the proximity with another edge $\mathbf{c} = (0.5, 1.06, -0.3)$ and $\mathbf{d} = (0.5, 1.06, 0.5)$ Using the above algorithm, the barycentric coordinates were determined as, $u = .5$ and $v = .375$. Then the closest point on the first edge $\mathbf{a} + u\mathbf{ab}$ was found as $(0.5, 1, 0)$ and that on the second edge $\mathbf{c} + v\mathbf{cd}$ was found as $(0.5, 1.06, 0)$. The distance minimum $d_{\min}$ between them was thus verified to be 0.06.

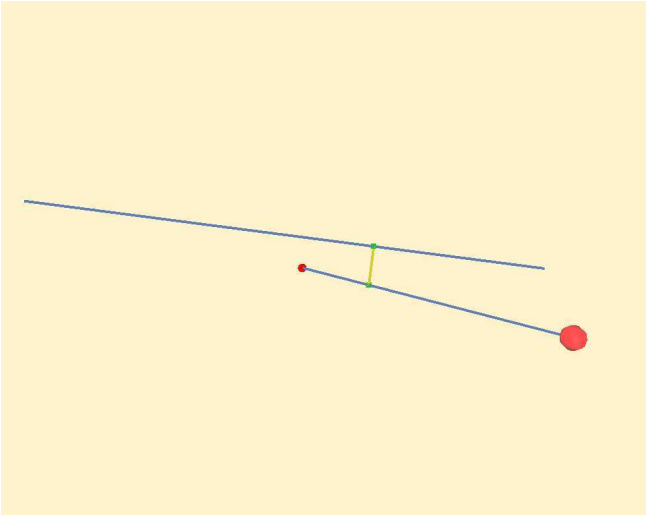


Fig. 4.7: Experiment for discrete edge edge collision detection.

4.2.6 Moving Edge - Fixed Edge

For testing the continuous collision between an edge **cd** with end-point velocities $\mathbf{v}_c$ and $\mathbf{v}_d$ with a fixed edge **ab**, we take account of the swept plane between positions at t_0 and the provisory positions $\mathbf{c}(t_0 + \Delta t)$ and $\mathbf{d}(t_0 + \Delta t)$ (see Fig. 4.8). We solve for the quadratic equation at a time of collision :

$$(\mathbf{ab} \times \mathbf{cd}(t)) \cdot \mathbf{ac}(t) = 0 \tag{4.13}$$

A valid time t_c should satisfy $t_c \in [t_0, t_0 + \Delta t]$. The colliding barycentric coordinates u, v are then found by plugging the t_c value to the condition at the point of collision :

$$\mathbf{a} + u\mathbf{ab} = \mathbf{c}(t_c) + v\mathbf{cd}(t_c) \tag{4.14}$$

The actual intersection point $\mathbf{p}_{\text{intersect}}$ is then found as $\mathbf{a} + u\mathbf{ab}$.

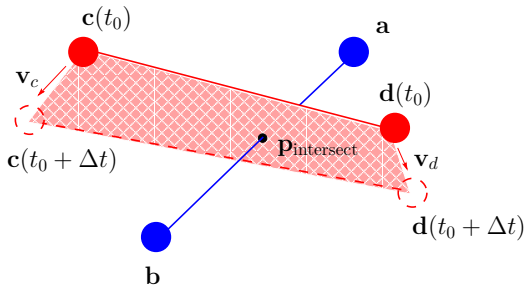


Fig. 4.8: Continuous collision detection between a moving edge and a fixed edge.

Experimental Validation : We now present the results of the continuous collision detection between a moving edge and a fixed edge. A fixed edge with end-point coordinates $\mathbf{a} = (0, 1, 0)$ and $\mathbf{b} = (1, 1, 0)$ collides with a moving edge $\mathbf{c} = (0.5, 1.06, -0.3)$ and $\mathbf{d} = (0.5, 1.06, 0.5)$ with

end-point velocities $\mathbf{v}_c = (0.0, -2.4, 0.0)$ and $\mathbf{v}_d = (0.0, -2.4, 0.0)$. Using the above algorithm, the time of collision was determined as $t_c = 25ms \in [0, \Delta t]$ (where $\Delta t = 40ms$) and the barycentric coordinates, $u = .5$ and $v = .375$. Then the intersection point $\mathbf{p}_{\text{intersect}}$ on the first edge $\mathbf{a} + u\mathbf{ab}$ is $(0.5, 1, 0)$. This is confirmed to be of the same value by computing the collision point on the second edge at the time of collision $\mathbf{c} + v(\mathbf{cd} + t_c\mathbf{v}_{cd})$.

4.2.7 Moving Edge - Moving Edge

For testing the collision between an edge $\mathbf{cd}$ with end-point velocities $\mathbf{v}_c$ and $\mathbf{v}_d$ with another edge $\mathbf{ab}$ with end-point velocities $\mathbf{v}_a$ and $\mathbf{v}_b$, we take account of the swept plane between positions at t_0 and the provisory positions $\mathbf{a}(t_0 + \Delta t)$, $\mathbf{b}(t_0 + \Delta t)$, $\mathbf{c}(t_0 + \Delta t)$ and $\mathbf{d}(t_0 + \Delta t)$ (see Fig. 4.8). We solve for the cubic equation at a time of collision :

$$(\mathbf{ab}(t) \times \mathbf{cd}(t)) \cdot \mathbf{ac}(t) = 0 \tag{4.15}$$

We used the method proposed in [Sch90]. A valid time t_c should satisfy $t_c \in [0, \Delta t]$. The colliding barycentric coordinates u, v are then found by plugging the t_c value to the condition a the point of collision :

$$\mathbf{a}(t_c) + u\mathbf{ab}(t_c) = \mathbf{c}(t_c) + v\mathbf{cd}(t_c) \tag{4.16}$$

The actual intersection point $\mathbf{p}_{\text{intersect}}$ is found as $\mathbf{a}(t_c) + u\mathbf{ab}(t_c)$.

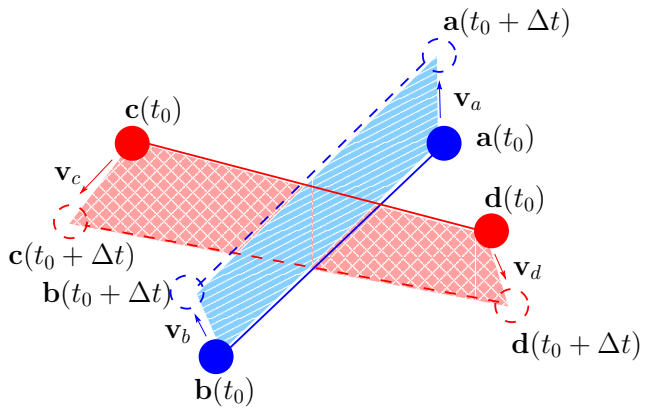


Fig. 4.9: Continuous collision detection between two moving edges.

Experimental Validation : We now present the results of the continuous collision detection of two moving edges. An edge with coordinates $\mathbf{a} = (0.0, 1.144, 0.0)$ and $\mathbf{b} = (0.0, 1.144, 0.0)$ with end-point velocities $\mathbf{v}_a = (0.0, 0.4, 0.0)$ and $\mathbf{v}_d = (0.0, 0.4, 0.0)$ collides with another edge $\mathbf{c} = (0.5, 1.204, -0.3)$ and $\mathbf{d} = (0.5, 1.204, 0.5)$ with end-point velocities $\mathbf{v}_c = (0.0, -1.6, 0.0)$ and $\mathbf{v}_d = (0.0, -1.6, 0.0)$. Using the algorithm, the time of collision was determined as $t_c = 30ms \in [0, \Delta t]$ (where $\Delta t = 40ms$) and the barycentric coordinates are $u = .5$ and $v = .375$. Then the collision point on the first edge $\mathbf{a} + u(\mathbf{ab} + t_c\mathbf{v}_{ab})$ is $(0.5, 1.56, 0.0)$. This is confirmed to be of the same value by computing the collision point on the second edge $\mathbf{c} + v(\mathbf{cd} + t_c\mathbf{v}_{cd})$.

4.2.8 Degenerate Edge Collisions

In the above case of moving edge - moving edge, we assume that we are able to formulate the problem as a cubic equation in t. But some times, we won't get a correct solution using the method described in §4.2.7. We first detail how to identify the degenerate cases in order to treat them in a special way. Expanding the cubic equation (4.15) we get :

$$((\mathbf{ab} + t\mathbf{v}_{ab}) \times (\mathbf{cd} + t\mathbf{v}_{cd})) \cdot (\mathbf{ac} + t\mathbf{v}_{ac}) = 0 \tag{4.17}$$

Or it can be factorized as :

$$C_0 + C_1 t + C_2 t^2 + C_3 t^3 = 0 \tag{4.18}$$

If the cubic part of (4.18) is zero (or in practice $-\epsilon < C_3 < \epsilon$), we treat the problem as a quadratic equation and find a solution. Further if even the quadratic equation becomes degenerate (or $-\epsilon < C_3, C_2 < \epsilon$), we treat it as a linear equation in t. Of course if all but the constant coefficient is zero (or $-\epsilon < C_3, C_2, C_1 < \epsilon$ and $|C_0| > \epsilon$), then there is no solution and hence no collision. But what if all the coefficients are zero or have a very small value, i.e. $-\epsilon < C_3, C_2, C_1, C_0 < \epsilon$. This is when the equation becomes degenerate. Fig. 4.10 illustrates the process of identifying the degenerate case. The equation becomes degenerate if either the velocity component of the cross-product or velocity component of the dot product in (4.17) is zero. This means that the two edges are co-planar or collinear at the time of collision. It is important to handle such cases since they will be undetected otherwise. We first describe the coplanar case in §4.2.9 and the collinear case in §4.2.10.

4.2.9 Coplanar Edges

First, we start by projecting the edges on to a 2D plane and try to find a common plane normal. If the latter exists we proceed with the steps described below. If there is no common plane normal $\mathbf{n}_p$, it means that the edges are collinear at the time of collision and need to treated differently. Fig. 4.11 shows the schema to take care of such a situation and Fig. 4.15 illustrates the overall method of handling the co-planar case. Note that in this section, we deal with degenerate case occurring in the continuous time instant. But the same principle can equally be applied to discrete time instants as well.

4.2.9.1 Finding the Plane Normal

Having presented the overview of handling the degenerate edge cases, we now present the co-planar case. We start by evaluating $\mathbf{ab} \times \mathbf{cd}$ as a possible plane normal. If it is null, we consider other planes such as $\mathbf{ab} \times \mathbf{ac}$ or $\mathbf{ab} \times \mathbf{ad}$. Fig. 4.12 illustrates the process of finding a suitable plane normal. Once we find a suitable plane with normal $\mathbf{n}_p$, we project (cf. §4.2.9.2) the 3D positions $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ and velocities $\mathbf{v}_a$, $\mathbf{v}_b$, $\mathbf{v}_c$ and $\mathbf{v}_d$ on to the selected plane. We then evaluate the collision time in this 2D space with the following cases :

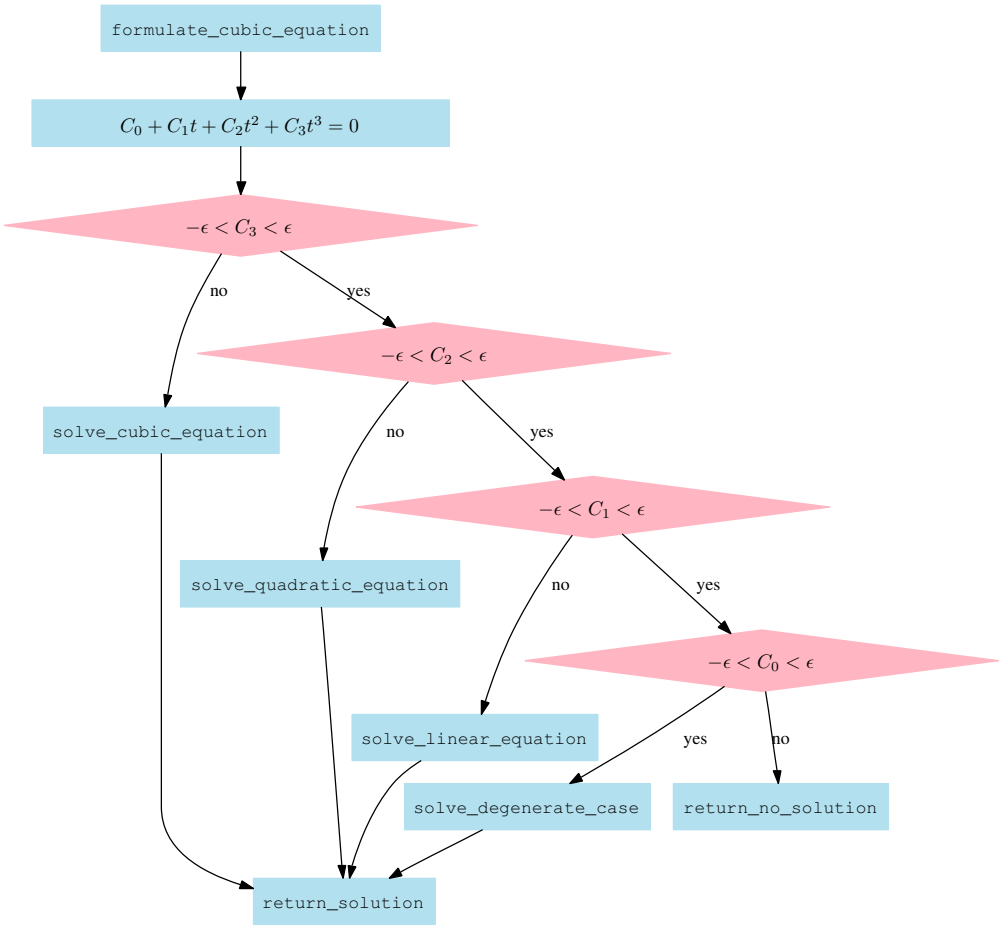


Fig. 4.10: Identifying degenerate cases for edge-edge collision.

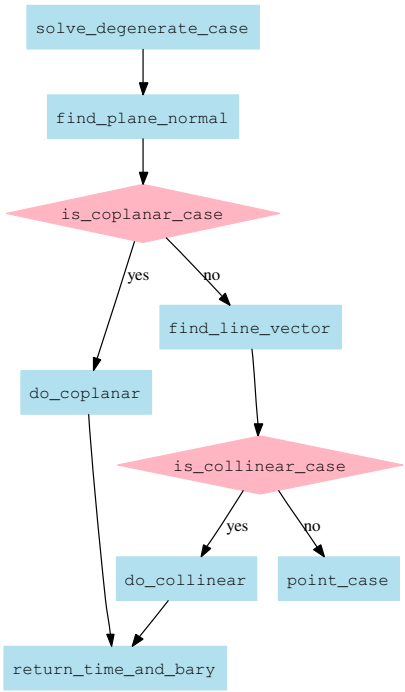


Fig. 4.11: High level flow chart for treating degenerate edge edge collisions.

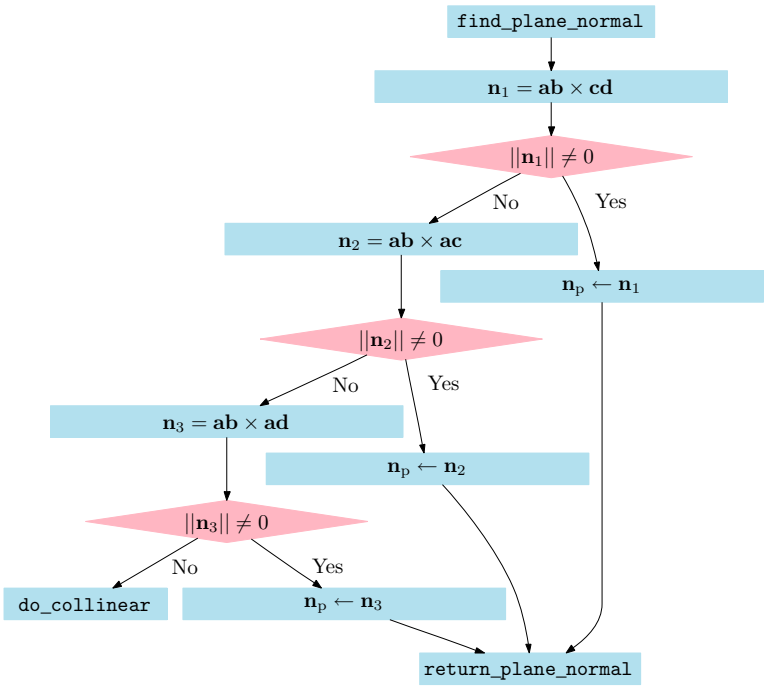


Fig. 4.12: Finding plane normal for co-planar edges.

- Collision between edge **ab** and end-point **c** (Fig. 4.13 (i))
- Collision between edge **ab** and end-point **d** (Fig. 4.13 (ii))
- Collision between edge **cd** and end-point **a** (Fig. 4.13 (iii))
- Collision between edge **cd** and end-point **b** (Fig. 4.13 (iv))

For each case, we solve for the quadratic equation to first find a valid time of collision between the two edges projected in 2D using (4.25). For example for case (i) we test if the edge $\mathbf{ab}_{\text{proj}}$ moving at a velocity $\mathbf{v}_{\text{abproj}}$ and a point $\mathbf{c}_{\text{proj}}$ moving at a velocity $\mathbf{v}_{\text{cproj}}$ (all values in 2D space) :

$$(\mathbf{ab}_{\text{proj}} + t\mathbf{v}_{\text{abproj}}) \times (\mathbf{c}_{\text{proj}} + t\mathbf{v}_{\text{cproj}}) = 0 \tag{4.19}$$

We similarly test the remaining cases as well. Since there could be multiple collisions which satisfy $t_c \in [t_0, t_0 + \Delta t]$ within a time-step, we will have to evaluate all four cases and find the first occurrence of collision, i.e. find the collision time with the least value. The valid time t_c is chosen by taking the minimum from up to four possible values, i.e. $t_c = \min\{t_1, t_2, t_3, t_4\}$.

The barycentric value on the edge for case (i) is then determined as :

$$\begin{aligned} u &= \frac{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ac}_{\text{proj}}}{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ab}_{\text{proj}}} \\ v &= 0 \end{aligned} \tag{4.20}$$

The formula for computing the barycentric coordinates for all the cases is presented in Table 4.1.

Tab. 4.1: Computation of barycentric coordinates for 2D edges.

Case	Barycentric values
(i) **ab** and **c**	$u = \frac{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ac}_{\text{proj}}}{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ab}_{\text{proj}}}, v = 0$
(ii) **ab** and **d**	$u = \frac{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ad}_{\text{proj}}}{\mathbf{ab}_{\text{proj}} \cdot \mathbf{ab}_{\text{proj}}}, v = 1$
(iii) **cd** and **a**	$u = 0,\ v = \frac{\mathbf{cd}_{\text{proj}} \cdot \mathbf{ca}_{\text{proj}}}{\mathbf{cd}_{\text{proj}} \cdot \mathbf{cd}_{\text{proj}}}$
(iv) **cd** and **b**	$u = 1, v = \frac{\mathbf{cd}_{\text{proj}} \cdot \mathbf{da}_{\text{proj}}}{\mathbf{cd}_{\text{proj}} \cdot \mathbf{cd}_{\text{proj}}}$

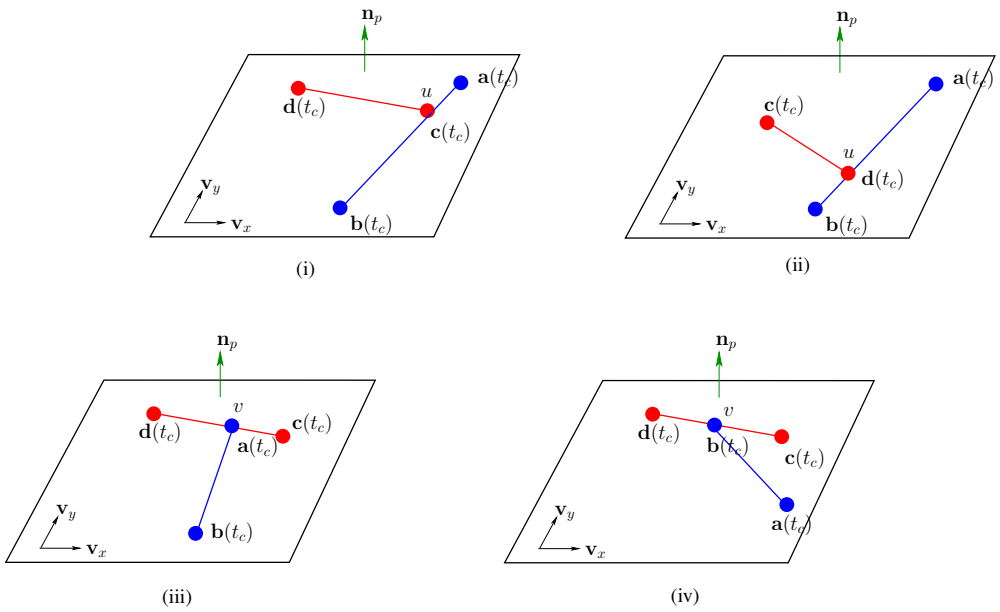


Fig. 4.13: Four possible cases when two edges collides in a plane.

4.2.9.2 Projecting Onto a Plane

Once a suitable plane vector $\mathbf{n}_p$ is found, we compute the projection of the positions and velocities on to this plane using the two orthogonal vectors $\mathbf{v}_x$ and $\mathbf{v}_y$ lying on this plane. First, $\mathbf{v}_x$ is computed by finding an arbitrary orthogonal vector to the plane normal $\mathbf{n}_p$, i.e.

$$\mathbf{v}_x = \mathbf{n}_p \times (1, 0, 0) \tag{4.21}$$

Or if $\mathbf{n}_p \times (1, 0, 0) = \mathbf{0}$,

$$\mathbf{v}_x = \mathbf{n}_p \times (0, 1, 0) \tag{4.22}$$

Or if $\mathbf{n}_p \times (0, 1, 0) = \mathbf{0}$,

$$\mathbf{v}_x = \mathbf{n}_p \times (0, 0, 1) \tag{4.23}$$

Then $\mathbf{v}_y$ is computed as :

$$\mathbf{v}_y = \mathbf{n}_p \times \mathbf{v}_x \tag{4.24}$$

Once $\mathbf{v}_x$ and $\mathbf{v}_y$ are found, a vector $\mathbf{vec}_{3d}$ in 3D space is transformed in the 2D plane represented by plane vectors $\mathbf{v}_x$ and $\mathbf{v}_y$ by :

$$vec_{2d} = (\mathbf{vec}_{3d} \cdot \mathbf{v}_x, \mathbf{vec}_{3d} \cdot \mathbf{v}_y) \tag{4.25}$$

We refer (4.25) as the method `project_vector_to_plane`. We apply the above transformation to the positions and velocities as follows. We first set the projection of $\mathbf{a}$ as :

$$\mathbf{a}_{\text{proj}} = (0, 0) \tag{4.26}$$

Then the rest of the positions are projected using (4.25) relative to $\mathbf{a}$ as :

$$\begin{aligned} \mathbf{b}_{\text{proj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{b} - \mathbf{a}) \\ \mathbf{c}_{\text{proj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{c} - \mathbf{a}) \\ \mathbf{d}_{\text{proj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{d} - \mathbf{a}) \end{aligned} \tag{4.27}$$

(4.26) and (4.27) represent the function `project_positions_to_plane` in Fig. 4.15. The velocities are similarly projected in a straight forward manner :

$$\begin{aligned} \mathbf{v}_{\text{aproj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{v}_a) \\ \mathbf{v}_{\text{bproj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{v}_b) \\ \mathbf{v}_{\text{cproj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{v}_c) \\ \mathbf{v}_{\text{dproj}} &= \texttt{project\_vector\_to\_plane}(\mathbf{v}_d) \end{aligned} \tag{4.28}$$

(4.28) represents the `project_velocities_to_plane` in Fig. 4.15.

