QP-Collide: A New Approach to Collision Treatment

Laks Raghupathi
François Faure

EVASION/GRAVIR, INRIA Rhône-Alpes
655, Avenue de l' Europe, 38334, Montbonnot
laks@imag.fr francois.faure@imag.fr

Abstract

Robust handling of collisions and contacts is important in physics-based animation and simulation scenarios. We 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. In this paper, we describe in detail on how such an approach can be integrated within an existing dynamics simulation environment. In addition, we also include implementation difficulties of this approach and discuss practical tricks to overcome the same.

1 Introduction

The robust treatment of collisions and contacts finds an important place for realistic dynamics simulation in virtual reality [PNTSD96], motion picture special effects [BFA02], surgery simulation [RGF$^+$04], computer games and other graphics applications. A major problem in physically-based animation is how to handle simultaneous collisions occurring between objects in the scene and the self-collisions within each object itself. In this paper we present a novel approach for treating multiple collisions and contacts.

2 Background and Motivation

For a 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]. But we would like to make a note on detecting collisions at discrete time intervals and continuous collision detection. Most applications detect collisions at discrete time intervals - i.e., we check for object intersections based on their current state (positions, etc.). However such an approach may miss collisions in thin or fast-moving objects or applications requiring large time-step simulations. In such cases, it is worth detecting the collisions by checking the trajectory of the objects in question to see if the objects crossed *between* the time intervals. A good discussion on continuous collision detection techniques is presented in [MC00] and [RKC02].

In this paper, we shall rather concentrate on the techniques for handling the collisions once they are detected. Among existing techniques, a first approach [Pro97] is to use only impulse-based collision response by instantaneous correcting the displacements and velocities corrections. This method while being more accurate nevertheless adds undesirable extra energy and strain into the system and may also provoke new collisions thus warranting additional treatment. The second approach [VT00] is by detecting the collisions at continuous intervals and computing the forces needed to correct the accelerations, velocities and positions of the present, next and next-to-next time steps. By directly manipulating the positions, the above two methods may introduce large amounts of strain and energy into the system. So, a third method [BFA02] was proposed which 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 approach while being the state-of-the-art is complicated requiring 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.

3 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 approaches 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]:

$$\left(\mathbf{M} - dt^2\frac{\partial \mathbf{f}}{\partial \mathbf{x}} - dt\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right)\Delta\mathbf{v} = dt\left(\mathbf{f} + dt\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\mathbf{v}\right) \tag{1}$$

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

We usually then perform the collision detection and apply the correction instantaneously to the positions and velocities (impulses) or apply stiff spring forces in the next time step (penalty forces). 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 solution to the problem in hand. Let us describe the method in detail:

First, we write the implicit equation (1) in a concise form:

$$\mathbf{K}\Delta\mathbf{v} = \mathbf{b} \tag{2}$$

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

$$q(\Delta\mathbf{v}) \equiv \frac{1}{2}\Delta\mathbf{v}^T\mathbf{K}\Delta\mathbf{v} - \mathbf{b}^T\Delta\mathbf{v} \tag{3}$$

subject to the constraints:

$$\mathbf{J}\Delta\mathbf{v} \leq \mathbf{c} \tag{4}$$

where $\mathbf{J}$ and $\mathbf{c}$ are the constraint matrix and values respectively (cf. §4 on how to compute the constraints). If (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, (4) is reduced to:

$$\mathbf{J}_i\Delta\mathbf{v}_i = \mathbf{c}_i, i \in \mathcal{A} \tag{5}$$

From (3) and (5), we write the Lagrangian function as:

$$\mathcal{L}(\Delta\mathbf{v}, \lambda) = \frac{1}{2}\Delta\mathbf{v}^T\mathbf{K}\Delta\mathbf{v} - \mathbf{b}^T\Delta\mathbf{v} - \lambda^T(\mathbf{J}\Delta\mathbf{v} - \mathbf{c}) \tag{6}$$

$$\begin{aligned} \nabla_{\Delta\mathbf{v}}\mathcal{L} = \mathbf{0} \quad &\Rightarrow \quad \mathbf{K}\Delta\mathbf{v} - \mathbf{b} - \mathbf{J}^T\lambda = 0 \\ \nabla_{\lambda}\mathcal{L} = \mathbf{0} \quad &\Rightarrow \quad \mathbf{J}\Delta\mathbf{v} - \mathbf{c} = 0 \end{aligned} \tag{7}$$

which can be rearranged as:

$$\begin{bmatrix} \mathbf{K} & -\mathbf{J}^T \\ -\mathbf{J} & \mathbf{0} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{v} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{c} \end{pmatrix} \tag{8}$$

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

$$\lambda_i \geq 0, i \in \mathcal{A} \tag{9}$$

$$\mathbf{J}_k\Delta\mathbf{v}_k \leq \mathbf{c}_k, k \notin \mathcal{A} \tag{10}$$

At the end of each iteration, if (9) is not satisfied, the corresponding constraint J_i is removed from the active set $\mathcal{A}$ and is put into $\mathcal{A}'$. Viceversa if a non-active constraint in (10) is violated, then the same is transfered 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 §6.