4.2.9.3 Projecting Back to 3D-space

Once the collision is detected, we compute the gradient of intersection from the slope of the edge which is not intersecting at an end point. For e.g. for case (i) referring to the collision between **ab** and **c**, the slope of the line **ab** is computed by the difference in x and y directions. In this case, $slope_x = b_x - a_x$ and $slope_y = b_y - a_y$ (see Fig.4.14). The normal of the line $(-slope_y, slope_x)$ is then back projected to find the normal of intersection in 3D as :

$$\mathbf{n}_{\text{intersect}} = (-slope_y * \mathbf{v}_x, slope_x * \mathbf{v}_y, 0) \tag{4.29}$$

(4.29) represents the method `project_plane_to_3dvector` in Fig. 4.15.

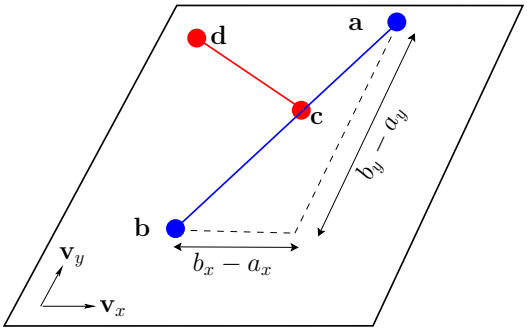


Fig. 4.14: Computing the line gradient for co-planar edges.

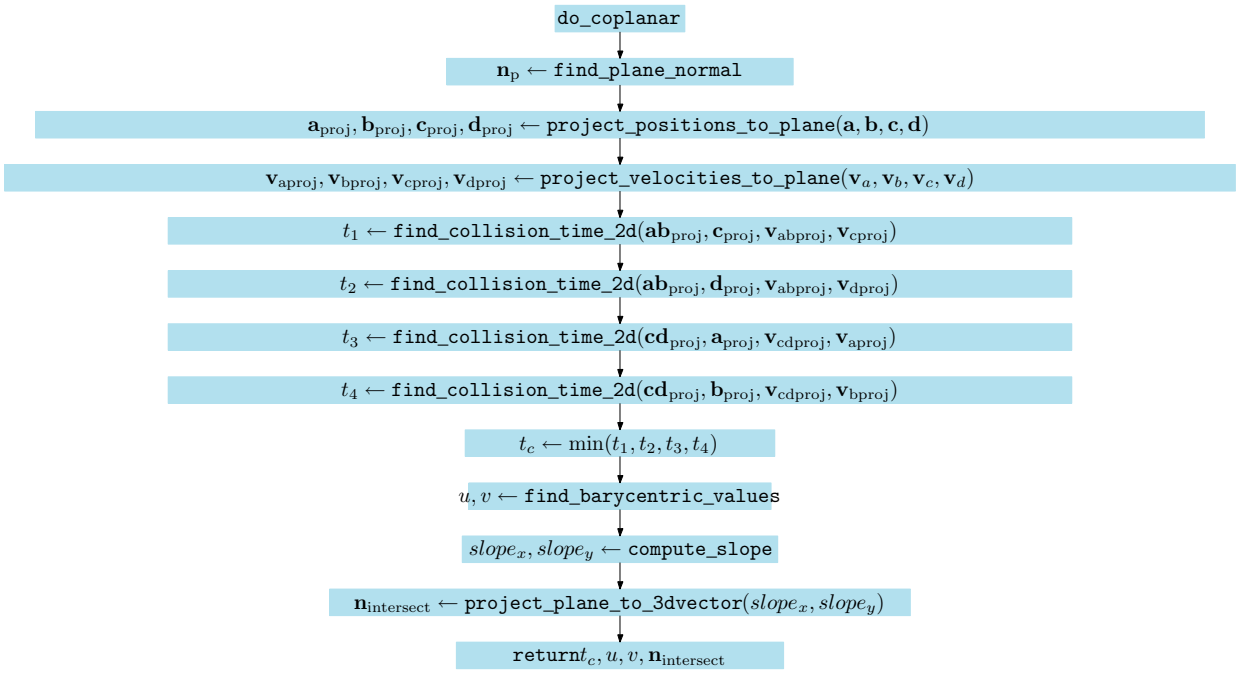


Fig. 4.15: Flowchart illustrating the handling of coplanar edges.

4.2.9.4 Co-planar Results

We now present the results of the continuous collision detection of two moving parallel edges (see Fig. 4.16). An edge with coordinates $\mathbf{a} = (0.5, 1, 0)$ and $\mathbf{b} = (1.5, 1, 0)$ with end-point velocities $\mathbf{v}_a = (0, 0, 0)$ and $\mathbf{v}_d = (0, 0, 0)$ collides with another edge $\mathbf{c} = (0, 1, 0.014)$ and $\mathbf{d} = (1, 1, -0.036)$ with end-point velocities $\mathbf{v}_c = (0, 0, 0.044)$ and $\mathbf{v}_d = (0, 0, 0.044)$. Using the algorithm, we determined a suitable plane normal $\mathbf{n}_p = \mathbf{ab} \times \mathbf{cd} = (0, 1, 0)$ with plane vectors $\mathbf{v}_x = (1, 0, 0)$ and $\mathbf{v}_y = (0, 0, -1)$. After projecting the positions and velocities onto this plane, the time of collision was determined as $t_c = 25ms \in [0, \Delta t]$ (where $\Delta t = 40ms$) and the barycentric coordinates are $u = 0$ and $v = .5$. Then the collision point on the first edge $\mathbf{a} + u(\mathbf{ab} + t_c\mathbf{v}_{ab})$ is $(0.5, 1, 0)$. This is confirmed to be of the same value by computing the collision point on the second edge $\mathbf{c} + v(\mathbf{cd} + t_c\mathbf{v}_{cd})$. The normal of intersection $\mathbf{n}_\mathrm{intersect}$ which is needed for correct collision response was found as $(-0.05, 0, -0.99)$.

(i) (ii)

Fig. 4.16: Continuous collision detection between two moving coplanar edges.

4.2.10 Collinear Edges

The co-planar algorithm will fail if the edges are aligned at the time of collision. The `find_plane_normal` routine (see Fig. 4.12) will return a null vector in all three cases. In such cases, we treat the collisions in one-dimension as a collinear case with the following four possibilities :

- **a** collides with **c** (Fig. 4.17 (i))
- **a** collides with **d** (Fig. 4.17 (ii))
- **b** collides with **d** (Fig. 4.17 (iii))
- **b** collides with **c** (Fig. 4.17 (iv))

4.2.10.1 Compute Line

We start by finding a common line direction $\mathbf{n}_{\text{line}}$ by evaluating one of the possible candidate vectors $\frac{\mathbf{ab}}{\|\mathbf{ab}\|}$, $\frac{\mathbf{ac}}{\|\mathbf{ac}\|}$ or $\frac{\mathbf{ad}}{\|\mathbf{ad}\|}$. This step represents the `find_line_vector` in Fig. 4.18. We then project the positions and velocities onto to this line by first setting :

$$a_{\text{line}} = 0 \tag{4.30}$$

The remaining positions are found by projecting them in the direction of the line with respect to **a** :

$$\begin{aligned} b_{\text{line}} &= \mathbf{n}_{\text{line}} \cdot (\mathbf{b} - \mathbf{a}) \\ c_{\text{line}} &= \mathbf{n}_{\text{line}} \cdot (\mathbf{c} - \mathbf{a}) \\ d_{\text{line}} &= \mathbf{n}_{\text{line}} \cdot (\mathbf{d} - \mathbf{a}) \end{aligned} \tag{4.31}$$

Tab. 4.2: Computation of barycentric coordinates for 1D edges.

Case	Parameters
(i) a_{line} and c_{line}	$u = 0, v = 0, t_1 = \frac{c_{\text{line}} - a_{\text{line}}}{v_{\text{aline}} - v_{\text{cline}}}$
(ii) a_{line} and d_{line}	$u = 0, v = 1, t_2 = \frac{d_{\text{line}} - a_{\text{line}}}{v_{\text{aline}} - v_{\text{dline}}}$
(iii) b_{line} and d_{line}	$u = 1, v = 1, t_3 = \frac{d_{\text{line}} - b_{\text{line}}}{v_{\text{bline}} - v_{\text{dline}}}$
(iv) b_{line} and c_{line}	$u = 1, v = 0, t_4 = \frac{c_{\text{line}} - b_{\text{line}}}{v_{\text{bline}} - v_{\text{cline}}}$

(4.30) and (4.31) represent the method `project_positions_to_line` in Fig. 4.18. The velocities are similarly projected as :

$$\begin{aligned} v_{\text{aline}} &= \mathbf{n}_{\text{line}} \cdot \mathbf{v}_a \\ v_{\text{bline}} &= \mathbf{n}_{\text{line}} \cdot \mathbf{v}_b \\ v_{\text{cline}} &= \mathbf{n}_{\text{line}} \cdot \mathbf{v}_c \\ v_{\text{dline}} &= \mathbf{n}_{\text{line}} \cdot \mathbf{v}_d \end{aligned} \tag{4.32}$$

and (4.32) represents the method `project_velocities_to_line` in Fig. 4.18. Once we have projected the positions and velocities, we check for the following cases of testing (see Fig. 4.17) and determine the time of collision and the barycentric coordinates for each of them as shown in Table 4.2. As we did for the coplanar case, we will have to evaluate all four cases and find the first occurrence of collision, i.e. find the collision time with the smallest value. The valid time t_c is chosen by taking the minimum from up to four possible values, i.e. $t_c = \min\{t_1, t_2, t_3, t_4\}$. Note that in each case the orientation of the line direction $\mathbf{n}_{\text{line}}$ is reversed to respect the relative velocity of the colliding segments. For e.g. for case (i), $\mathbf{n}_{\text{line}} = -\mathbf{n}_{\text{line}}$, if $v_a < v_c$. We similarly alter the orientation for other cases as well. Fig. 4.18 illustrates the overall method of handling the collinear case.

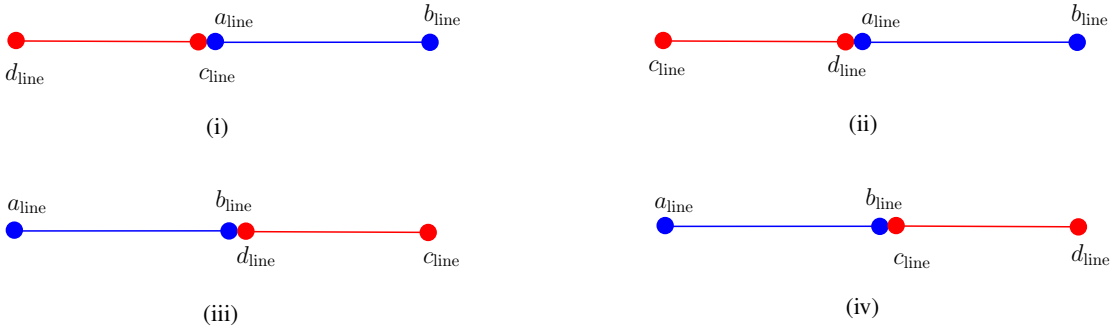


Fig. 4.17: Four possible cases when two edges collides in a line.

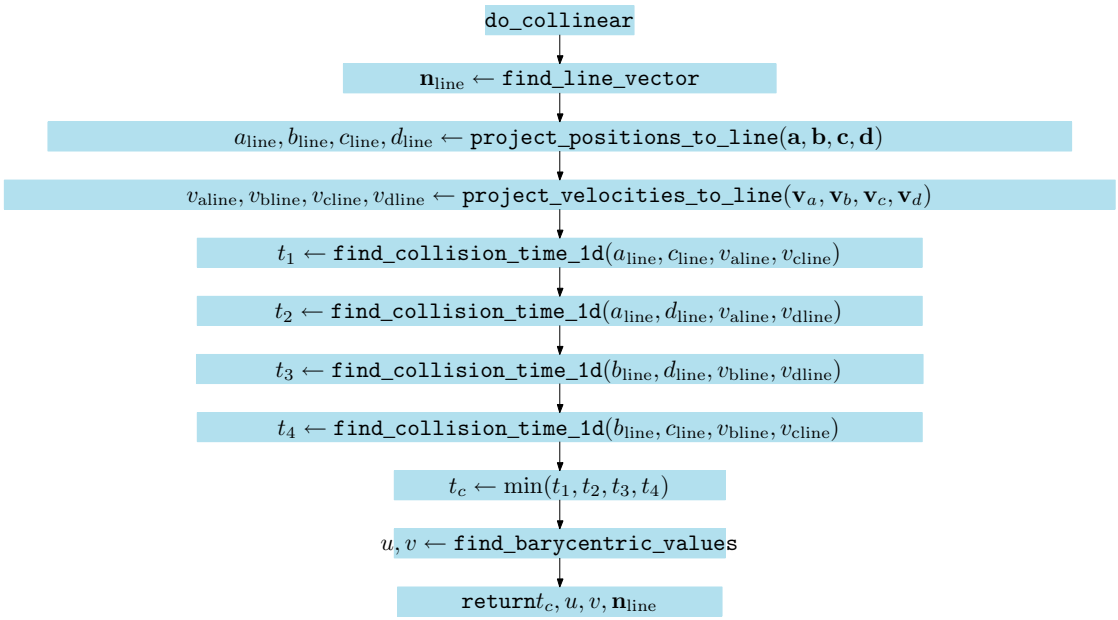


Fig. 4.18: Flowchart illustrating the handling of collinear edges.

Experimental Validation : We now present the results of the continuous collision detection of two moving collinear edges (see Fig. 4.19). An edge with coordinates $\mathbf{a} = (0.5, 1, 0)$ and $\mathbf{b} = (1.5, 1, 0)$ with end-point velocities $\mathbf{v}_a = (0, 0, 0)$ and $\mathbf{v}_d = (0, 0, 0)$ collides with another edge $\mathbf{c} = (1.5184, 1, 0)$ and $\mathbf{d} = (2.5184, 1, 0)$ with end-point velocities $\mathbf{v}_c = (-.48, 0, 0)$ and $\mathbf{v}_d = (-.48, 0, 0)$. Here, the co-planar algorithm will fail since there exists no valid plane normal. Hence using the above algorithm, we determined a suitable line direction $\mathbf{n}_{\text{line}} = (1, 0, 0)$. After projecting the positions and velocities onto this line with $b_{\text{line}} = 1$, $c_{\text{line}} = 1.0184$, $v_{\text{bline}} = 0$ and $v_{\text{cline}} = -.48$, the time of collision was determined as $t_c = 38ms \in [0, \Delta t]$ (where $\Delta t = 40ms$) and the barycentric coordinates are $u = 1$ and $v = 0$ (case of b_{line} colliding with c_{line}). The point of contact was of course verified as $(1.5, 1, 0)$.

(i) (ii)

Fig. 4.19: Continuous collision detection between two moving edges collinear at the time of collision.

4.3 Conclusion

In this chapter, we have comprehensively treated the collision detection occurring between primitives. Here, we have summarized the existing approaches for the sake of completeness. In addition, we have proposed new methods for handling complicated cases such as co-planar and collinear edges. We have demonstrated the validity of these algorithms through quantitative experiments. Our methods are robust and will come handy in the next chapter for treating complex, real-world examples.

CHAPTER 5

Robust Response to Multiple Collisions

In this chapter, we present two new methods which treat the multiple simultaneous collisions occurring in deformable objects. The first is a constrained based approach using quadratic programming techniques which expresses collisions as linear constraints within a global framework. The second is a spring based approach which exploits temporal coherence to guarantee that no collision is left untreated. We analyze these two methods and compare their advantages and weaknesses.

5.1 Introduction

The robust handling of collisions and contacts is important in physics-based animation and simulation scenarios. In chapters 2 and 3, we highlighted the shortcomings of the existing approaches to deal with multiple collisions. In chapter 4 we presented robust methods for collision detection. In this chapter, we first present a new approach which handles dynamics and collision treatment simultaneously. We consider the collisions as linear constraints and the dynamics equation as an objective function to be minimized. We thus get a unified equation modeled as a quadratic programming (QP) problem and solve it using an active set method. We iterate the QP until the solution satisfies all the constraints with the appropriate sign of the Lagrange's multipliers. Thus we get a solution to the dynamics equation which responds to all the collisions. Other constraints such as assigning a constant velocity to a particle, limiting strain/strain rate, etc. too can be easily modeled as linear constraints. We call this

first approach *Quadratic Programming Collide* or *QP-Collide*. In this chapter, we describe in detail how such an approach can be integrated within an existing dynamics simulation environment. In addition, we also include implementation issues of this approach and discuss practical tricks to overcome the same.

Secondly, we present a robust and easy to implement approach which guarantees that no collision will be missed. Our novel spring-based response solves the issues of multiple collisions and strain control by naturally exploiting temporal coherence. A colliding primitive pair coming into contact in the "proximity" region is kept apart by springs inserted by the discrete-time detection routine. By retaining and updating the springs between the primitives over time steps we are able to constantly monitor for new interpenetrations. Any new instantaneous traversals are detected and gradually pushed apart over multiple passes thanks to continuous-time detection. Our approach thus guarantees a "collision-free" final state while minimizing the creation of "secondary collisions" and without inserting additional strain into the system. We call this second approach *Guaranteed Collision Response*.

We present both these methods one after the other before discussing the relative merits and disadvantages of each of them and how they compare with the methods already existing in literature.

5.2 Quadratic Programming Collide

5.2.1 Our Approach

Typical dynamics problems formulated as an ordinary differential equation (ODE) can be solved either by explicit integration techniques such as forward Euler, Runge-Kutta, etc. or implicit techniques like the backward Euler. The latter is especially useful for solving "stiff" set of equations and provides a stable solution even while simulating at large time-steps [BW98]. We make no assumption on the type of integration method for our QP-based approach works with both the methods though we have used the implicit Euler for the advantages mentioned above. Typically we write the implicit Euler equation to solve for $\Delta\mathbf{v}$ as [BW98] :

$$\left(\mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}} - \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right) \Delta\mathbf{v} = \Delta t \left(\mathbf{f} + \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{v}} \mathbf{v}\right) \tag{5.1}$$

We integrate the new velocities $\mathbf{v} + \Delta\mathbf{v}$ to get the new positions $\mathbf{x} + \Delta t(\mathbf{v} + \Delta\mathbf{v})$.

As we noted in chapter 2, the usual way is to perform collision detection at this step and apply the correction instantaneously to the positions and velocities (impulses) or apply stiff spring forces in the next time step (penalty forces). Referring to conclusions we drew in chapter 3 through our case studies, the problems of treating the collisions one by one are principally twofold :

- Treating one set of collisions may provoke new "secondary" collisions
- Correcting a colliding pair in isolation may introduce additional strain into the system

Here, we propose a new method by handling dynamics and collision treatment simultaneously. We consider the collisions as linear constraints and the above dynamics equation as an objective function to be minimized. We thus get a unified global solution to the problem in hand. Let us describe the method in detail. First, we write the implicit equation (5.1) in a concise form :

$$\mathbf{K}\Delta\mathbf{v} = \mathbf{b} \tag{5.2}$$

We then write this equation as a quadratic function with linear constraints. Minimize :

$$q(\Delta\mathbf{v}) \equiv \frac{1}{2}\Delta\mathbf{v}^T\mathbf{K}\Delta\mathbf{v} - \mathbf{b}^T\Delta\mathbf{v} \tag{5.3}$$

subject to the constraints :

$$\mathbf{J}\Delta\mathbf{v} \leq \mathbf{c} \tag{5.4}$$

where $\mathbf{J}$ and $\mathbf{c}$ are the constraint matrix and values respectively (cf. §5.2.3 on how to compute these constraints). If (5.4) is an equality, this is a classical *quadratic programming* (QP) problem which can be solved in a finite number of steps. In practice, we consider the above inequality as an equality by means of an *active set method* described in [Fle87]. Accordingly, those constraints belonging to the active set $\mathcal{A}$ are considered as equalities while the remaining are temporarily ignored. Thus, (5.4) is reduced to :

$$\mathbf{J}_a\Delta\mathbf{v} = \mathbf{c}_a, a \in \mathcal{A} \tag{5.5}$$

From (5.3) and (5.5), we write the Lagrangian function representing the *Karush-Kuhn-Tucker* (KKT) optimality condition as :

$$\mathcal{L}(\Delta\mathbf{v}, \lambda) = \frac{1}{2}\Delta\mathbf{v}^T\mathbf{K}\Delta\mathbf{v} - \mathbf{b}^T\Delta\mathbf{v} - \lambda^T(\mathbf{J}_a\Delta\mathbf{v} - \mathbf{c}_a) \tag{5.6}$$

$$\begin{aligned} \nabla_{\Delta\mathbf{v}}\mathcal{L} = \mathbf{0} \quad &\Rightarrow \quad \mathbf{K}\Delta\mathbf{v} - \mathbf{b} - \mathbf{J}_a^T\lambda = 0 \\ \nabla_{\lambda}\mathcal{L} = \mathbf{0} \quad &\Rightarrow \quad \mathbf{J}_a\Delta\mathbf{v} - \mathbf{c} = 0 \end{aligned} \tag{5.7}$$

which can be rearranged as :

$$\begin{bmatrix} \mathbf{K} & -\mathbf{J}_a^T \\ -\mathbf{J}_a & \mathbf{0} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{v} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{c}_a \end{pmatrix} \tag{5.8}$$

For notational convenience the left hand side of (5.8) is referred to as the $\mathbf{A}$ matrix, i.e.

$$\mathbf{A} = \begin{bmatrix} \mathbf{K} & -\mathbf{J}_a^T \\ -\mathbf{J}_a & \mathbf{0} \end{bmatrix} \tag{5.9}$$

The solution obtained $(\Delta\mathbf{v}, \lambda)$ is verified such that :

$$\lambda_a \geq 0, a \in \mathcal{A} \tag{5.10}$$

$$\mathbf{J}_k \Delta\mathbf{v}_k \leq \mathbf{c}_k, k \notin \mathcal{A} \tag{5.11}$$

At the end of each iteration, if (5.10) is not satisfied, the corresponding constraint $\mathbf{J}_i$ is removed from the active set $\mathcal{A}$ and is put into $\mathcal{A}'$. Viceversa if a non-active constraint in (5.11) is violated, then the same is transferred from $\mathcal{A}'$ to be part of $\mathcal{A}$. The iteration stops when we find a solution with satisfied both these conditions. We describe the algorithm in detail in §5.2.6.

5.2.2 A Note on Earlier Approach

Before we detail on how these constraints can be computed, we would like to draw the attention towards an earlier attempt at using quadratic programming for calculating contact forces. As we described in §2.4.2, Baraff proposed [Bar94] an algorithm for calculating the contact forces between colliding solid objects. We had already briefly discussed as to why this method is very difficult to apply for deformable objects. In order to understand the problem better, let us describe his method in brief and try to expand it to deformable case.

The problem is as follows. Let a_i be the relative acceleration between two bodies. If $a_i > 0$, the bodies are breaking the contact $\mathbf{p}_i$. Conversely if $a_i < 0$ the bodies are accelerating towards each other to interpenetrate. An acceleration $a_i = 0$ means that the bodies are in resting contact. Thus to prevent interpenetration, we require $a_i > 0$ for each contact point. With multiple contact points let the vector $\mathbf{a}$ represent the set of accelerations.

Similarly, a positive force f_i between the two bodies indicate a repulsive force and vice versa. Since frictionless contact forces are conservative we require that $f_i a_i = 0$ for each contact point i. Thus the condition requires that at least one of f_i and a_i be zero for each contact. Let the vector $\mathbf{f}$ represent the set of forces.

The vectors $\mathbf{a}$ and $\mathbf{f}$ are related and can be written as :

$$\mathbf{a} = \mathbf{A}\mathbf{f} + \mathbf{b} \tag{5.12}$$

such that

$$a_i \geq 0, f_i \geq 0 \tag{5.13}$$

and

$$f_i a_i = 0 \tag{5.14}$$

Baraff then presented the above as a quadratic programming problem by minimizing

$$\min \ \mathbf{f}^T(\mathbf{A}\mathbf{f} + \mathbf{b}) \tag{5.15}$$

subject to

$$\mathbf{Af} + \mathbf{b} \geq \mathbf{0} \tag{5.16}$$

and

$$\mathbf{f} \geq \mathbf{0} \tag{5.17}$$

Baraff subsequently went about finding a solution by computing the forces one by one as follows. First, all the contact points except the first are ignored, i.e. $f_i = 0$ for all i. Only f_1 is computed using (5.13) and (5.14) by disregarding the remaining conditions. Then f_2 is computed by ignoring the rest while maintaining the condition for $i = 1$ and so on. Let us take a case where we need to find the force for an arbitrary n^{th} contact. This means that we already know $f_1, f_2, ..., f_{n-1}$. Let there be two sets C and NC where $C = 1, 2, ..., k$ and $NC = k+1, k+2, ..., n-1$.

In order to better appreciate the problem with this approach applied to deformable objects, we now expand on the structure of the $\mathbf{A}$ matrix which was not elaborated in [Bar94]. The force for the n^{th} contact f_n needed to bring the acceleration $a_n = 0$ is computed by solving.

$$\begin{pmatrix} \mathbf{M} & \mathbf{J}_C^T & \mathbf{J}_{NC}^T & \mathbf{j}_n^T \\ \mathbf{J}_C & 0 & 0 & 0 \\ \mathbf{J}_{NC} & 0 & 0 & 0 \\ \mathbf{j}_n & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \gamma \\ \lambda_C \\ \lambda_{NC} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{\text{ext}} \\ 0 \\ \geq 0 \\ \geq 0 \end{pmatrix} \tag{5.18}$$

such that

$$\lambda_{NC} = 0 \tag{5.19}$$

and

$$\lambda_C \geq 0 \tag{5.20}$$

where $\mathbf{M}$ is the inertia matrix of the rigid body system, $\mathbf{f}_{\text{ext}}$ is the sum total of all the external forces, λ_C and λ_{NC} are the lagrangian multiplier values of the sets C and NC respectively. The equality part of the above can be rewritten as :

$$\mathbf{M}\gamma + \mathbf{J}_C^T\lambda_C + \mathbf{j}_n^T = \mathbf{f}_{\text{ext}} \tag{5.21}$$

and

$$\mathbf{J}_C\gamma = 0 \tag{5.22}$$

Thus we need to first solve for λ_C :

$$\left(\mathbf{J}_C\mathbf{M}^{-1}\mathbf{J}_C^T\right)\lambda_C = \mathbf{J}_C\mathbf{M}^{-1}\mathbf{f}_{\text{ext}} - \mathbf{J}_C\mathbf{M}^{-1}\mathbf{j}_n^T \tag{5.23}$$

The accelerations γ are then calculated by :

$$\gamma = \mathbf{M}^{-1}\left(\mathbf{f}_{\text{ext}} - \mathbf{J}_C^T\lambda_C - \mathbf{j}_n^T\right) \tag{5.24}$$

Note that this approach is already computationally heavy since it takes multiple QP-passes to calculate the force of each contact point one by one. Further the equivalent of the $\mathbf{M}^{-1}$ matrix for deformable objects will be $\left(\mathbf{M} - (\Delta t)^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}} - (\Delta t) \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right)^{-1}$ as in (5.1). Thus for each pass, in order to solve (5.23) directly, we should find the inverse of $\mathbf{J}_C \mathbf{M}^{-1} \mathbf{J}_C^T$ which is very difficult to compute. We cannot precalculate this since the matrix changes *every* time step as the object deforms. The other option is to formulate the problem as an iterative solution. With this approach, we may not need an explicit matrix representation (cf. §3.3.1). But this approximation will further reduce the accuracy of the solution leading to increased numerical precision errors. Thus this method is not applicable for the problem at hand.

5.2.3 On Modeling the Constraints

We now describe how the QP-collide method introduced in §5.2 can be used to model various constraints starting with the collisions. Though the focus of this research is the treatment of collisions, this approach enables us to take account of several other issues we often encounter in dynamics simulation such as fixing a point, controlling the strain, etc.

5.2.3.1 Collision Constraints

We refer to the collision detection techniques described in chapters 3 and 4. Recall that by solving (5.8) for $\Delta\mathbf{v}$, we approach the problem of collision response solely through velocity impulse corrections on the lines of [BFA02]. We do not modify positions explicitly thus avoid creating new collisions and also prevent introducing additional mechanical strain into the system. We now describe how to deal with the collision primitives (vertex-triangle and edge-edge) once they are detected (described in detail in chapter 4).

5.2.3.2 Vertex-Triangle Collision Constraint

This constraint can be applied to both discrete and continuous situations described in §4.2.1, §4.2.2 and §4.2.3. Let a vertex $\mathbf{x}_i$ with thickness r_i moving at velocity $\mathbf{v}_i$ collide with a triangle $(\mathbf{x}_{j0}, \mathbf{x}_{j1}, \mathbf{x}_{x2})$ with thickness r_j moving at $(\mathbf{v}_{j0}, \mathbf{v}_{j1}, \mathbf{v}_{j2})$ and normal $\hat{\mathbf{n}}$ (see Fig. 5.1). The above are the instantaneous values at time t. Our objective is to find the new $\Delta\mathbf{v}$ at

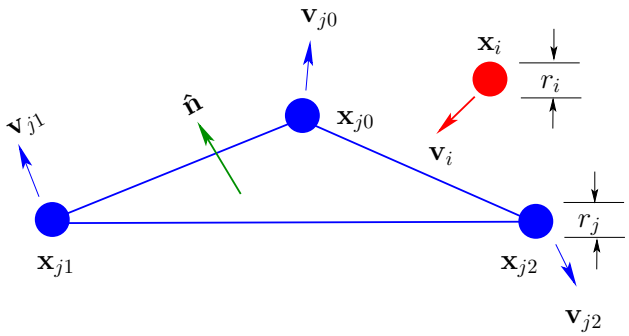


Fig. 5.1: Modeling the vertex-triangle collision constraint.

time $t + \Delta t$. Assuming the barycentric coordinates are (w_0, w_1, w_2) such that velocity at the colliding point is $\mathbf{v}_j = w_0\mathbf{v}_{j0} + w_1\mathbf{v}_{j1} + w_2\mathbf{v}_{j2}$. Let g be the gap $\parallel \mathbf{x}_i - \mathbf{x}_j \parallel$ such that distance which the vertex and triangle needs to be separated into a non-interfering state is $g-(r_i+r_j)$. Now the equation constraining the new relative velocity change $\Delta\mathbf{v}_{ij} = \Delta\mathbf{v}_j - \Delta\mathbf{v}_i$ can be written as :

$$(\mathbf{v}_{ij} + \Delta\mathbf{v}_{ij}) \cdot \hat{\mathbf{n}} \ \ \Delta t \leq g - (r_i + r_j) \tag{5.25}$$

It can be rearranged as :

$$(w_0\Delta\mathbf{v}_{j0} + w_1\Delta\mathbf{v}_{j1} + w_2\Delta\mathbf{v}_{j2} - \Delta\mathbf{v}_i).\hat{\mathbf{n}} \leq c \tag{5.26}$$

where $c = (g-(r_i+r_j))/\Delta t+(w_0\mathbf{v}_{j0}+w_1\mathbf{v}_{j1}+w_2\mathbf{v}_{j2}-\mathbf{v}_i)\cdot\hat{\mathbf{n}}$. Each such constraint corresponds to one row of the $\mathbf{J}$ matrix of (5.4). The structure of the k^{th} row $\mathbf{J}_k$ will then look as follows :

$$\begin{pmatrix} \cdots & j_0 & j_1 & j_2 & \cdots & i & \cdots \\ \cdots\ 0 & w_0\hat{\mathbf{n}} & w_1\hat{\mathbf{n}} & w_2\hat{\mathbf{n}} & \cdots\ 0\ \cdots & \hat{\mathbf{n}} & 0\ \cdots \end{pmatrix} \tag{5.27}$$

We present an example result produced by implementing the vertex-triangle collision constraint in Fig. 5.2 where a falling point is prevented from interpenetrating a triangle.

5.2.3.3 Edge-Edge Collision Constraint

Now, let an edge with end-points $(\mathbf{x}_{i0}, \mathbf{x}_{i1})$, thickness r_i and velocities $(\mathbf{v}_{i0}, \mathbf{v}_{i1})$ collide with another edge with end-points $(\mathbf{x}_{j0}, \mathbf{x}_{j1})$, thickness r_j and velocities $(\mathbf{v}_{j0}, \mathbf{v}_{j1})$ (again, all values at time t) (see Fig. 5.3). The barycentric coordinates at the collision point is $[a, b]$ such that the velocities at the colliding point equals $\mathbf{v}_i = (1-a)\mathbf{v}_{i0}+a\mathbf{v}_{i1}$ and $\mathbf{v}_j = (1-b)\mathbf{v}_{j0}+b\mathbf{v}_{i1}$. The equation constraining the relative velocity is similar to (5.25). It can be further elaborated as :

$$((1-b)\Delta\mathbf{v}_{j0} + b\Delta\mathbf{v}_{j1} - (1-a)\Delta\mathbf{v}_{i0} - a\Delta\mathbf{v}_{i1}) \cdot \hat{\mathbf{n}} \leq c \tag{5.28}$$

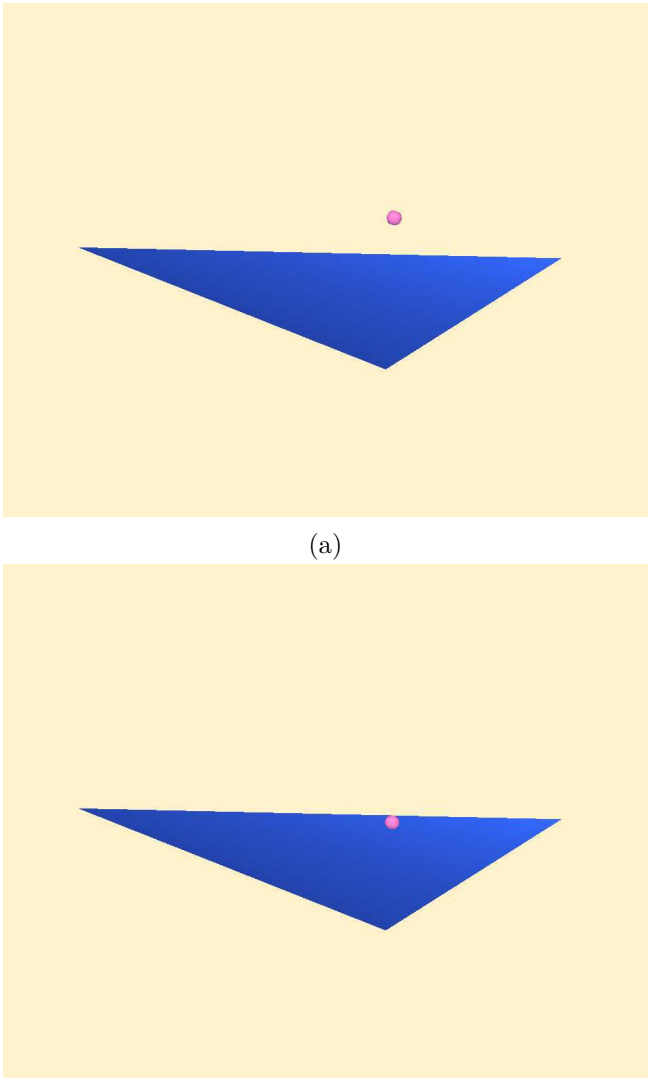
where $\hat{\mathbf{n}}$ is the normalized direction of the gap $(\mathbf{x}_i - \mathbf{x}_j)/ \parallel \mathbf{x}_i - \mathbf{x}_j \parallel$ and $c = (g - (r_i + r_j))/\Delta t + \mathbf{v}_{ij} \cdot \hat{\mathbf{n}}$. Similar to the vertex-triangle case, the structure of the k^{th} row $\mathbf{J}_k$ will then look as follows :

$$\begin{pmatrix} \cdots & j_0 & j_1 & \cdot & \cdot & \cdot & i_0 & i_1 & \cdots \\ 0 & (1-b)\hat{\mathbf{n}} & b\hat{\mathbf{n}} & \cdots & 0 & \cdots & -(1-a)\hat{\mathbf{n}} & -a\hat{\mathbf{n}} & 0\cdots \end{pmatrix} \tag{5.29}$$

We present an example result produced by implementing the edge-edge collision constraint in Fig. 5.4.

5.2.3.4 Fixed Constraints

Fixed constraints are those which can be used to assign a constant velocity value. Once the constraint directions and values are set, it is considered as part of the active set throughout



(a)

(b)

Fig. 5.2: Results from the simulation of a vertex-triangle collision constraint.

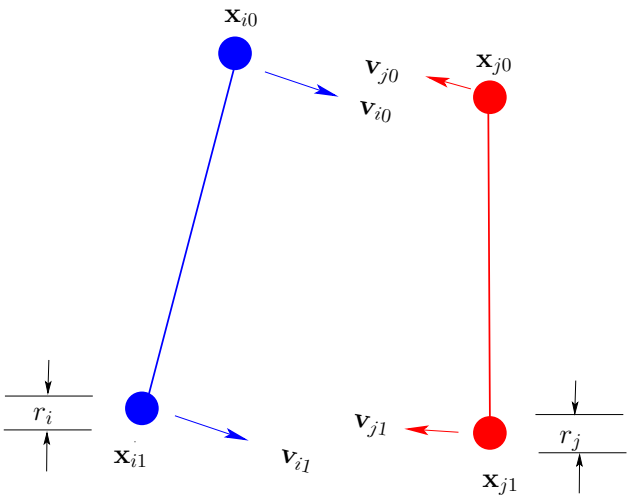
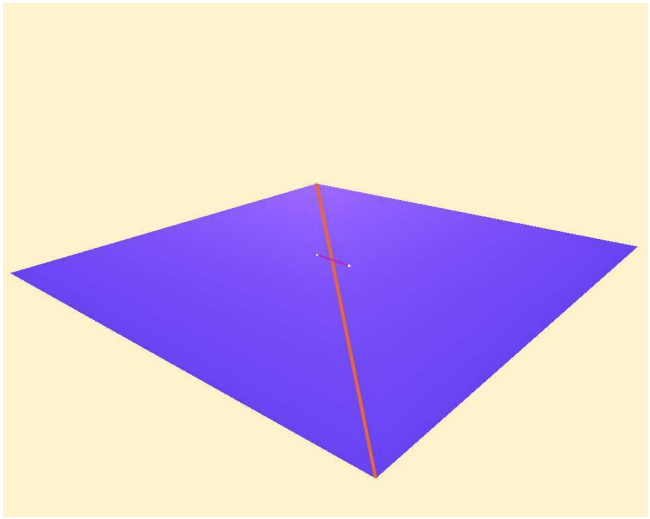
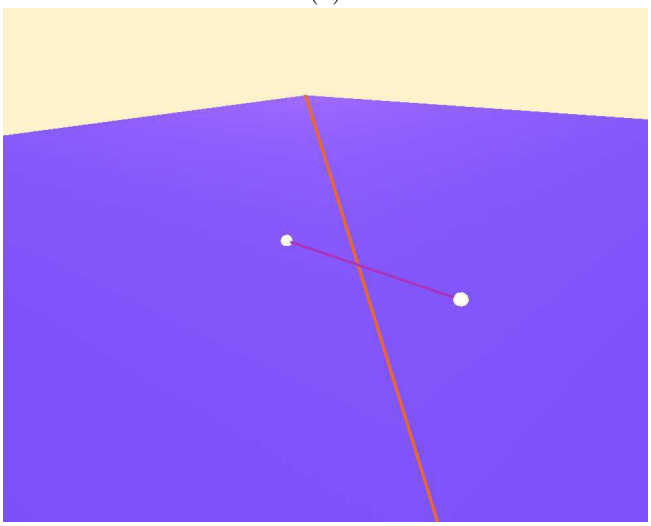


Fig. 5.3: Modeling the edge-edge collision constraint.



(a)



(b)

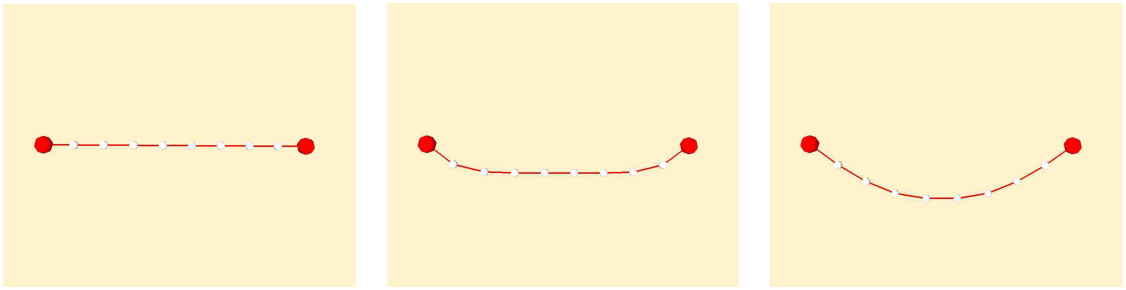
Fig. 5.4: Results from the simulation of an edge-edge collision constraint.

the iteration and is thus treated as an equality all along. Fig. 5.5 illustrates the dynamics of a connected particle system with its two end points fixed using our constraint method.

5.2.3.5 Strain Constraints

Some times we would like to control the strain values during the simulation typically to avoid excessive elongation or compression of spring (cf. §3.3.5). Our approach provides a straightforward means to implement this. We define the strain of a mechanical element (say, a spring) as the change in its current length l relative to its rest length l_0, i.e. $(l - l_0)/l_0$. Let $l_{\max}$ be the maximum strain allowed (say 10% over l_0). For a spring connected by two end-points (i, j) (see Fig. 5.6), this can be written as :

$$\| \mathbf{x}_i + \Delta t(\mathbf{v}_i + \Delta \mathbf{v}_i) \ - \mathbf{x}_j - \Delta t(\mathbf{v}_j + \Delta \mathbf{v}_j) \| < l_{\max} \tag{5.30}$$



(i) (ii) (iii)

Fig. 5.5: A particle system with the two ends fixed using constraints.

By ignoring the $(\Delta t)^2$ terms, we get :

$$\sqrt{2\Delta t(\Delta \mathbf{v}_i - \Delta \mathbf{v}_j) \cdot \hat{\mathbf{n}} l + l^2} < l_{\max} \tag{5.31}$$

where $l = \| \mathbf{x}_i + \Delta t \mathbf{v}_i - \mathbf{x}_j - \Delta t \mathbf{v}_j \|$ and $\hat{\mathbf{n}}$ is the normalized direction of the spring elongation $(\mathbf{x}_i + \Delta t \mathbf{v}_i - \mathbf{x}_j - \Delta t \mathbf{v}_j)/l$. Using a first order approximation :

$$\sqrt{1+\epsilon} = 1 + \frac{1}{2}\epsilon + O(\epsilon^2) \tag{5.32}$$

we can linearize (5.31) as :

$$(\Delta \mathbf{v}_i - \Delta \mathbf{v}_j) \cdot \hat{\mathbf{n}} \leq l(l_{\max} - l)/\Delta t \tag{5.33}$$

A similar equation can be written so that the spring does not compress below a certain length $l_{\min}$. Note that unlike the explicit strain control methods proposed in [BFA02], our approach can incorporate this as another set of linear constraints. While their approach is prone to suffer from unending loops with sequential methods such as Gauss-Seidel approach, our approach ensures that the solution satisfies the user-defined strain control constraints. We present an example result produced by implementing the strain control constraint in Fig. 5.7. Here, we notice that the simulations without strain control (see Fig. 5.7(i)) suffers from a large deformation (upto 250% of the the initial length) without any strain control measures. In contrast, we applied the above constraint (with $l_{\max}$ set at 10% of the initial length) to our QP solver, and we observed that the solution perfectly respected this constraint. Fig. 5.7(ii) illustrates that the suspended cable does not elongate more than 10% under the exact same physical conditions.

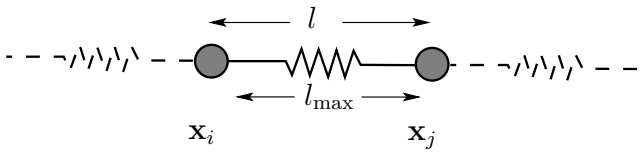
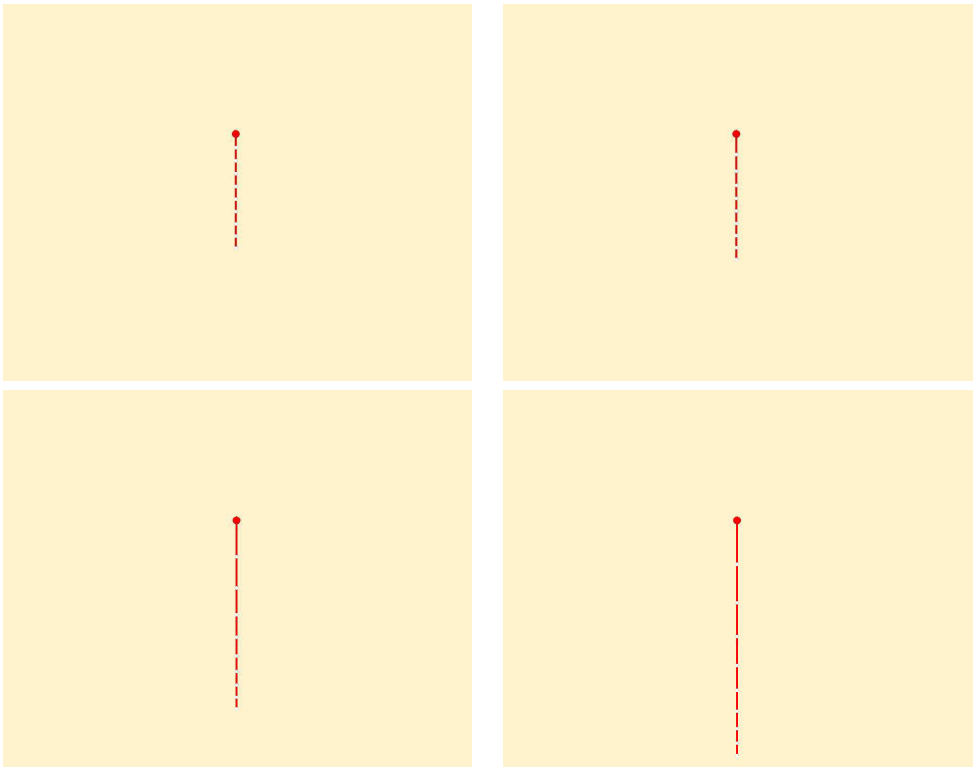
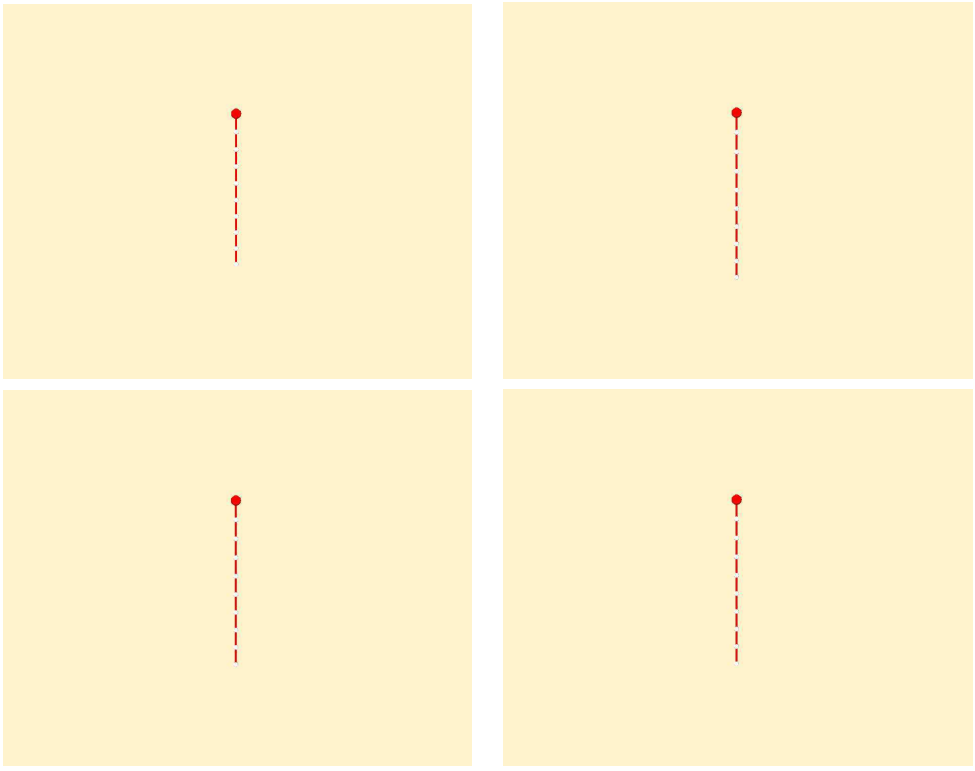


Fig. 5.6: Modeling the strain constraint.



(i) A cable suspended from the top elongating *without* strain control.



(ii) A cable suspended from the top elongating *with* strain control.

Fig. 5.7: Results from the simulation of strain control constraint.

5.2.4 On Using the Conjugate Gradient Algorithm

5.2.4.1 Without Inverting K

We generally assume that the **K** matrix of (5.3) is symmetric and positive-definite. Hence the composite matrix **A** of (5.9) is also symmetric and can be solved with a standard *Conjugate Gradient* (CG) algorithm [PFTV92]. Note that sometimes due to numerical errors, the symmetric form of (5.8) can be ill-conditioned due to negative eigenvalues. In such cases we found that the non-symmetric version gave an acceptable solution. Of course since it is no longer symmetric we will have to use a *Bi-Conjugate Gradient* algorithm which takes twice the calculations than a standard CG algorithm. Accordingly, the modified form is :

$$\begin{bmatrix} \mathbf{K} & -\mathbf{J}_a^T \\ \mathbf{J}_a & \mathbf{0} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{v} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{c}_a \end{pmatrix} \tag{5.34}$$

We would also prefer **A** is positive semi-definite (PSD) so that we get the global solution to the equation. However there are cases where the matrix is not full rank due to linearly dependant constraints. We shall deal with such cases in detail in §5.2.7.1. Otherwise, both the **K** and **J** matrices tend to be sparse and can be solved by iterative methods such as the conjugate gradient algorithm in linear time complexity. Hence it would make sense to create methods which perform the product of the matrix vector $\mathbf{y} = \mathbf{A} \cdot \mathbf{x}$, where $\mathbf{x}$ consists of the vector triplet $\Delta\mathbf{v}$ and the Lagrangian multiplier λ. This avoids the need for having an explicit representation of the **A** matrix. Assuming that there are n equations and m active constraints, the multiplication operators required for computing the left hand side of (5.8) are :

$$\mathbf{y}_i = \mathbf{K} \cdot \Delta\mathbf{v} - \mathbf{J}_a^T \cdot \lambda, i \in [1, \cdots, n] \tag{5.35}$$

$$\mathbf{y}_i = -\mathbf{J}_a \cdot \Delta\mathbf{v}, i \in [n+1, \cdots, n+m] \tag{5.36}$$

For n particles and m constraints, the conjugate gradient should generally converge within $3n + m$ iterations.

5.2.4.2 Inverting K

In certain cases, it is possible to quickly invert **K** matrix using techniques such as LU decomposition (cf. §3.3.1 for more discussion on this). When $\mathbf{K}^{-1}$ exists, from (5.8) we find the solution λ and $\Delta\mathbf{v}$ as follows :

$$\mathbf{J}\mathbf{K}^{-1}\mathbf{J}^T\lambda = \mathbf{c} - \mathbf{J}\mathbf{K}^{-1}\mathbf{b} \tag{5.37}$$

$$\Delta\mathbf{v} = \mathbf{K}^{-1}(\mathbf{b} + \mathbf{J}^T\lambda) \tag{5.38}$$

(5.37) reduces the maximum CG iterations to m.

5.2.5 Condition for Convergence

After having presented the details of the solver, we will now elaborate on how to terminate the QP iteration. Recall that the solver provides the set of velocity corrections $\Delta\mathbf{v}$ and the Lagrangian values λ_a of the active constraint set. We exit the iteration if the solution satisfies both the conditions (5.10) and (5.11). Otherwise we have to perform the constraint swapping procedure and continue with the iteration as follows. For each active constraint, if the solution does not satisfy (5.10), i.e., $\lambda_q < 0, q \in \mathcal{A}$, the constraint is moved to the non-active set $\mathcal{A}'$ and thus is not considered during the next iteration. Likewise for each non-active constraint, if the solution does not satisfy (5.11), i.e., $\mathbf{J}_p\Delta\mathbf{v}_p > c_p, p \in \mathcal{A}'$, the constraint is moved to the active set $\mathcal{A}$. But sometimes due to numerical errors we may not get the required solution even after several iterations. Hence, we set an upper limit `maxIter` to the number of QP iterations that can be performed. Note that the case we use an iterative solver such as the conjugate gradient, the solution obtained also depends on an user-set tolerance value. The value of the residue at the end of each CG step should be less than this tolerance value. Thus we need to judiciously take into account of these nuances while dealing with convergence issues. We detail some of the techniques we developed to alleviate this problem in §5.2.7.

5.2.6 Overall Algorithm

Our algorithm is flexible to work with any suitable integration technique and collision detection method, this making it easier to fit in with an existing dynamics simulation architecture. The overall algorithm is described in Algorithm 2 :

5.2.7 Practical Difficulties

Throughout this method, we have tried to eliminate as many magic parameters as possible. All the parameters of our system such as stiffness, thickness, etc. are user-given physical parameters. But certain situations during simulations cause numerical errors while computing the solution to the conjugate gradient algorithm. We propose the following techniques to tackle against commonly occurring numerical problems.

5.2.7.1 Linearly Dependant Constraints

There can be some cases where there are linearly dependant rows in the $\mathbf{J}$ matrix. A simple illustration of this situation is as follows. Let us imagine the case of an edge colliding with a fixed plane where each end-points of the edge collides with two adjacent triangles of the plane and the edge itself collides with the edge of the plane shared by the two triangles (see Fig. 5.8). Note that we have three collision constraints in this case - two vertex-triangle constraint and one edge-edge constraint. If the direction of the normal $\hat{\mathbf{n}}$ is (n_x, n_y, n_z) and the barycentric coordinates of the colliding edge is $[a_1, a_2]$ where $a_2 = 1 - a_1$, then the

Algorithm 2 Quadratic Programming Collide

1: Solve (5.1) for $\Delta\mathbf{v}$
2: Find constraints (detect collisions) and populate the constraint matrix $\mathbf{J}$ and values $\mathbf{c}$
3: **for all** constraint $\mathbf{J}_i \in \mathbf{J}[1 \cdots m]$ **do**
4: **if** $\mathbf{J}_i \Delta\mathbf{v}_i > \mathbf{c}_i$ **then**
5: Add i^{th} constraint to $\mathcal{A}$
6: **else**
7: Add i^{th} constraint to $\mathcal{A}'$
8: **end if**
9: **end for**
10: `maxIter` $\leftarrow 3n + m$
11: $k \leftarrow 0$
12: endloop$\leftarrow$ `true`
13: Compute the new $\Delta\mathbf{v}^{(k)}$ and the Lagrange multipliers $\lambda^{(k)}$ by solving (5.8)
14: **for all** $\lambda_q^{(k)}$, $q \in \mathcal{A}$ **do**
15: **if** $\lambda_q^{(k)} < 0$ **then**
16: Move q from $\mathcal{A}$ to $\mathcal{A}'$ (active to non-active)
17: endloop $\leftarrow$ `false`
18: **end if**
19: **end for**
20: **for all** $\Delta\mathbf{v}_p^{(k)}$, $p \in \mathcal{A}'$ **do**
21: **if** $\mathbf{J}_p \Delta\mathbf{v}_p^{(k)} > c_p$ **then**
22: Move p from $\mathcal{A}'$ to $\mathcal{A}$ (non-active to active)
23: endloop $\leftarrow$ `false`
24: **end if**
25: **end for**
26: **if** endloop is `false` and $k <$ `maxIter` **then**
27: $k \leftarrow k + 1$
28: Goto step (2)
29: **else**
30: $\Delta\mathbf{v}^* = \Delta\mathbf{v}^{(k)}$
31: Quit loop
32: **end if**
33: Compute final velocity $\mathbf{v}(t + \Delta t) \leftarrow \mathbf{v}(t) + \Delta\mathbf{v}^*$
34: Compute final position $\mathbf{x}(t + \Delta t) \leftarrow \mathbf{x}(t) + \Delta t \cdot \mathbf{v}(t + \Delta t)$

corresponding rows of $\mathbf{J}$ will be :

$$\begin{pmatrix} \cdots & 0 & n_x & n_y & n_z & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & n_x & n_y & n_z & 0 & \cdots \\ \cdots & 0 & a_1 n_x & a_1 n_y & a_1 n_z & a_2 n_x & a_2 n_y & a_2 n_z & 0 & \cdots \end{pmatrix} \tag{5.39}$$

This is a common occurrence in mechanical simulations and yet it results in a singular $\mathbf{A}$ matrix. The applied mathematics community typically use sequential quadratic programming approaches to deal with such problems [GMS02]. Unfortunately, such methods are very

expensive to compute in real-time applications using standard workstations. Instead we introduce a small perturbation in $\mathbf{J}$ in order to reduce the conditioning number of the matrix. Accordingly, we fill up the bottom right part of $\mathbf{A}$ of (5.6) with a "perturbation" matrix $\mathbf{L} = (l_1, \cdots, l_m)$ with each l_i's value around 10^{-3}. Thus the new $\mathbf{A}$ will look like :

$$\mathbf{A} = \begin{bmatrix} \mathbf{K} & -\mathbf{J}_a^T \\ -\mathbf{J}_a & \mathbf{L} \end{bmatrix} \tag{5.40}$$

The corresponding matrix-vector multiplication routine in (5.36) is appropriately modified as :

$$\mathbf{y}_i = -\mathbf{J} \cdot \Delta\mathbf{v} - \mathbf{L} \cdot \lambda, i \in [n+1, \cdots, n+m] \tag{5.41}$$

This not only reduces the numerical errors but is also extremely easy to compute. The cost of such calculation is m multiplications.

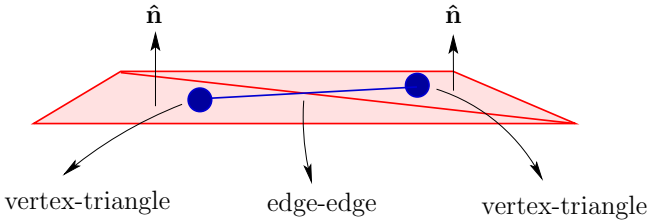


Fig. 5.8: A commonly occurring collision scenario which results in linearly dependant constraints.

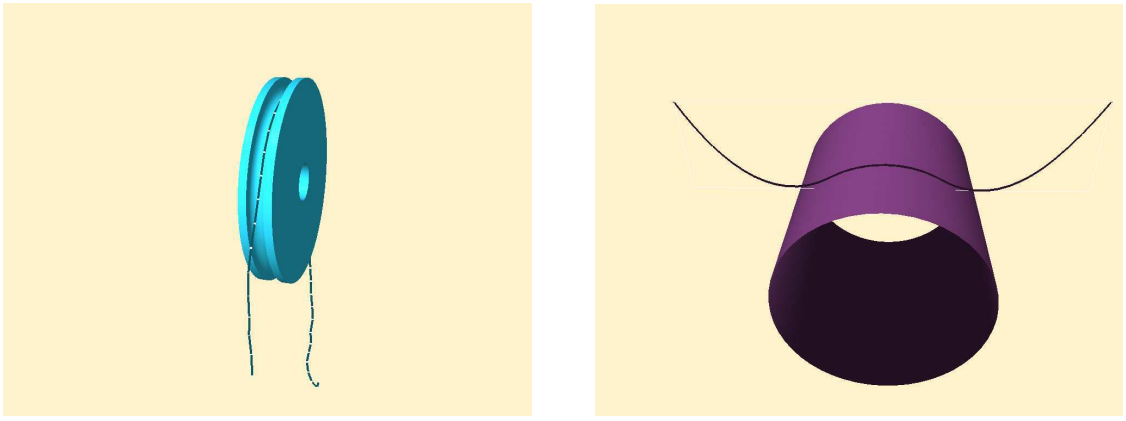
5.2.7.2 Suppressing Toggling Constraints

Other numerical errors results while trying to simulate very stiff objects. This sometimes causes the constraint to toggle between the active set $\mathcal{A}$ and the non-active set $\mathcal{A}'$ resulting in an endless QP iteration loop. Hence such cases should be put in a watch list and in case of repeated toggling, the constraint should be permanently removed from $\mathcal{A}$ and $\mathcal{A}'$. While this may not be theoretically justified, it nonetheless gives satisfactory results in our experiments simulating a mechanical cable with large stiffness values ($> 10^5$ N/m).

5.2.8 Results and Summary

Performance : We have presented an elegant and novel solution for simultaneously treating multiple collisions and contacts. We now present the results of our algorithm. Fig. 5.9(i) shows the case of a free falling cable sliding over a pulley. Fig. 5.9(ii) shows the simulation of a rigid cable fixed at its end-point falling over a rigid cylinder. We note that the algorithm handles both the collisions and fixed constraints well for the case of a mechanical cable.

Fig. 5.10 shows the simulation of cable fixed at two ends falling simultaneously over two fixed cylinders. We present the simulation parameters in Table 5.1. The simulation ran at 30-35 Hz on a 3GHz Pentium ® 4 PC with 1 GB RAM and GeForce ® 3 graphics card. Fig. 5.11 plots the time and the number of iterations required for the simulation presented in Fig.



(i) (ii)

Fig. 5.9: Snapshots of simulation using QP-Collide. A stiff cable (i) falling off a pulley and (ii) suspended (fixed at the end points) over a cylinder.

Tab. 5.1: QP-Collide results : Parameters used in the simulation shown in Fig. 5.10.

Parameter	Value
Number of cylinder triangles	600
Number of cable particles	80
Mass	1.0 kg
Stiffness	$10^4 Nm^{-1}$
Gravity	$-10ms^{-2}$

5.10. Fig. 5.11 (i) shows the simulation time (in ms) for the QP-solver. Note that this time excludes the time required for collision detection. Fig. 5.11 (ii) shows the total number of conjugate gradient (CG) iterations necessary for the algorithm to converge (cf. §5.2.5). Here note that there might be multiple QP iterations with each requiring a certain number of CG iterations. Hence we added up the CG iterations for each QP iteration and presented the total. The pattern of these two graphs are similar showing that the time needed is proportional to the number of CG iterations solved. Fig. 5.11 (iii) shows the total number of variables and constraints solved during each time step. Note that we have at the minimum 80 particles and 6 fixed constraints (three for fixing each end which are active all the time). Fig. 5.11 (iv) shows only the number of constraints solved during each time step. We performed a similar experiment using the same parameters as Table 5.1 except reducing the stiffness value of the cable to $10^3 Nm^{-1}$. We present the plots arising out of the simulation in Fig. 5.12. We note that the number of collision constraints for each case is similar (compare 5.11 (iv) and 5.12 (iv)). But the average time required for the simulation (see Fig. 5.12 (i)) is considerably lower than the corresponding high stiffness case (see Fig. 5.11 (i)).

Drawbacks :There are some drawbacks of this method compared with a more classical approach like penalty springs. The addition of the constraint matrices $\mathbf{J}$ and $\mathbf{J}^T$ somewhat degrades the conditioning of the matrix. Hence the solution given by the QP is susceptible to

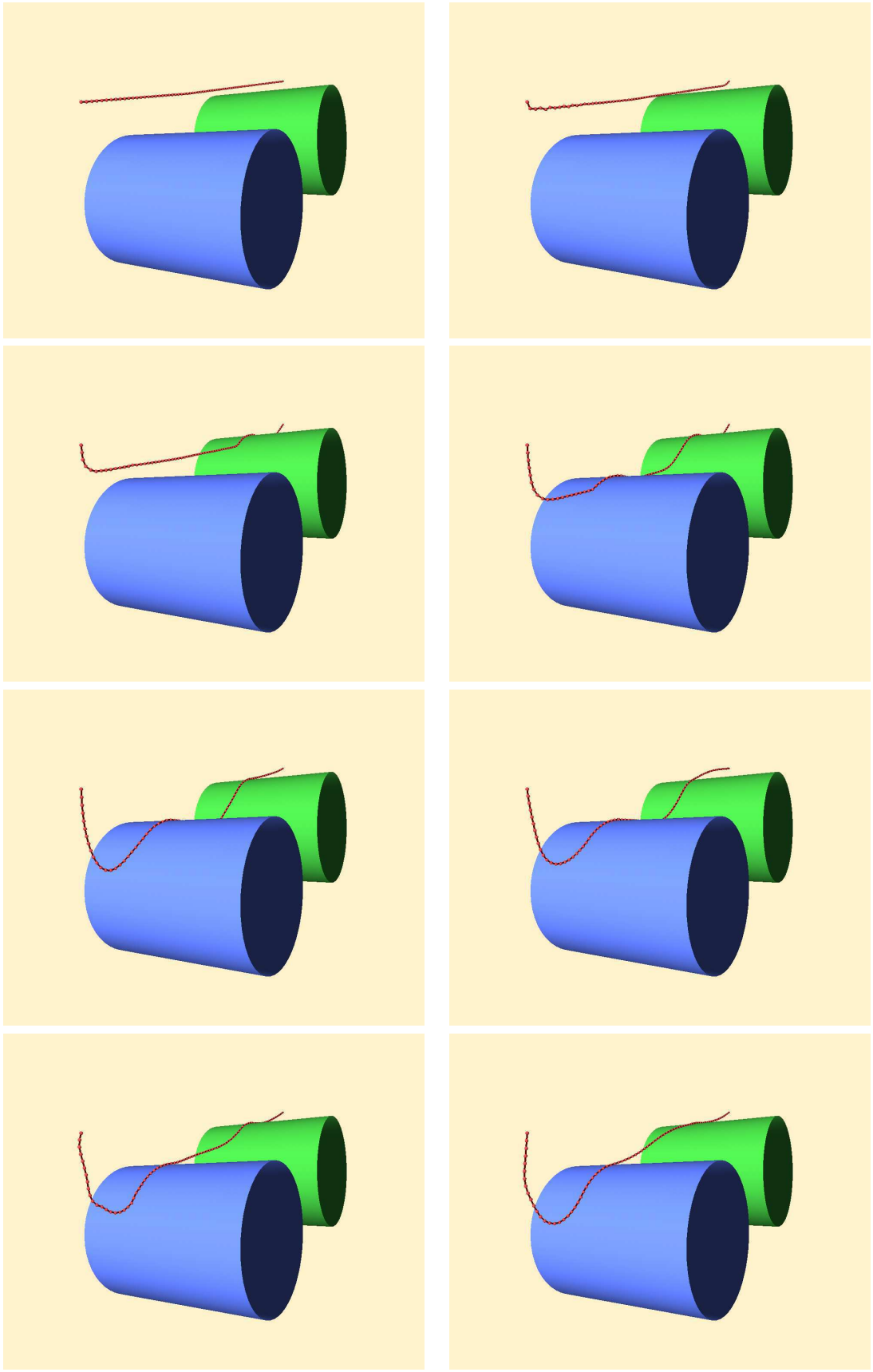


Fig. 5.10: Snapshots of the simulation of a stiff cable with 80 mass particles fixed at two ends falling over two cylinders creating multiple collisions and contacts which are handled using the QP-collide collision response method.

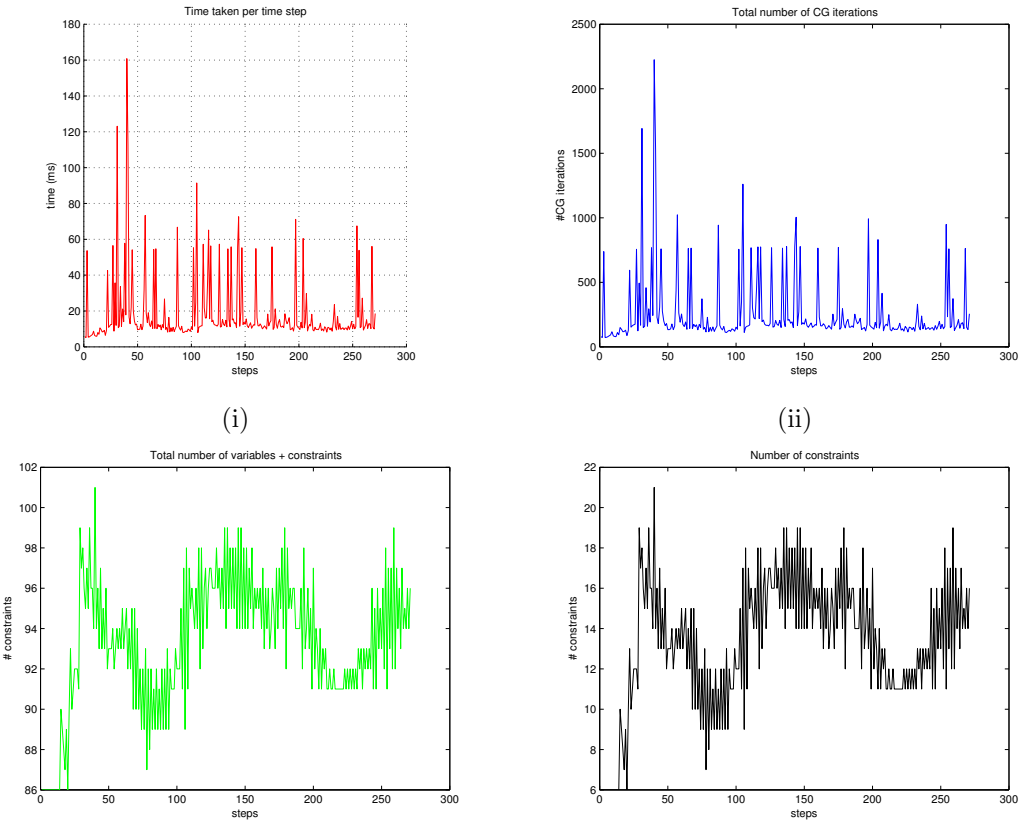


Fig. 5.11: Plot of timings and iterations taken from the simulation shown in Fig. 5.10. (i) Time taken to compute the solution per time step. (ii) Total number of conjugate gradient iterations performed by the QP solver. (iii) Total number of variables + constraints. (iv) Number of constraints.

numerical errors. Though we have proposed some tricks to take care of them, they may not work for all the scenarios. Hence more theoretical work is needed to analyze such problems. In addition, our algorithm may not handle the case of multiple collisions such as a cloth colliding with itself due to the following reasons. Here, we need to perform multiple collision detection passes in order to find new collisions which by itself is not a problem with our robust methods presented in chapter 4. But the problem is that we need go through multiple QP passes to respond to these new collisions which is computationally expensive. In effect this might be the equivalent of a sequential quadratic programming on the lines of Baraff described in §5.2.2. Nonetheless, we hope that this method be further explored to solve the problem in hand.

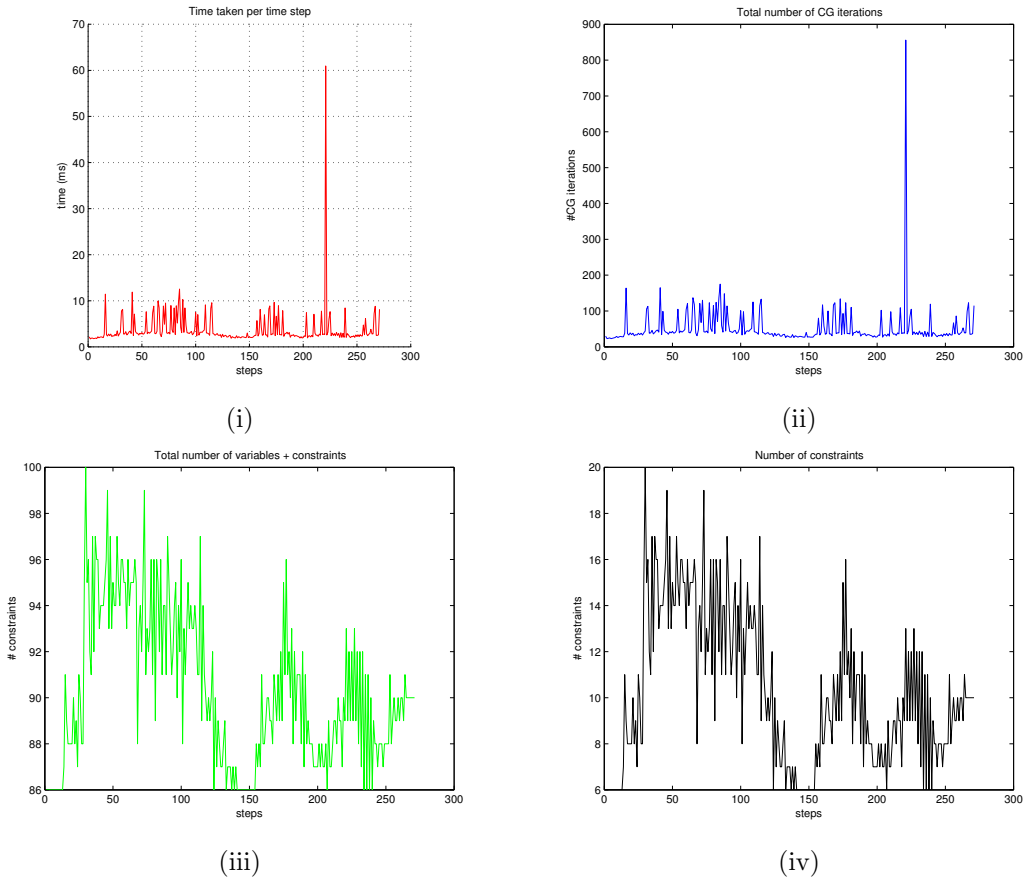


Fig. 5.12: Plot of timings and iterations taken for the simulation shown in Fig. 5.10. (i) Time taken to compute the solution per time step. (ii) Total number of conjugate gradient iterations performed by the QP solver. (iii) Total number of variables + constraints. (iv) Number of constraints.

5.3 Guaranteed Collision Response Method

5.3.1 Motivation

In this section, we present a second method for treating multiple collisions and contacts. In contrast to the quadratic programming approach where we responded to collisions with velocity impulses, here we use a spring based approach. One immediate advantage is that the conditioning of the matrix is not deteriorated as we saw earlier with the QP case. With springs, we just add more entries into the $\mathbf{K}$ matrix and it largely remains positive semi definite. However, we may be introducing "stiff" springs into the system - hence this requires the use of implicit solvers such as the backward Euler.

5.3.2 Overview

In this approach too, we prefer to use the implicit form of the dynamics equation (5.1). As again, we detect collisions between primitives (vertex-triangle, edge-edge, etc.) both at the discrete state $\mathbf{x}(t_0)$ and the sweep interval $\mathbf{x}(t_0)$ and $\mathbf{x}(t_0 + \Delta t)$ using techniques described

in chapter 4. However, we significantly differ in the way we compute and apply the response. In particular, we :

- add a discrete collision spring for those contacts in the "proximity" area
- add a continuous collision spring for those collisions which occur *during* the time step
- exploit temporal coherence which continues to apply spring forces due the above two
- detect new "secondary" collisions within a loop and make sure all of them are treated

5.3.3 Types of Springs

We first present how these springs can be incorporated for the primitive cases of vertex-triangle and edge-edge collisions and then present the overall algorithm in the context of a deformable body dynamics simulation. We maintain a separate list of collision springs for each type of collision, (described in chapter 4). This being a hybrid method, we deal with both discrete and continuous collisions. In particular we have to account for the following cases :

- *Vertex-Triangle Spring* is used to treat the collision between a moving vertex and a moving triangle (which also handles self-collisions)
- *Vertex Spring* is used to treat the collision between a moving point (belonging to a deformable object) colliding and a fixed triangle (belonging to a rigid object)
- *Edge-Edge Spring* is used for collisions between two moving edges.
- *Edge Spring* is used to treat the collision between a moving edge and a fixed edge

Note that a *Vertex Spring* exerts a response force *only* on the vertex as opposed to a *Vertex-Triangle Spring* which exerts force *both* on the vertex and the triangle. Similar, an *Edge Spring* exerts a response force only on the end-points of the moving edge whereas an *Edge-Edge Spring* acts on both the edges. We will first describe the vertex-triangle in detail followed by the rest. The underlying philosophy is largely similar for the rest as well, we shall nevertheless describe the differences in treatments in specific instances. Table 5.2 summarizes the set of parameter values stored in each type of spring.

5.3.3.1 Vertex-Triangle Springs

Vertex Triangle Springs exert forces to push apart a colliding vertex-triangle pair. For dynamic collisions, the detection routine (cf. §4.2.1, §4.2.2 and §4.2.3) gives the exact point of collision $\mathbf{x}(t_c)$ (see Fig. 5.13). A penalty spring is inserted between $\mathbf{x}(t_0)$ and the closest point on the triangle surface in the direction of the normal $\mathbf{x}(t_c)$ with rest length :

$$l_0 = |(\mathbf{x}(t_0) - \mathbf{x}(t_c)) \cdot \hat{\mathbf{n}}| \tag{5.42}$$

Notice that we detect the collisions between the point and the triangle with half thickness $\frac{r}{2}$. This is different from the classic continuous case where we consider the primitives as strictly geometric objects with no thickness. For instance, in §5.2.3.2 where we described the vertex-

triangle collision constraint, we did not take into account of the thickness during the detection stage. We did that only while computing the constraint for response in (5.25).

Let us explain the rationale behind this choice. In the case of QP, we relied on velocity impulses for response which corrected the interpenetrations *instantaneously*. Hence we could afford the luxury of not taking into account of the thickness. But here with an exclusively spring-based solution, the response will be gradual since the response forces needs to be translated into corresponding velocities only after an integration step. Hence we take this additional precaution so that there are no "violent" crossings during the time steps which even when detected cannot be immediately corrected by the spring force. This will also ensure that the initial continuous spring will be placed in the relatively safer "proximity" zone where there are no interpenetrations.

For proximity collisions, the approach is similar. The detection routine (cf. §4.2.1) gives the point in the triangle **abc** closest to the position $\mathbf{x}(t_0)$ i.e. $w_1\mathbf{a}(t_0)+w_2\mathbf{b}(t_0)+w_3\mathbf{c}(t_0)$. The rest length of the spring is thus computed by projecting the gap between these two points in the direction of the triangle normal $\hat{\mathbf{n}}$ using (5.42).

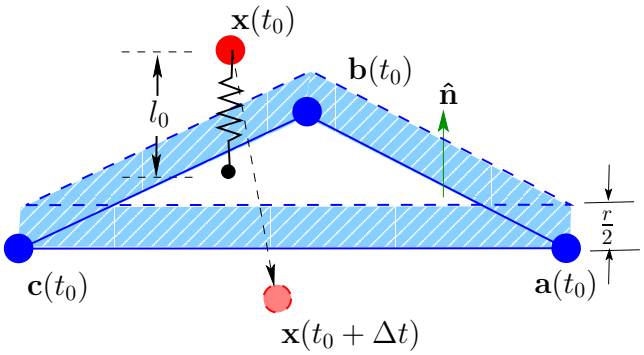


Fig. 5.13: A vertex-triangle spring with rest length l_0 is inserted between the position $\mathbf{x}(t_0)$ and the nearest point to the actual surface $\mathbf{x}_c$.

For both the cases, the force is then computed as :

$$\mathbf{f} = \begin{cases} k(l-l_0)\hat{\mathbf{n}}, & \text{for } l < l_0 \\ 0, & \text{for } l \geq l_0. \end{cases} \tag{5.43}$$

Thus the spring applies a net repulsive force when compressed and a zero force when the length is greater than or equal to the initial rest length. This will ensure that we do not apply any attractive forces. Once computed, the force is split through the appropriate barycentric coordinates. For e.g., a vertex-triangle pair intersecting at the triangle barycentric coordinates (w_1, w_2, w_3), it is $\mathbf{f}$ for the vertex and $-w_1\mathbf{f}$, $-w_2\mathbf{f}$ and $-w_3\mathbf{f}$ respectively for each triangle vertices. The spring force is normally applied if the object continues to be compressed in the forthcoming passes.

The computed force needed are then added to the derivative matrix of (5.1) to find the provisory new state $\mathbf{v}'(t+\Delta t)$ and $\mathbf{x}'(t+\Delta t)$ supposedly devoid of any interpenetrations. It

is provisory because of two reasons. Firstly, the force applied may not be sufficient to prevent persistent "crossings". In such cases, we propose to *double* the stiffness (and hence the force) during the next iteration. Note that we start with a spring stiffness equal to the stiffness of the material. And after each iteration, we progressively double it in case of persistent collision. Secondly, since we are dealing with multiple collisions scenario, this vertex or triangle may still be in contact with other primitives which needs to be resolved as well. The final state is determined only after we have ensured there are no more collisions after we exit the loop (cf. § 5.3.4 for a detailed description of the algorithm).

Also note that once the forces due to continuous springs are applied and the traversing point is "pushed" into the "proximity" zone (the shaded region in Fig. 5.14), we consider it as a discrete spring and continue to apply forces with the original spring stiffness, i.e. we do not double the stiffness for discrete springs since they lie in the "proximity" zone. We remove the spring at the end of the step using a house-keeping routine only if its present length l exceeds the thickness r. This routine also alters the rest length l_0 to the present length l, if $l_0 < l < r$. However, during an iteration the spring stiffness is doubled if the pair continues to interpenetrate.

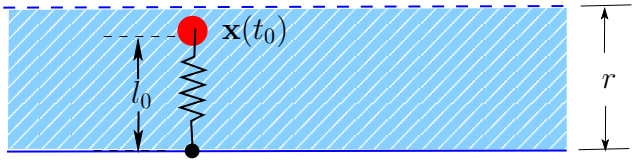


Fig. 5.14: A collision spring with rest length l_0 lying in the 'proximity' zone.

5.3.3.2 A Note on Forces

Let us expand on the characteristic of the collision force. In contrast to penalty approaches, our springs do not introduce repulsive forces at the outset. As long as the present spring length is more than or equal the initial rest length, per (5.43) this spring contributes zero force to $\mathbf{f}$ on the right hand side of (5.1). Only when the spring is further compressed do we apply a net repulsive force. This automatically avoids the addition of extra energy into our system. Note however that there may be net force applied while calculating the gradients $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{v}}$ in (5.1). In contrast, Bridson et al. [BFA02] use an initial repulsive spring to deter away initial collisions while putting a "cap" on the spring deformation ensuring that the movement does not exceed a pre-determined percent of the initial length. In addition, they also perform explicit strain, strain-rate control methods using an iterative Jacobi and Gauss-Seidel techniques. Our algorithm largely avoids these laborious complications.

5.3.3.3 A Note on Normal

For proximity collisions, the normal can be computed using the usual procedure by taking the values at time t, i.e $\hat{\mathbf{n}} = \frac{\mathbf{ab}(t) \times \mathbf{ac}(t)}{\|\mathbf{ab}(t) \times \mathbf{ac}(t)\|}$. However for dynamic cases, it may so happen that

the triangle changes its orientation *during* the time step. Hence for such cases, the normal has to be recalculated based on the positions at time t_c as described in §4.2.4.

5.3.3.4 Vertex Springs

Vertex Springs are used to exert forces upon a vertex colliding with a fixed triangle. We now describe how this spring is activated once a collision is detected for either dynamic (continuous) or static (proximity) case. For dynamic collisions when a moving point traverses a fixed triangle between two time steps, the detection routine (cf. §4.2.2) gives the exact collision point $\mathbf{x}(t_c)$, $t_c \in [t_0, t_0 + \Delta t]$ (see Fig. 5.15). A penalty spring is inserted between $\mathbf{x}(t_0)$ and the closest point on the triangle surface $\mathbf{x}_c$ in the direction of the normal $\hat{\mathbf{n}}$ with rest length l_0 computed using (5.42).

For proximity collisions, the approach is similar. The detection routine gives the point in the triangle **abc** closest to the position $\mathbf{x}(t_0)$ i.e. $w_1\mathbf{a}(t_0) + w_2\mathbf{b}(t_0) + w_3\mathbf{c}(t_0)$. The rest length of the spring is thus computed by projecting the gap between these two points in the direction of the normal $\hat{\mathbf{n}}$. A force $\mathbf{f}$ computing using (5.43) is then applied to the colliding vertex. The update and house keeping routines are similar to the vertex-triangle case. Note that here since the triangle is fixed we do not have to compute the normal at time t_c.

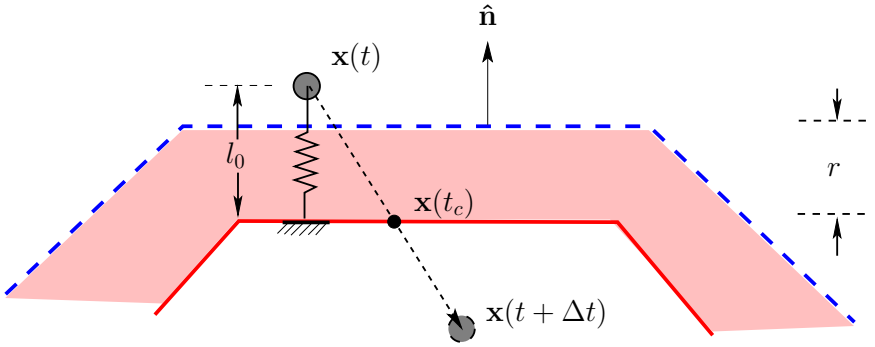


Fig. 5.15: A vertex spring with rest length l_0 is inserted between the position $\mathbf{x}(t_0)$ and the nearest point to the actual surface $\mathbf{x}_c(t)$. The shaded area represents the 'proximity' collision zone denoting the thickness of the surface r.

5.3.3.5 Edge-Edge Springs

We now present the technique for incorporating an edge-edge spring which applies forces on *both* the colliding edges. For dynamic collisions we first detect (cf. §4.2.7) the collision between edges **ab** and **cd** and find the valid time $t_c \in [t_0, t_0 + \Delta t]$ and the barycentric coordinates $(u, v) \in [0, 1]$. We insert a spring between the points at time t_0 with a rest length (see Fig. 5.16) :

$$l_0 = |(\mathbf{a}(t_0) + u\mathbf{ab}(t_0) - \mathbf{c}(t_0) - v\mathbf{cd}(t_0)) \cdot \hat{\mathbf{n}}| \tag{5.44}$$

As we did for the vertex-triangle springs, here again we detect the collisions taking into account of the thickness. Accordingly, we detect collisions between edges $\mathbf{ab}(t_0) - r_i\hat{\mathbf{n}}_{\text{pre}}$ and

$\mathbf{cd}(t_0) + r_j\hat{\mathbf{n}}_{\text{pre}}$. Note that here we need to compute a normal even to determine if there is a collision. The pre-collision normal $\hat{\mathbf{n}}_{\text{pre}}$ can be computed taking the cross product at time t_0, $\hat{\mathbf{n}}_{\text{pre}} = \frac{\mathbf{ab}(t_0)\times\mathbf{cd}(t_0)}{\|\mathbf{ab}(t_0)\times\mathbf{cd}(t_0)\|}$.

The force ($\mathbf{f}$ computed using (5.43)) is split through the appropriate barycentric coordinates i.e. $-(1-u)\mathbf{f}$, $-u\mathbf{f}$ for the first edge $\mathbf{ab}$ and $(1-v)\mathbf{f}$, $v\mathbf{f}$ for the second edge $\mathbf{cd}$. We present the results of the edge-edge spring tested in isolation in Fig. 5.17. Here, we see the edge at the top topple over a fixed edge at the bottom.

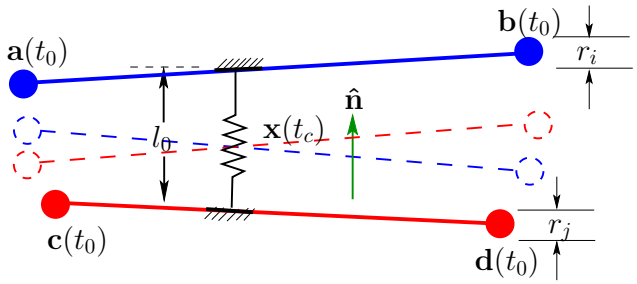


Fig. 5.16: An edge-edge collision spring with rest length l_0 is inserted between the nearest barycentric points at time t_0.

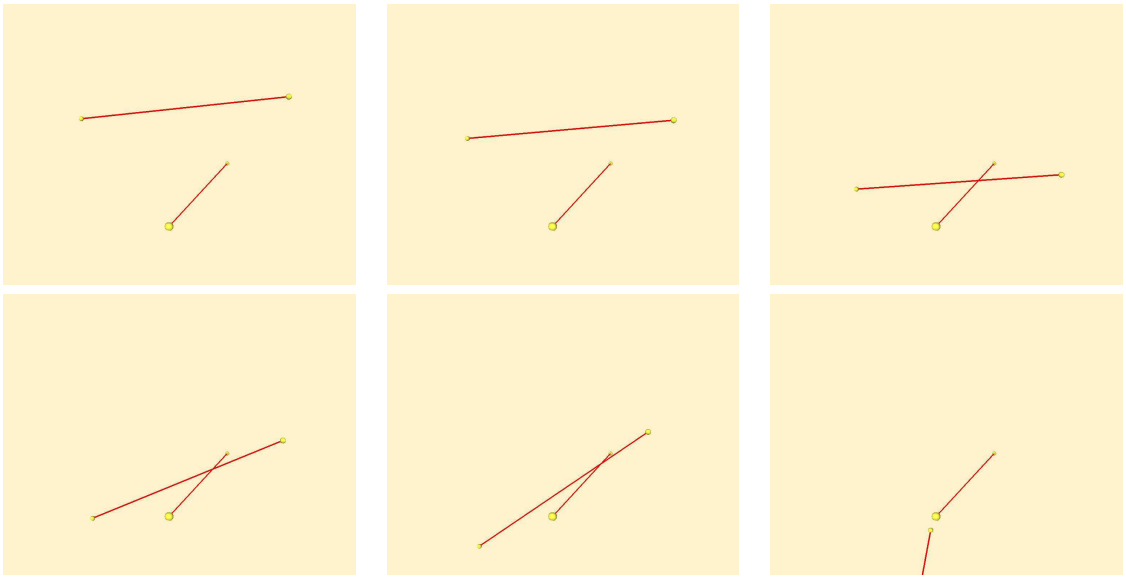


Fig. 5.17: Snapshots of the simulation edge-edge collisions using the collision springs method.

5.3.3.6 Edge Springs

We now present the technique for incorporating an *Edge Spring* which is used to respond to a moving edge $\mathbf{cd}$ colliding with a fixed edge $\mathbf{ab}$. The underlying philosophy is largely similar to the vertex cases. But nonetheless, there are important differences when it comes to the details. The detection routine (cf. §4.2.6), provides the exact time of collision $t_c \in [t_0, t_0 + \Delta t]$ and the barycentric coordinate $(u, v) \in [0, 1]$. We then insert a spring between the points at time t_0 in the direction of the normal $\hat{\mathbf{n}}$ with a rest length l_0 computed using (5.44). For

static collisions, the detection routine (cf. §4.2.5) registers a collision if the distance between them is less than the thickness $r = r_i + r_j$. We then insert a spring between the points at time t_0 in the direction of the normal $\hat{\mathbf{n}}$ as above. The force ($\mathbf{f}$ computed using (5.43)) is split through the appropriate barycentric coordinates i.e. $(1-v)\mathbf{f}$, $v\mathbf{f}$ at each end points of the moving edge $\mathbf{c}$ and $\mathbf{d}$ respectively.

5.3.4 Overall Algorithm

Tab. 5.2: Values stored for different types of spring

Spring Type	Value	Description
Vertex-Triangle Spring	i_v	index of the colliding vertex
	i_a, i_b, i_c	indices of the colliding triangle
	l_0	restlength
	w_1, w_2, w_3	barycentric coordinates of colliding point in triangle
	$\mathbf{n}$	normal
	k	user-defined spring stiffness
	r	thickness of the contact
	t	type of collision (`PROXIMITY` or `CONTINUOUS`)
Vertex Spring	i_v	index of the colliding vertex
	l_0	spring restlength
	$\mathbf{n}$	normal of the fixed triangle
	$\mathbf{x}_c$	other end-point where spring is fixed
	k	user-defined spring stiffness
	r	thickness of the contact
	t	type of collision (`PROXIMITY` or `CONTINUOUS`)
Edge-Edge Spring	i_a, i_b	indices of the first colliding edge
	i_c, i_d	indices of the second colliding edge
	l_0	restlength
	u, v	barycentric coordinates of colliding points in edges
	$\mathbf{n}$	normal
	$\mathbf{x}_c$	other end of the spring fixing
	k	user-defined spring stiffness
	r	thickness of the contact
	t	type of collision (`PROXIMITY` or `CONTINUOUS`)
Edge Spring	i_c, i_d	indices of the colliding edge
	l_0	restlength
	v	barycentric coordinate of the colliding point in edge
	$\mathbf{n}$	normal
	$\mathbf{x}_c$	other end-point where spring is fixed
	k	user-defined spring stiffness
	r	thickness of the contact
	t	type of collision (`PROXIMITY` or `CONTINUOUS`)

The full algorithm is summarized in Algorithm 3. We now explain it in detail each step

of the algorithm.

`detectProximityCollisions` : We perform this method at the beginning of the time step. This routine returns if there is a proximity collision for *each* of the case described in §5.3.3. It returns the indices of the colliding primitive(s), the barycentric values, the rest length and the type of collision. This will enable us to subsequently compute the current spring length and use them to apply forces and update when required.

`addToCollisionSprings` : This method takes up the collision information and adds a spring for each contact. We store the indices of the colliding primitives, the user-set spring stiffness, the rest length (computed using (5.42) and (5.44)), the thickness of the contact (i.e. the sum of the radii $r = r_i + r_j$) and the barycentric coordinates (needed for calculating the present length) and the type of collision (proximity or continuous). Table 5.2 summarizes the set of values stored for each type of spring.

`detectContinuousCollisions` : This method detects the continuous collisions occurring between primitives which is performed *inside* the iteration. This will ensure that any new collision created as a resulting of treating existing ones are detected and handled. This is much needed in the multiple collisions scenario.

`verifyAndUpdateToSprings` : This method again is executed *inside* the loop. The purpose of this method is to update the state of existing springs (say if it becomes a *proximity* from *dynamic*), and change the values of the barycentric coordinates and the normals accordingly. Here, we first check if a spring to be added already exist. If yes, we go ahead and update the above values. If it happens to be a continuous spring, this means that the point has traversed *despite* the presence of a spring. In such cases, we simply double the stiffness value of the spring. This ensures that the a repulsive force twice the value is applied during the next iteration. If the new spring is not present among the existing list of springs, we simply add it to the list.

For edge-edge collisions, this method is slightly altered as follows in order to give a provision of multiple springs between the edges as they slide over. Here in addition to checking if a new edge-edge spring exists already, we also check if it lies very close to the existing one. If yes, it considered as the same spring and the values are updated as the vertex-triangle case. On the other hand, if the new spring lies farther than a certain user-set distance (say more than 10% of the spring length), it is considered as a new spring and is added to the listed. Of course if the spring did not exist in the first place, it is added to the list as above.

`doSpringHouseKeeping` : As shown in the algorithm (see Algo 3) and the flowchart (see Fig. 5.18), we perform this method after we exit the collision loop just at the end of the time step where we update the final states $\mathbf{x}(t_0 + \Delta t)$ and $\mathbf{v}(t_0 + \Delta t)$. Here we evaluate the present rest length l of every collision spring and compare it with the stored thickness value r. If $l > r$, the collision no longer exists and is hence removed. Otherwise, if $l_0 < l < r$, it means that the point has moved further away from collision but still remains in the proximity zone. Hence we update the rest length l_0 with this new length l and springs continue to keep the points from interpenetrating. The above can be expressed as a flowchart in Fig. 5.18.

Algorithm 3 Guaranteed Collision Response Method

```
1: p ← detectProximityCollisions(x(t_0))
2: S ←addToCollisionSprings(p)
3: repeat
4:    Advance x(t_0), v(t_0) to provisory state x'(t_0 + Δt), v'(t_0 + Δt)
5:    Contacts C ← detectContinuousCollisions(x, v', Δt)
6:    /* begin : verifyAndUpdateToSpring*/
7:    for each contact c ∈ C, each spring s ∈ S do
8:       if contact c ≡ s then
9:          Double stiffness k(s) ← k(s) × 2
10:      else
11:         S ← addToCollisionSprings(c)
12:      end if
13:   end for
14:   /* end : verifyAndUpdateToSpring*/
15: until C ← ∅
16: begin : /*doSpringHouseCleaning*/
17: for each collision spring s ∈ S do
18:   Find present rest length l(s) from x'(t_0 + Δt)
19:   if (l(s) ≥ r) then
20:      Remove spring s from the set S
21:   else if (l_0(s) ≥ l) then
22:      Update restlength l_0(s) ← l
23:   end if
24: end for
25: /* end : doSpringHouseCleaning */
26: x(t_0 + Δt) ← x'(t_0 + Δt)
27: v(t_0 + Δt) ← v'(t_0 + Δt)
```

5.3.5 Guaranteed Method Results

We now present the results of the guaranteed collision response algorithm applied to collisions occurring in cloth simulation. Fig. 5.19 shows the case of a initially horizontal cloth (Fig. 5.19(i)) with fixed selected points in the middle highlighted by dark spheres. We then let gravity and internal forces leading to numerous collisions and contacts between the folds (Fig. 5.19 (ii), (iii) and (iv)). Table 5.3 shows the simulation of the parameters used in the simulation. The simulation ran at 8-30 Hz depending on the collision complexity of the scene on a 3GHz Pentium ® 4 PC with 1 GB RAM and GeForce ® 3 graphics card. Our collision response method is able to handle these cases very well as well.Our collision response method is able to handle these cases very well.

Fig. 5.20 shows the case of a initially vertical cloth falling over a flat plane leading to numerous collisions and contacts between the folds. Table 5.4 shows the simulation of the parameters used in the simulation. The simulation ran at 8 Hz on a 3GHz Pentium ® 4 PC with 1 GB RAM and GeForce ® 3 graphics card.. Our collision response method is able to

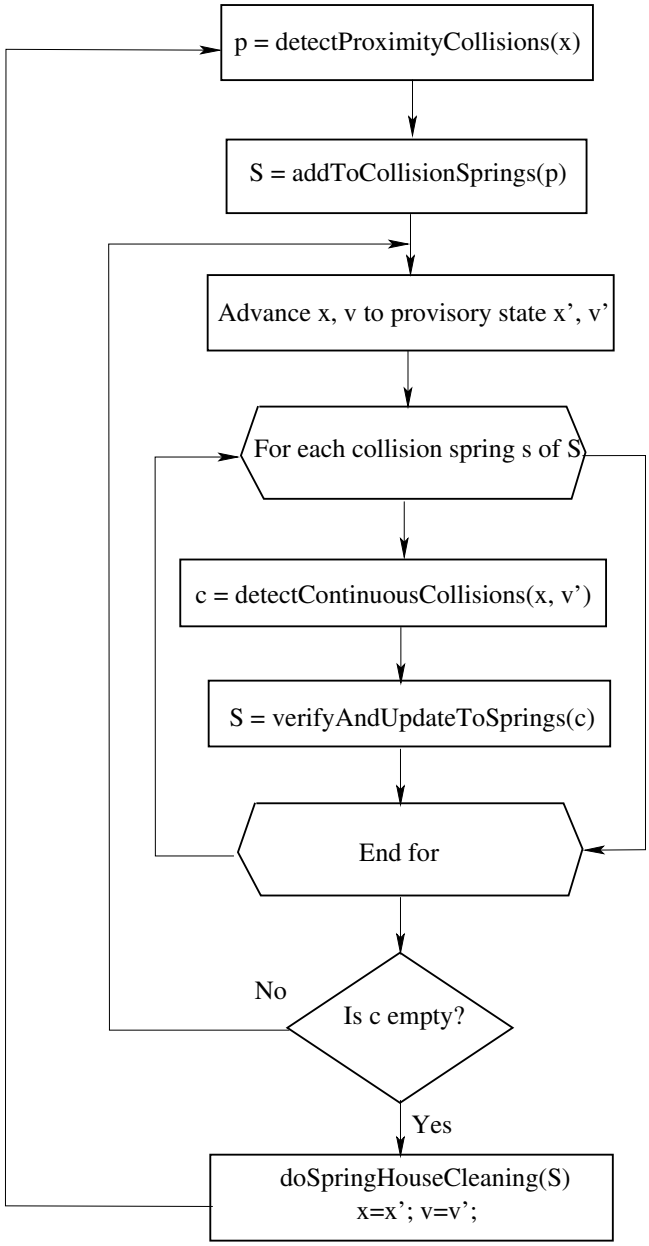


Fig. 5.18: Flowchart illustrating the algorithm.

Tab. 5.3: Guaranteed method results : Parameters used in the simulation shown in Fig. 5.19.

Parameter	Value
Number of cloth particles	400
Thickness	0.01 m
Mass	1.0 kg
Stiffness	$100Nm^{-1}$
Gravity	$-10ms^{-2}$

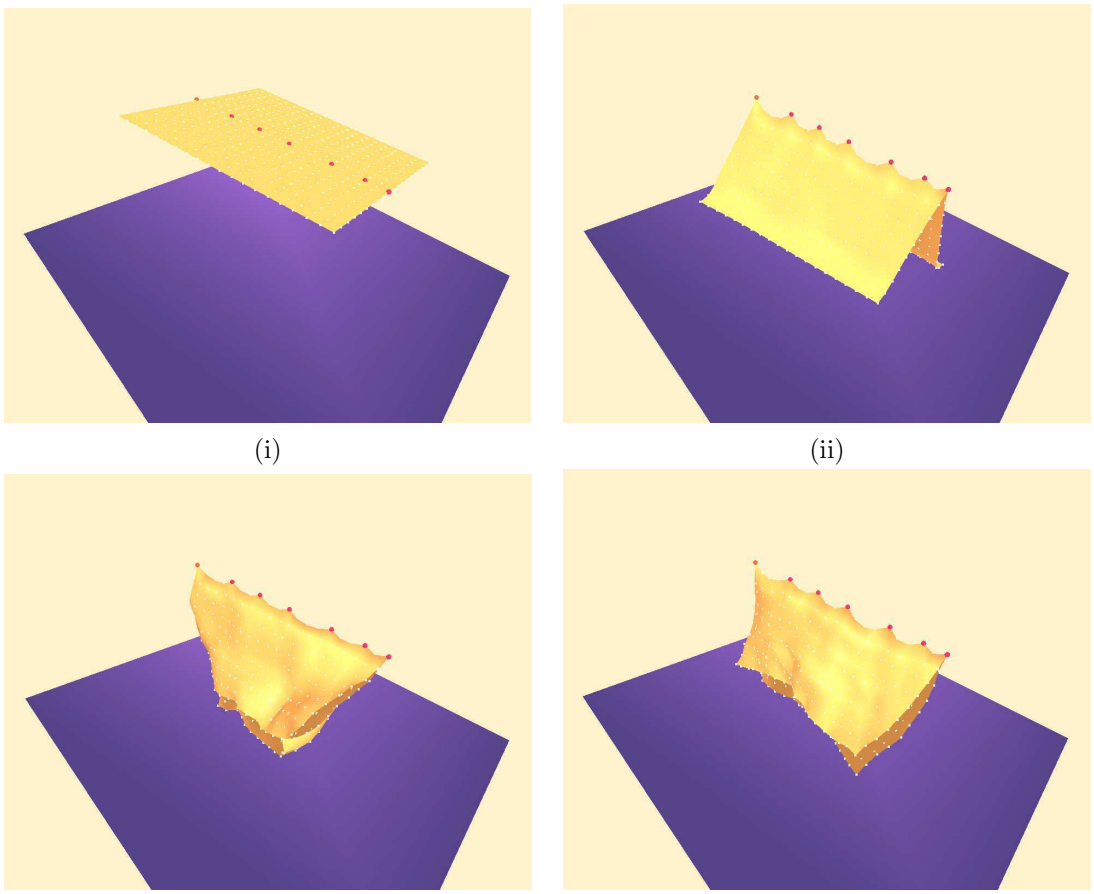


Fig. 5.19: Snapshots of the simulation of a 20x20 deformable cloth fixed in the middle, freely falling and folding over itself creating multiple auto-collisions and contacts which are handled using the guaranteed collision response method.

handle these cases very well as well. Fig. 5.21 shows the plot of the time, iterations and the number of collisions handled for the simulation shown in Fig. 5.20. Fig. 5.21 (i) shows the animation time (implicit Euler integration with the collision springs introduced) required per time step. Fig. 5.21 (ii) separately shows the time taken only for collision detection. Fig. 5.21 (iii) shows the maximum number of collisions handled per time step and Fig. 5.21 (iv) shows the total number of iterations required to resolve all the collisions.

Tab. 5.4: Guaranteed method results : Parameters used in the simulation shown in Fig. 5.20.

Parameter	Value
Number of cloth particles	100
Thickness	0.01
Mass	1.0 kg
Stiffness	$100Nm^{-1}$
Gravity	$-10ms^{-2}$

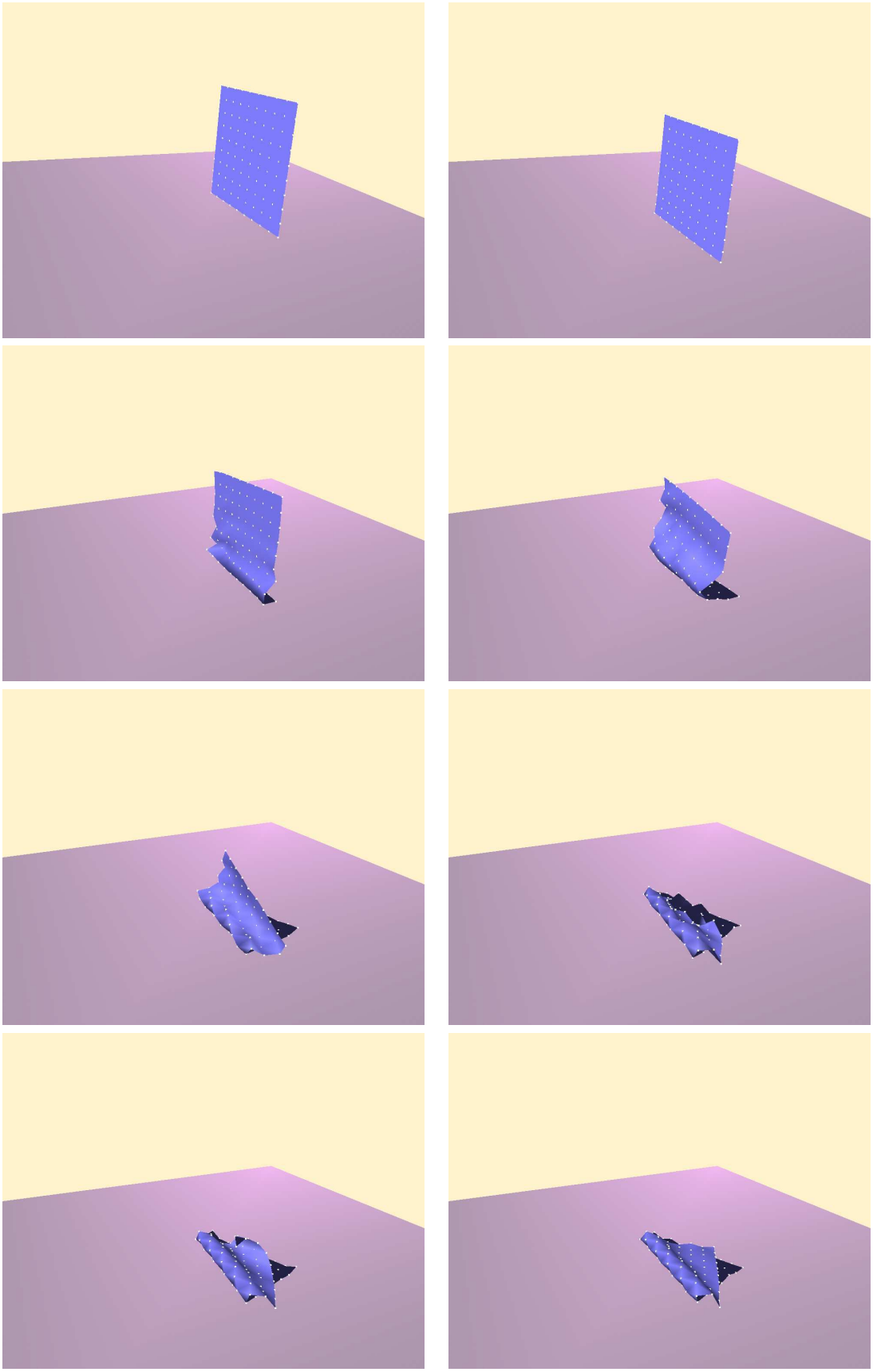


Fig. 5.20: Snapshots of the simulation of a 10x10 deformable cloth falling over a rigid plane creating multiple collisions and contacts which are handled using the guaranteed collision response method.

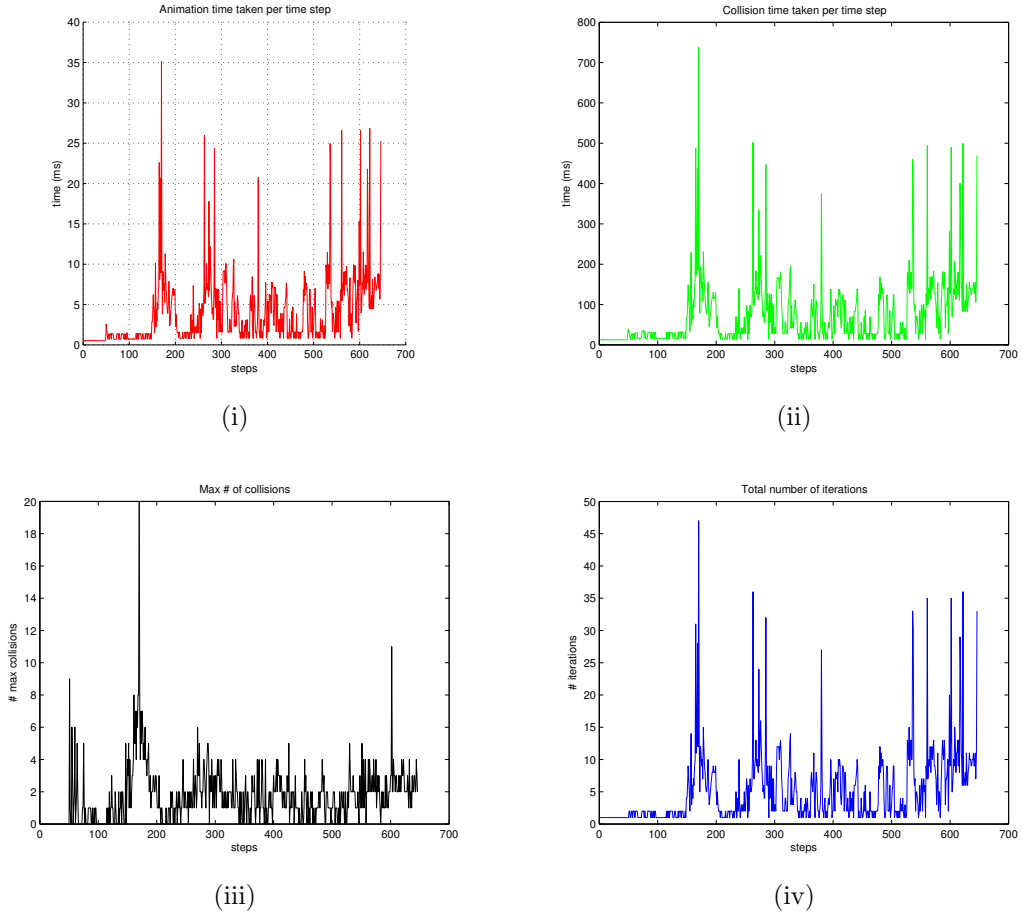


Fig. 5.21: Plot of timings and iterations taken for the simulation shown in Fig. 5.20. (i) Animation time taken to compute the solution per time step. (ii) Collision time taken to compute the solution per time step. (iii) Maximum number of collisions handled within the loop. (iv) Total number of loop iterations required for the algorithm to converge.

5.3.6 Guaranteed Method : Performance Analysis

Having presented the results of our second approach to deal with multiple collisions, we now analyze the performance of this method. We call this method a "guaranteed" collision response method since it ensures that every collision is detected and treated. By continuing to augment the spring stiffness value over multiple collisions, we gradually disentangle the collisions. Thus we neither introduce extra energy into the system nor use Gauss-Seidel style strain rate control system like in [BFA02]. In our experiments we found that the this method does not suffer from numerical errors of constraint-based approaches such as our QP-Collide (cf. §5.2.7). We will also show in the next section §5.4 that the guaranteed method is faster than the QP-based approach by some magnitudes.

However there are certain shortcomings of this method. This method fails on cases when the algorithm is unable to resolve all the collisions. This will result in the spring stiffness doubled at every iteration eventually to a very high numerical value resulting in a bad solution. Note that we were able to simulate the cable case very well with this method (cf. §5.4). The

problem becomes particularly acute due to the nature of the simulations we were attempting such as large piece of deformable cloth colliding with itself. Our straight forward approach of increasing the spring stiffness becomes problematic when there are multiple collisions involving the same primitive.

5.4 Comparative Analysis and Summary

Comparison of performance : After having presented our two new approaches to treating multiple collisions and contacts, we now analyze the computational performance of each of them. For that we first present a simple experiment simulating a cable fixed at the two ends falling under the influence of gravity and coming in contact with a fixed plane below. We used the parameters shown in Table 5.5 for the simulation. Fig. 5.22 shows the simulation of a rigid cable, fixed at two ends colliding with a flat plane. While the simulation appeared

Tab. 5.5: Comparison of the two collision response methods : Parameters used in the simulation.

Parameter	Value
Number of cable particles	40
Mass	1.0 kg
Stiffness	$10^3 Nm^{-1}$
Gravity	$-10ms^{-2}$

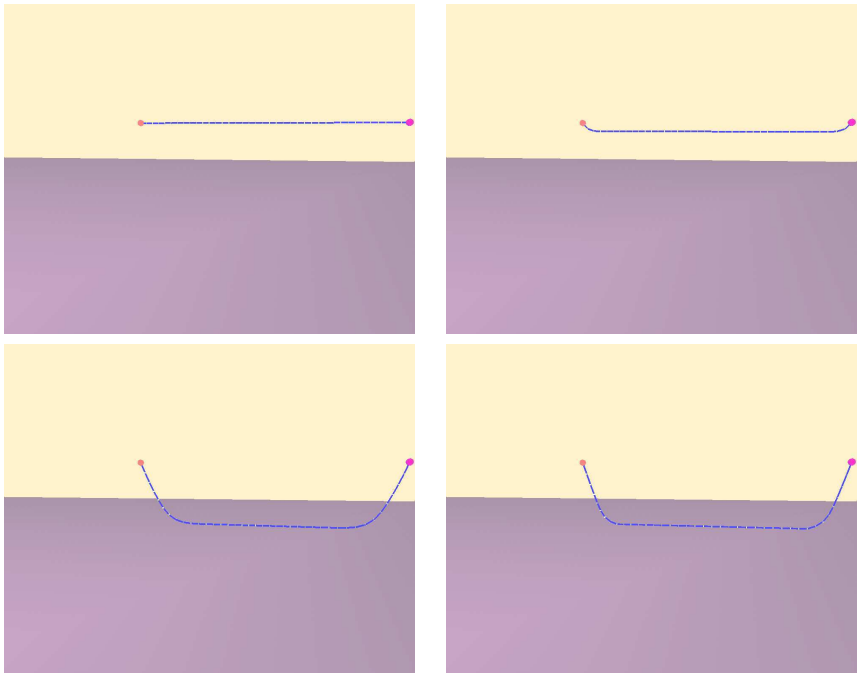


Fig. 5.22: Simulation to compare the performances of the QP-Collide method and the guaranteed method.

similar visually for both the methods, there are differences in performance. Fig. 5.23, 5.24 and 5.25 show the plots of the animation time, number of collisions and the total number variables and constraints handled using each method for the simulation shown above. The number of constraints handled were more or less similar (see Fig. 5.24 and 5.25), the time required (*excluding* the time needed for collision detection) for the spring-based approach is lower than that of the QP (see Fig. 5.23). Also note that the timings for QP-collide method has several "spikes" much more than that of the guaranteed method. We attribute them to the numerical errors as a result of the degradation of the matrix conditioning. The guaranteed method is largely devoid of such spikes resulting in a constant simulation time per time step.

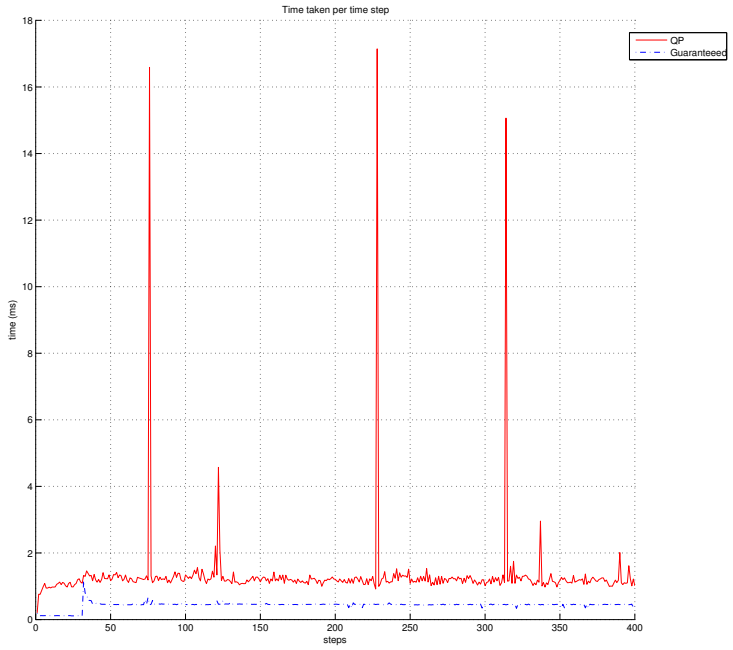


Fig. 5.23: Plot of the animation time (*excluding* collision detection) taken to compute the solution per time step for the simulation shown in Fig. 5.22.

Summary : In this chapter, we first presented a new approach of solving the problem of multiple collisions. The quadratic programming algorithm uses velocity impulses within a global framework which can handle many types of constraints including collisions. With an iterative solver such as the conjugate gradient, we can get an efficient $O(n)$ solution. A global approach naturally does not introduce additional strain into the system. However, we saw cases where the conditioning of the matrices were deteriorated. While we proposed several fixes to these known problems, it is clear that the numerical errors in general are more than a spring based matrix. Also note that with this method, if we have to really ensure that all the collisions are treated, we have got to perform multiple collision detection and response passes. This is computationally very expensive and will then become the equivalent of performing a

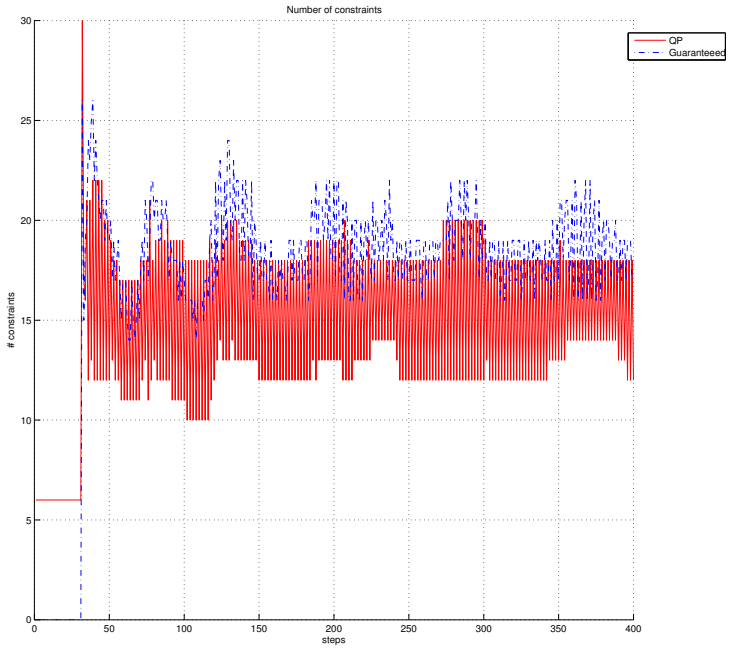


Fig. 5.24: Plot of the number of collision constraints handled per time step for the simulation shown in Fig. 5.22.

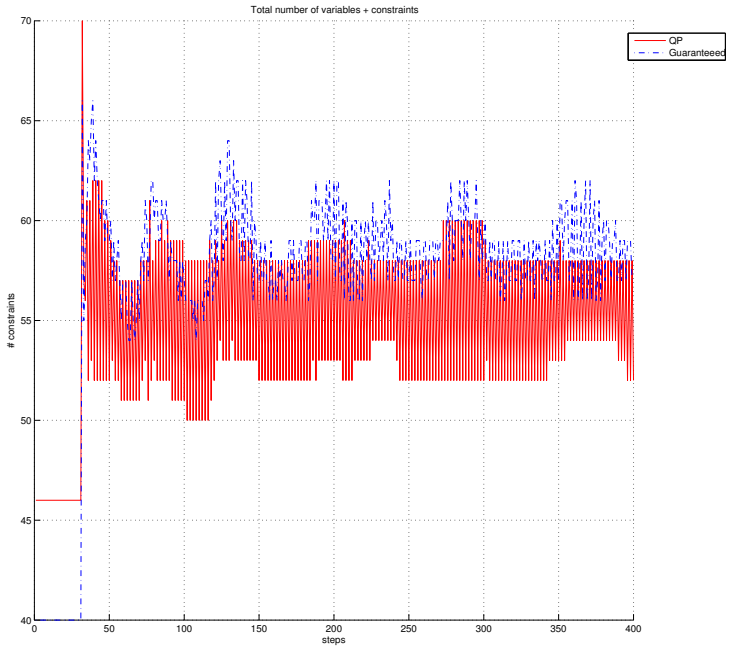


Fig. 5.25: Plot of the total number of variables + constraints handled per time step for the simulation shown in Fig. 5.22.

sequential quadratic programming.

This brings to our second contribution based on springs exploiting temporal coherence. Here we saw how discrete and continuous collisions are handled slightly differently. By checking for new collisions inside a loop, we are able to ensure that none of them are missed. Unlike the QP approach, the addition of new springs does not make the matrix ill-conditioned. It is also computationally advantageous since the addition of new springs can be taken care using efficient sparse matrix calculations.

CHAPTER 6

Conclusions

All good things come to an end.

So far, we presented the results of research work we did to order solve the difficult problem of multiple collisions and contacts. In this chapter, we recap the list of our contributions and give a certain number of perspectives for future work before concluding with remarks on the general state of physically based modeling.

6.1 Thesis Summary

In this thesis work, we have attempted to solve some of the open problems towards the robust simulation of deformable objects. We began in chapter 1 by introducing the domain of the problem we seek to address within the field of computer graphics. They can be broadly classified as physically-based modeling in general and animation, collision detection and response in particular. Then in chapter 2, we comprehensively listed and described the relevant earlier research work presented by the computer graphics community. By highlighting the relevant techniques and their drawbacks, we set the stage to take on the problem.

We then proceeded to chapter 3 in order to explain the problem we seek solve with the help of two real-world examples. First, we proposed a new approach for modeling and animating the intestine and the mesentery. The challenge here was to model and animate

it in and the more difficult problem of having to deal with the multiple collisions and self-collisions occurring during simulation. The problem became compounded due to real-time requirements. We saw how a simple geometric model can solve the problem of modeling a complicated organ such as the intestine. We then proposed a novel approach for efficient collision and self-collision detection based on stochastic feature pair tracking. In addition, our displacement-velocity collision response technique was fast enough to make the entire simulation run in real-time on normal PCs. Second, we proposed an efficient approach with mechanics based on implicit mass-spring system integration coupled with fast octree-based collision detection for modeling extremely stiff objects such as a mechanical cable. Here in addition to using generic approaches such as the conjugate gradient method, we saw how a specialized solver such as the banded LU can be used for efficiently treating the specific problem. While the above two approaches fairly took care of the problems at hand, they had their drawbacks. In the case of intestine we saw interpenetrations involving thin tissues partly due to the lack of continuous collision detection. In the case of the stiff cable example, we saw how a simple localized collision response can lead to increase in strain rate and hence instabilities in the simulation. We thus drew two conclusions from our initial research foray. First, it illustrated that there is not a single best solution that works well in all cases. Second, there is a case for a more robust collision handling scheme.

This brings to chapter 4, where we comprehensively presented the methods needed for the robust detection of colliding geometric primitives. In addition to describing some of the existing efficient methods, we also presented new approaches which also handled degenerate cases. This is the first time to our knowledge that such cases were handled.

In chapter 5, we addressed one of the core problems of this thesis work, that of dealing with multiple collisions and contacts. Here, we attempted to respond to the collisions once they are detected using the techniques presented in chapters 3 and 4. We proposed two new algorithms to solve the problem at hand. The first one is a constraint-based method which treated collisions as linear constraints. Using quadratic programming techniques, we computed a global solution which gave a set of velocity impulses solving both the dynamic equations and the collision constraints. This method while simple to implement was also versatile being able to handle other constraints such as strain control, fixing a point, etc. Thanks to this method, we were able to simulate very stiff objects such as a mechanical cable colliding at multiple points. We observed that this method has linear computational complexity proportional to the number of constraints handled. Of course, since we used an iterative solver that number also depends on the error tolerance limit which the user set. We also presented the drawbacks of this approach. The addition of constraints somewhat degrades the conditioning of the global stiffness matrix resulting in numerical errors accumulating over iteration. There were also cases when the algorithm failed to converge since the constraints kept toggling between the active and passive sets. Nevertheless, we hoped that this approach could be a new start to solve the problem in dynamics simulation of deformable bodies. This method which overall worked well for the cable nevertheless is not suitable for treating

collisions and self-collisions occurring in objects such as cloth. The problem is not that of collision detection itself, but the need to do multiple passes in order to account for the creation of secondary collisions while we treat the existing ones. Using QP for such cases effectively means we have to do multiple passes which will computationally heavy. Hence believed that a new spring based technique will be more suitable for this kind of problem.

Our second method effectively handled more complex problems involving multiple collisions and contacts. By using a spring-based method it avoided the numerical problems suffered by adding constraints. By repeatedly checking for new collisions, we ensure that no collision will be missed. For this purpose, we exploit temporal coherence by updating the collision springs existing from the previous steps or loops. We were able to simulate more complex simulations such as cloth folding over itself while also colliding with another object.

Finally, comparing both these methods for the same simulation showed us that the spring-based approach took relatively lesser time while showing similar visual results. Fig. 6.1 presents a sample of the results we produced as a result of this thesis work.

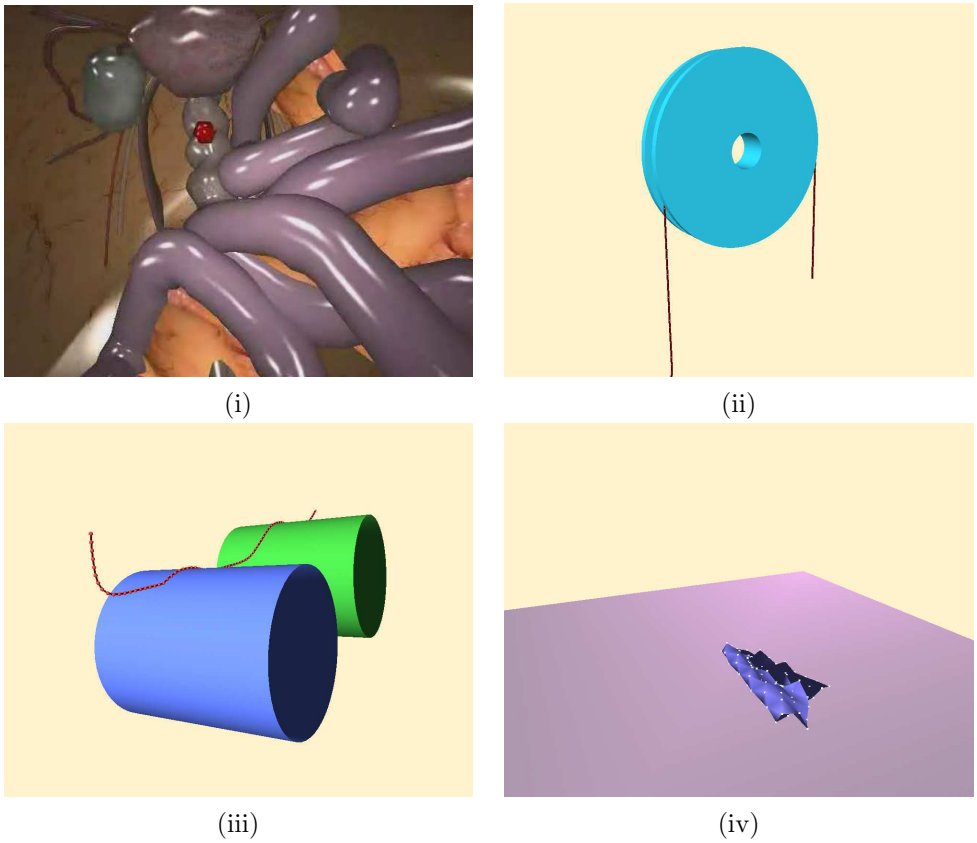


Fig. 6.1: A sample of results from this thesis work. (i) Simulation of intestine and mesentery inside a virtual abdominal cavity. (ii) A cable falling over a pulley with an initial velocity and and sliding over. (iii) Simulation of a stiff cable with 80 mass particles fixed at two ends falling over two cylinders. (iv) Simulation of a 10x10 deformable cloth falling over a rigid plane and self-colliding.

6.2 Future Work

We realize that one of the key issues for scientific advancement is through constant innovation. Here, we present the possible extensions to our work given the rapid technological progress we are witnessing.

Addressing numerical errors in QP : We showed in this thesis that for solving the problem of multiple collisions and contacts, a global solution approach is well-suited. However, we fall into the familiar trap of numerical errors and the degradation of matrix conditioning. We feel that more research can be pursued to analyze this problem in order to find a cost-effective solution. The solution most likely will involve some sort of pre-treatment in a sequential quadratic programming setup [dGO97]. This will require a factorization step such as QR decomposition with $O(n^3)$ complexity. While this may appear computationally expensive by today's standards, with the fast growth of processor speeds and the advent of multi-core architecture available from commercial microprocessor vendors such as Intel ® [Int06] and AMD ® [AMD06].

Exploiting GPU Power : The amount of raw computational power available in modern graphics processors (GPUs) far exceed that of a CPU. For example an nVIDIA ® GeForce ® 6800 can deliver a peak performance of 45 GFLOPS [1] as opposed to an Intel ® 3 GHz Pentium ® 4 at 12 GFLOPS. Moreover, Fig. 6.2 shows that the rate of the increase in computational power of a GPUs far exceeds that of CPUs which generally follow the Moore's law [Moo65] which states that the number of components (like transistors) inside a processor doubles every 18 months. In general CPUs are designed to deal with general purpose applications which have less parallelism and more complex control requirements compared to a GPU which targets a rendering pipeline. Hence GPUs exploit data parallelism much better than a CPU [Owe05]. There have been recent attempts by graphics card vendors such as nVIDIA to present a system which can handle the computationally heavy parts in GPU. This is done by empowering the user through the availability of programmable GPUs with vertex and pixel shaders and use of a high-level shading language such as the Cg. Indeed, recently there have been attempts to harness power of GPU to do calculations needed for physical simulations [Har06].

6.3 Final Thoughts

As we approach the end of this thesis, we would like to present some personal reflections on research in physically-based modeling. It is evident that we have come a long way in proposing bold and innovative approaches to solve the existing problems and open up new vistas. The realism of the simulation is getting increasingly sophisticated by the day thanks to the improvement in computational power and new algorithms. As more computational power

[1] 10^9 floating point operations per second.

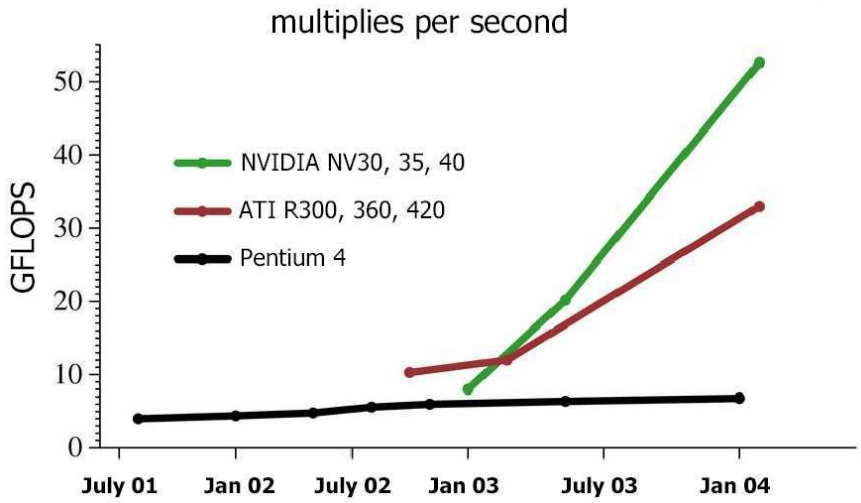


Fig. 6.2: Recent trends comparing the computational performance of GPUs and CPUs [BFH+04]

reaches the end-user, this trend is likely to accelerate resulting in the widespread adaption of new technologies. And yet, we as researchers should not sit back and relax assuming that a job is well done. If at all, the journey has got more interesting.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[AB03] U. Ascher and E. Boxerman. On the Modified Conjugate Gradient Method in Cloth Simulation. *The Visual Computer*, 19(7–8) :523–531, 2003.

[ABR02] L. Augusten, R.A. Bowen, and M. Rouge. *Pathophysiology of the Digestive System*. Department of Biomedical Sciences, Colorado State University, 2002. Available at : `http ://arbl.cvmbs.colostate.edu/hbooks/pathphys/digestion`.

[AdG+02] P.K. Agarwal, M.T. de Berg, J.G. Gudmundsson, M. Hammar, and H.J. Haverkort. Box-Trees and R-Trees with Near-Optimal Query Time. *Discrete and Computational Geometry*, 28(3) :291–312, 2002.

[AMD06] AMD. Multi-Core Processors - The Next Evolution in Computing. Advanced Micro Devices, Inc., 2006. Available at : `http ://multicore.amd.com`.

[BAC+06] F. Bertails, B. Audoly, M.-P. Cani, B. Querleux, F. Leroy, and J.-L. Lévêque. Super-Helices for Predicting the Dynamics of Natural Hair. In *Proc. SIGGRAPH '06*, pages 1180–1187. ACM Press, 2006.

[Bar89] D. Baraff. Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies. In *Proc. SIGGRAPH '89*, pages 223–232. ACM Press, 1989.

[Bar90] D. Baraff. Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation. In *Proc. SIGGRAPH '90*, pages 19–28. ACM Press, 1990.

[Bar91] D. Baraff. Coping with Friction for Non-penetrating Rigid Body Simulation. In *Proc. SIGGRAPH '91*, pages 31–40. ACM Press, 1991.

[Bar94] D. Baraff. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *Proc. SIGGRAPH '94*, pages 23–34. ACM Press, 1994.

[Bar97] R. Barzel. Faking Dynamics of Ropes and Springs. *IEEE Computer Graphics and Applications*, 17(3) :31–39, 1997.

[Bat95] K. J. Bathe. *Finite Element Procedures*. Prentice Hall Inc., 1995.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. In *Proc. SIGGRAPH '02*, pages 594–603. ACM Press, 2002.

[BFH+04] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs : Stream Computing on Graphics Hardware. In *Proc. SIGGRAPH '04*, pages 777–786. ACM Press, 2004.

[BHG91] D.E. Breen, D.H. House, and P.H. Getto. A Particle-Based Computational Model of Cloth Draping Behavior. In *Proc. Computer Graphics International '91*, pages 113–134, 1991.

[BJ05] J. Barbic and D. L. James. Real-Time Subspace Integration of St.Venant-Kirchhoff Deformable Models. In *Proc. SIGGRAPH '05*, pages 982–990. ACM Press, 2005.

[Bli98] J. Blinn. SIGGRAPH '98 Keynote Address. In *ACM SIGGRAPH Conference Report*. ACM Press, 1998.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of Clothing with Folds and Wrinkles. In *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 28–36. ACM Press, 2003.

[BMWM01] D. E. Breen, S. Mauch, R. T. Whitaker, and J. Mao. 3D Metamorphosis Between Different Types of Geometric Models. In *Proc. Eurographics '01*, pages 36–48. Blackwell Publishing, 2001.

[BO02] G. Bradshaw and C. O'Sullivan. Sphere-Tree Construction Using Dynamic Medial Axis Approximation. In *Proc. Symposium on Computer Animation*, pages 33–40. ACM Press, 2002.

[Boy79] J. W. Boyse. Interference Detection Among Solids and Surfaces. *Communications of the ACM*, 22(1) :3–9, 1979.

[BPK+02] P.-T. Bremer, S. Porumbescu, F. Kuester, B. Hamann, K. I. Joy, and K.-L. Ma. Virtual Clay Modeling Using Adaptive Distance Fields. In *Proc. International Conference on Imaging Science, Systems, and Technology (CISST '02)*, 2002.

[BW92] D. Baraff and A. Witkin. Dynamic Simulation of Non-penetrating Flexible Bodies. In *Proc. SIGGRAPH '92*, pages 303–308. ACM Press, 1992.

[BW98] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *Proc. SIGGRAPH '98*, pages 43–54. ACM Press, 1998.

[BW01] D. Baraff and A. Witkin. Physically Based Modeling. In *Proc. ACM SIGGRAPH Course Notes*. ACM Press, 2001.

[BW02] G. Baciu and W. S. K. Wong. Hardware-assisted Self-Collision for Deformable Surfaces. In *Proc. ACM Symposium on Virtual Reality Software and Technology*, pages 129–136. ACM Press, 2002.

[BWK03] D. Baraff, A. Witkin, and M. Kass. Untangling Cloth. In *Proc. SIGGRAPH '03*, pages 862–870. ACM Press, 2003.

[BWS99] G. Baciu, W. S-.K. Wong, and H. Sun. RECODE : An Image-based Collision Detection Algorithm. *Journal of Visualization and Computer Animation*, 10 :181–192, 1999.

[Can86] J. F. Canny. Collision Detection for Moving Polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2) :200–209, 1986.

[CDA99] S. Cotin, H. Delingette, and N. Ayache. Real-time Elastic Deformations of Soft Tissues for Surgery Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1) :62–73, 1999.

[CGC+02] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. A Multiresolution Framework for Dynamic Deformations. In *Proc. SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 41–48. ACM Press, 2002.

[CJY02] J. Chang, J. Jin, and Y. Yu. A Practical Model for Hair Mutual Interactions. In *ACM SIGGRAPH /Eurographics Symposium on Computer Animation*, pages 73–80. ACM Press, July 2002.

[CK02] K-J. Choi and H.-S. Ko. Stable but Responsive Cloth. In *Proc. SIGGRAPH '02*, pages 604–611. ACM Press, 2002.

[COSL98] D. Cohen-Or, A. Solomovic, and D. Levin. Three-Dimensional Distance Field Metamorphosis. *ACM Transactions on Graphics*, 17(2) :116–141, 1998.

[DB00] S. De and K. J. Bathe. The Method of Finite Spheres. *Computational Mechanics*, 25(4) :329–345, 2000.

[DC95] M. Desbrun and M.-P. Cani. Animating Soft Substances With Implicit Surfaces. In *Proc. SIGGRAPH '95*, pages 287–290. ACM Press, 1995.

[DC96] M. Desbrun and M.-P. Cani. Smoothed Particles : A New Paradigm for Animating Highly Deformable Dodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76. Springer-Verlag, 1996.

[DDCB01] G. Debunne, M. Desbrun, M. P. Cani, and A. H. Barr. Dynamic Real-Time Deformations Using Space and Time Adaptive Sampling. In *Proc. SIGGRAPH '01*, pages 31–36. ACM Press, 2001.

[DG04] G. Debunne and S. Guy. Monte-Carlo Collision Detection. Tech. Report RR-5136, INRIA, March 2004.

[dGO97] M.T. de Gouvêa and D. Odloak. Dealing with Inconsistent Quadratic Programs in a SQP Based Algorithm. *Brazilian Journal of Chemical Engineering*, 14(1) :63–80, 1997.

[Ebe00] D. H. Eberly. *3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2000.

[EL01] S. A. Ehmann and M. C. Lin. Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition. *Computer Graphics Forum*, 20(3) :500–510, 2001.

[ES99] J. Eckstein and E. Schömer. Dynamic Collision Detection in Virtual Reality Applications. In *Proc. 7th International Conference in Central Europe on Computer Graphics and Visualization and Interactive Digital Media*, pages 71–78, 1999.

[FAM+02] L. France, A. Angelidis, P. Meseure, M. P. Cani, J. Lenoir, F. Faure, and C. Chaillou. Implicit Representations of the Human Intestines for Surgery Simulation. In M. Thiriet, editor, *Modeling & Simulation for Computer Aided Medicine & Surgery*, Proc. ESAIM, vol. 12, pages 42–47. SMAI, Paris, 2002.

[Fin00] A. P. Fine. *Patient Information : Laparoscopic Colon Surgery.* Society of Laparoendoscopic Surgeons, Inc., 2000. Available at : `http ://www.sls.org/patientinfo/colon.html`.

[FL01] S. Fisher and M. C. Lin. Deformed Distance Fields for Simulation of Non-Penetrating Flexible Bodies. In *Proc. Eurographics Workshop on Computer Animation and Simulation '01*, pages 99–111. Springer-Verlag, 2001.

[Fle87] R. Fletcher. *Practical Methods of Optimization*, chapter 10.3 : Quadratic Programming, pages 229–241. John Wiley & Sons, New York, second edition, 1987.

[FLMC02] L. France, J. Lenoir, P. Meseure, and C. Chaillou. Simulation of a Minimally Invasive Surgery of Intestines. In *Proc. Virtual Reality International Conference*, 2002.

[FPRJ00] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively Sampled Distance Fields : A General Representation of Shape for Computer Graphics. In *Proc. SIGGRAPH '00*, pages 249–254. ACM Press, 2000.

[FSG03] A. Fuhrmann, G. Sobotka, and C. Gross. Distance Fields for Rapid Collision Detection in Physically Based Modeling. In *Proc. GraphiCon 2003*, pages 58–65, September 2003.

[fVS98] Center for Videoendoscopic Surgery. *Laparoscopic Colectomy.* Deptartment of Surgery, University of Washington, 1998. Available at : `http ://depts.washington.edu/cves/lapcol.html`.

[FW99] C.-S. Fahn and J.-L. Wang. Efficient Time-Interrupted and Time-Continuous Collision Detection Among Polyhedral Objects in Arbitrary Motion. *Journal of Information Science and Engineering*, 15 :769–799, 1999.

[GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex Rigid Bodies With Stacking. In *Proc. SIGGRAPH '03*, pages 871–878. ACM Press, 2003.

[GC94] J.-D. Gascuel and M.-P. Cani. Displacement Constraints for Interactive Modeling and Animation of Articulated Structures. *The Visual Computer*, 10(4), March 1994.

[GDO00] F. Ganovelli, J. Dingliana, and C. O'Sullivan. BucketTree : Improving Collision Detection Between Deformable Objects. In *Proc. of Spring Conference on Computer Graphics SCCG '00*, pages 156–163, 2000.

[Gib97] S. Gibson. 3D Chainmail : A Fast Algorithm for Deforming Volumetric Objects. In *Proc. Symposium on Interactive 3D Graphics*, pages 149–154. ACM Press, 1997.

[GK03] G. Grabner and A. Kecskeméthy. Reliable Multibody Collison Detection Using Runge-Kutta Integration Polynomials. In *Proc. IDMEC/IST*, Lisbon, Portugal, July1–4 2003.

[GKS02] E. Grinspun, P. Krysl, and P. Schröder. CHARMS : A Simple Framework for Adaptive Simulation. In *Proc. SIGGRAPH '02*, pages 281–90. ACM Press, 2002.

[Gla89] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[GLM96] S. Gottschalk, M.C. Lin, and D. Manocha. OBBTree : A Hierarchical Structure for Rapid Interference Detection. In *Proc. SIGGRAPH '96*, pages 171–80. ACM Press, 1996.

[GM03] L. Grisoni and D. Marchal. High Performance Generalized Cylinder Visualization. In *Proc. Shape Modeling International '03*, pages 257–63. IEEE CS Press, 2003.

[GMS02] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT : An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Journal on Optimization*, 12(4) :979–1006, 2002.

[GNRZ02] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision Detection for Deforming Necklaces. In *ACM Symposium on Computational Geometry*, pages 33–42. ACM Press, 2002.

[GRLM03] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE : Interactive Collision Detection Between Complex Models in Large Environments Using Graphics Hardware. In *Proc. of ACM Graphics Hardware*, pages 25–32. ACM Press, 2003.

[GS87] J. Goldsmith and J. Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7(5) :14–20, May 1987.

[Hah88] J. K. Hahn. Realistic Animation of Rigid Bodies. In *Proc. SIGGRAPH '88*, pages 299–308. ACM Press, 1988.

[Har06] M. Harris. Physics on NVIDIA GPUs. In *Proc. ACM SIGGRAPH Course Notes*. ACM Press, 2006. Available at : `http ://developer.nvidia.com/`.

[HBZ90] B. Von Herzen, A. H. Barr, and H. R. Zatz. Geometric Collisions for Time-Dependent Parametric Surfaces. In *Proc. SIGGRAPH '90*, pages 39–48. ACM Press, 1990.

[HKL+99] K. E. Hoff III, J. Keyser, M.C. Lin, D. Manocha, and T. Culver. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In *Proc. SIGGRAPH '99*, pages 277–286. ACM Press, 1999.

[HMT01] S. Hadap and N. Magnenat-Thalmann. Modeling Dynamic Hair as a Continuumn. *Computer Graphics Forum*, 20(3) :329–338, 2001.

[HTG03] B. Heidelberger, M. Teschner, and M. Gross. Real-Time Volumetric Intersections of Deforming Objects. In *Proc. Vision, Modeling, Visualization '03*, pages 461–468, 2003.

[HTG04] B. Heidelberger, M. Teschner, and M. Gross. Detection of Collisions and Self-Collisions Using Image-Space Techniques. In *Proc. of WSCG '04*, 2004.

[Hub96] P. M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3) :179–210, July 1996.

[Int06] Intel. Intel Multi-Core Processors : Leading the Next Digital Revolution. Intel Corp., 2006. Available at : `http ://www.intel.com/multi-core`.

[IZLM01] K. E. Hoff III, A. Zaferakis, M. C. Lin, and D. Manocha. Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware. In *Symposium on Interactive 3D Graphics*, pages 145–148. ACM Press, 2001.

[JF03] D. L. James and K. Fatahalian. Precomputing Interactive Dynamic Deformable Scenes. In *Proc. SIGGRAPH '03*, pages 879–887. ACM Press, 2003.

[JP99] D. L. James and D. K. Pai. ArtDefo - Accurate Real Time Deformable Objects. In *Proc. SIGGRAPH '99*, pages 65–72. ACM Press, 1999.

[JP02] D. L. James and D. K. Pai. DyRT : Dynamic Response Textures for Real Time Deformation Simulation with Graphics Hardware. In *Proc. SIGGRAPH '02*, pages 582–85. ACM Press, 2002.

[JP04] D. L. James and D. K. Pai. BD-Tree : Output-Sensitive Collision Detection for Reduced Deformable Models. In *Proc. SIGGRAPH '04*, pages 393–398. ACM Press, 2004.

[JTT01] P. Jiménez, F. Thomas, and C. Torras. 3D Collision Detection : A Survey. *Computers and Graphics*, 25(2) :269–85, 2001.

[KGL+98] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and Accurate Contact Determination Between Spline Models Using ShellTrees. *Computer Graphics Forum*, 17(3), September 1998.

[KHM+98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowrizal, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1) :21–36, January 1998.

[KmLM02] Y. J. Kim, K. E. Hoff III, M. C. Lin, and D. Manocha. Closest Point Query Among the Union of Convex Polytopes Using Rasterization Hardware. *Journal of Graphics Tools*, 7(4) :43–52, 2002.

[KOLM02] Y. Kim, M. Otaduy, M. Lin, and D. Manocha. Fast Penetration Depth Computation for Physically-based Animation. In *Proc. ACM Symposium on Computer Animation '02*, pages 23–31. ACM Press, 2002.

[KP03] D. Knott and D. Pai. CinDeR : Collision and Interference Detection in Real-Time Using Graphics Hardware. In *Proc. Graphics Interface '03*, 2003.

[KR03] B. Kim and J. Rossignac. Collision Prediction for Polyhedra Under Screw Motions. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 4–10. ACM Press, 2003.

[LAM01] T. Larsson and T. Akenine-Möller. Collision Detection for Continuous Deforming Bodies. In *Proc. Eurographics Short Presentations '01*, pages 325–333. Blackwell Publishing, 2001.

[Las95] J. Lasseter. *Toy Story*, Pixar/Walt Disney Pictures, 1995.

[LC92] M.C. Lin and J.F. Canny. Efficient Collision Detection for Animation. In *Proc. 3rd Eurographics Animation & Simulation Workshop*, 1992.

[LCN99] J. C. Lombardo, M. P. Cani, and F. Neyret. Real-Time Collision Detection for Virtual Surgery. In *Proc. Computer Animation '99*, pages 82–91. IEEE CS Press, 1999.

[LG98] M.C. Lin and S. Gottschalk. Collision Detection between Geometric Models : A Survey. In *Proc. IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[LSW99] C. Lennerz, E. Schöemer, and T. Warken. A Framework for Collision Detection and Response. In *11th European Simulation Symposium*, pages 309–314, 1999.

[Luc99] G. Lucas. *Star Wars : Episode I - The Phantom Menace*, Lucasfilm Ltd., 1999.

[MC00] P. Meseure and C. Chaillou. A Deformable Body Model for Surgical Simulation. *Journal of Visualization and Computer Animation*, 11(4) :197–208, September 2000.

[MDDB00] M. Meyer, G. Debunne, M. Desbrun, and A.H. Barr. Interactive Animation of Cloth-like Objects in Virtual Reality. *Journal of Visualization and Computer Animation*, 12(1) :1–12, 2000.

[MDH+03] P. Meseure, J. Davanne, L. Hilde, J. Lenoir, L. France, F. Triquet, and C. Chaillou. A Physically based Virtual Environment Dedicated to Surgical Simulation. In *Surgery Simulation & Soft Tissue Modeling*, pages 38–46. Springer, 2003.

[Mel00] S. Melax. Dynamic Plane Shifting BSP Traversal. In *Proc. Graphics Interface '00*, pages 213–220. Lawrence Erlbaum Associates, 2000.

[Mir96] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, 1996.

[Mir97] B. Mirtich. Efficient Algorithms for Two-Phase Collision Detection. Technical Report TR-97-23, Mitsubishi Electric Research Laboratory, 1997.

[Mir00] B. Mirtich. Timewarp Rigid Body Simulation. In *Proc. SIGGRAPH '00*, pages 193–200. ACM Press, 2000.

[MKE03] J. Mezger, S. Kimmerle, and O. Etzmuss. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2) :322–329, 2003.

[MOK95] K. Myszkowski, O. Okunev, and T. Kunii. Fast Collision Detection Between Complex Solids Using Rasterizing Graphics Hardware. *The Visual Computer*, 11(9) :497–512, 1995.

[Moo65] G. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8), 1965.

[MS01] V. J. Milenkovic and H. Schmidl. Optimization-based Animation. In *Proc. SIGGRAPH '01*, pages 37–46. ACM Press, 2001.

[MT97] T. Möller and B. Trumbore. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphical Tools*, 2(1) :21–28, 1997.

[MW88] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. In *Proc. SIGGRAPH '88*, pages 289–298. ACM Press, 1988.

[NMK+05] A. Nealen, M. Muller, R. Keiser, E. Boxerman, and M. Carlson. Physically Based Deformable Models in Computer Graphics. In *Eurographics State of the Art Reports*. Blackwell Publishing, 2005.

[Owe05] J. Owens. Streaming Architecture and Technology Trends. In *GPU Gems 2*, pages 457–470. Addison-Wesley, 2005.

[Pai02] D.K. Pai. STRANDS : Interactive Simulation of Thin Solids Using Cosserat Models. *Computer Graphics Forum*, 21(3) :347–352, 2002.

[PB88] J. Platt and A. Barr. Constraints Methods for Flexible Models. In *Proc. SIGGRAPH '88*, pages 279–88. ACM Press, 1988.

[PCP01] E. Plante, M.-P. Cani, and P. Poulin. A Layered Wisp Model for Simulating Interactions Inside Long Hair. In *Proc. Eurographics Workshop on Computer Animation and Simulation*, pages 139–148. Springer, 2001.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, 1992.

[PG95] I. J. Palmer and R. L. Grimsdale. Collision Detection for Animation Using Sphere-Trees. *Computer Graphics Forum*, 14(2) :105–116, June 1995.

[PKA+05] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, and L. J. Guibas. Meshless Animation of Fracturing Solids. *ACM Transactions on Graphics*, 24(3) :957–964, 2005.

[Pro95] X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigide Cloth Behavior. In *Proc. Graphics Interface '95*, pages 147–154. AK Peters Ltd., 1995.

[Pro97] X. Provot. Collision and Self-Collision Handling in Cloth model Dedicated to Design Garments. In *Proc. Eurographics Workship on Animation and Simulation*. Blackwell Publishing, 1997.

[PS] PCI-SIG. PCIExpress ® Specifications. PCI-SIG. Available at : `http ://www.pcisig.com/specifications/pciexpress/`.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[RCFC03] L. Raghupathi, V. Cantin, F. Faure, and M.-P. Cani. Real-time Simulation of Self-collisions for Virtual Intestinal Surgery. In *Surgery Simulation & Soft Tissue Modeling*, pages 15–26. Springer, 2003.

[Red04] S. Redon. Continuous Collision Detection for Rigid and Articulated Bodies. In *Proc. ACM SIGGRAPH Course Notes*. ACM Press, 2004.

[RF06] L. Raghupathi and F. Faure. QP-Collide : A New Approach to Collision Treatment. In *Journées du Groupe de Travail Animation et Simulation (GTAS)*, pages 91–101. Institut de Recherche en Informatique de Toulouse, June 2006. Available at : `http ://www-evasion.inrialpes.fr/Publications/2006/RF06`.

[RGF+04] L. Raghupathi, L. Grisoni, F. Faure, D. Marchall, M.-P. Cani, and C. Chaillou. An Intestine Surgery Simulator : Real-Time Collision Processing and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 10(6) :708–718, Nov/Dec 2004.

[RKC00] S. Redon, A. Kheddar, and S. Coquillart. An Algebraic Solution to the Problem of Collision Detection for Rigid Polyhedral Objects. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3722–38. IEEE Press, 2000.

[RKC01] S. Redon, A. Kheddar, and S. Coquillart. CONTACT : Arbitrary In-between Motions for Continuous Collision Detection. In *Proc. IEEE International Workshop on Robot and Human Interactive Communication*, pages 106–111. IEEE Press, 2001.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Fast Continuous Collision Detection Between Rigid Bodies. In *Proc. Eurographics '02*, pages 278–288. Blackwell Publishing, 2002.

[RL85] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 17–31, Austin, Texas, May28–31 1985.

[SB95] S. Sgambati and G. Ballantyne. Minimally Invasive Surgery for Diseases of the Colon and Rectum : The Legacy of an Ancient Tradition. In R. Jager and S. Wexner, editors, *Laparoscopic Colectomy*, pages 13–23. Churchill & Livingstone, NY, 1995.

[Sch90] J. Schwarze. Cubic and Quartic Roots. In A. Glassner, editor, *Graphics Gems*, pages 404–407. Academic Press, NY, 1990.

[SF91] M. Shinya and M. Forgue. Interference Detection Through Rasterization. *Journal of Visualization and Computer Animation*, 2 :132–134, 1991.

[SGHS98] J. Shade, S. Gortler, L-W. He, and R. Szeliski. Layered Depth Images. In *Proc. SIGGRAPH '98*, pages 231–242. ACM Press, 1998.

[SHB01] D. N. Metaxas S. Huh and N. I. Badler. Collision Resolutions in Cloth Simulation. In *Proc. Computer Animation '01*, pages 122–127. IEEE CS Press, 2001.

[Sny92] J. Snyder. Interval Analysis for Computer Graphics. In *Proc. SIGGRAPH '92*, pages 121–30. ACM Press, 1992.

[SSW95] E. Schömer, J. Sellen, and M. Welsch. Exact Geometric Collision Detection. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 211–216, 1995.

[SWF+93] J. M. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, and A. H. Barr. Interval Methods for Multi-point Collisions Between Time-dependent Curved Surfaces. In *Proc. SIGGRAPH '93*, pages 321–334. ACM Press, 1993.

[TF88a] D. Terzopoulos and K. Fleischer. Deformable Models. *The Visual Computer*, 4 :306–331, 1988.

[TF88b] D. Terzopoulos and K. Fleischer. Modeling Inelastic Deformation : Viscoelasticity, Plasticity, Fracture. In *Proc. SIGGRAPH '88*, pages 269–278. ACM Press, 1988.

[THM+03] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proc. Vision, Modeling, Visualization '03*, pages 47–54, 2003.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1) :61–81, March 2005.

[TPBF87] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically Deformable Models. *Proc. SIGGRAPH'87*, 21(4) :205–214, 1987.

[Tur90] G. Turk. Interactive Collision Detection for Molecular Graphics. Technical Report TR90-014, University of North Carolina at Chapel Hill, 1990.

[US97] S. Uno and M. Slater. The Sensitivity of Presence to Collision Response. In *Proc. IEEE VRAI Symposium*, pages 95–103. IEEE CS Press, 1997.

[VCMT95] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects. In *Proc. SIGGRAPH '95*, pages 137–144. ACM Press, 1995.

[vdB97] G. van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 2(4) :1–14, 1997.

[VMT94] P. Volino and N. Magnenat-Thalmann. Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity. *Computer Graphics Forum*, 13(3) :155–166, 1994.

[VMT95] P. Volino and N. Magnenat-Thalmann. Collision and Self-Collision Detection : Efficient and Robust Solutions for Highly Deformable Surfaces. In *Eurographics Workshop on Animation and Simulation*, pages 55–65. Springer-Verlag, 1995.

[VMT00] P. Volino and N. Magnenat-Thalmann. Accurate Collision Response on Polygonal Meshes. In *Proc. Computer Animation '00*, pages 154–163. IEEE CS Press, 2000.

[VMT01] P. Volino and N. Magnenat-Thalmann. Comparing Efficiency of Integration Methods for Cloth Simulation. In *Proc. Computer Graphics International '01*, pages 265–274. IEEE CS Press, 2001.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast Cloth Animation on Walking Avatars. In *Proc. Eurographics '01*, pages 260–267. Blackwell Publishing, 2001.

[WBK+06] K. Ward, F. Bertails, T.-Y. Kim, S. Marschner, M.-P. Cani, and M. Lin. A Survey on Hair Modeling : Styling, Simulation and Rendering. *IEEE Transcations on Visualization and Computer Graphics*, 2006. To be published.

[WW90] A. Witkin and W. Welch. Fast Animation and Control of Non-Rigid Structures. In *Proc. SIGGRAPH '90*, pages 243– 252. ACM Press, 1990.

[ZL03] G. Zachmann and E. Langetepe. Geometric Data Structures for Computer Graphics. In *Proc. ACM SIGGRAPH Tutorials*, pages 27–31. ACM Press, 2003.

[ZTK+05] G. Zachmann, M. Teschner, S. Kimmerle, B. Heidelberger, L. Raghupathi, and A. Fuhrmann. Real-Time Collision Detection for Dynamic Virtual Environments. In *Proc. IEEE Virtual Reality Tutorials*. IEEE CS Press, March 2005.

[ZWF+03] Y. Zhao, X. Wei, Z. Fan, A. Kaufman, and H. Qin. Voxels on Fire. In *Proc. IEEE Visualization '03*, pages 271–278. IEEE CS Press, 2003.

[ZY00] D. Zhang and M. Yuen. Collision Detection for Clothed Human Animation. In *Proc. Pacific Conference on Computer Graphics and Applications '00*, pages 328–337. IEEE CS Press, 2000.

Vers une Animation Robuste d' Objets Complexes en Interaction

Mots clés : synthèse d' image, animation par modèle physique, traitement de collision et contact.

Résumé

La traitement robuste des collisions et des contacts est important dans les scénarios d'animation et de simulation par modèle physique. Un problème important est comment traiter les collisions et les auto-collisions simultanées se produisant entre les objets dans la scène. Nous présentons de nouvelles méthodes pour la simulation robuste des objets déformables complexes. Notre technique se base sur la détection continue et la détection discrète des collisions. Nous proposons deux approches pour traiter les collisions une fois qu'elles sont détectées. D'abord, nous modélisons les collisions comme des contraintes linéaires adjointes aux équations de la dynamique, ce qui aboutit à un problème de programmation quadratique (QP). À cet effet, nous avons développé un algorithme qui traite les équations dynamiques et les contraintes linéaires d'une manière globale. En second lieu, nous avons développé une méthode utilisant des pénalités qui traite les collisions multiples en exploitant la cohérence temporelle. Le but est de garantir un état final sans collisions tout en réduisant au minimum la création "des collisions secondaires". Nous présentons nos résultats et analysons les avantages et la pertinence de notre méthode vis-à-vis des méthodes existantes. Les applications incluent les simulations mécaniques et chirurgicales, ainsi que les jeux vidéo et les effets spéciaux pour le cinéma.

Towards Robust Animation of Complex Objects in Interaction

Keywords : image synthesis, physically-based animation, collision and contact treatment.

Abstract

The robust handling of collisions and contacts is important in physics-based animation and simulation scenarios. A major problem is how to handle the simultaneous collisions and auto-collisions occurring between objects in the scene. We present new methods for the robust simulation of complex deformable objects. Our basic technique consists of detecting collisions both in continuous and discrete time steps. We propose two approaches for handling the collisions once they are detected. First, we conceptualize the collisions as linear constraints and the dynamics equations as an objective function to be minimized thus formulating it as a quadratic programming (QP) problem. To this effect, we have developed a novel integrated QP solver taking into account of both the underlying dynamics and the simultaneous multiple collisions. Second, we have have developed a spring-based method which handles multiple collisions by exploiting temporal coherence. Our approach is expected to guarantee a collision-free final state while minimizing the creation of "secondary collisions". We present our results and comprehensively analyze the advantages and relevance of our method vis-a-vis the existing methods. Applications include mechanical and surgery simulations, computer games and motion picture special effects.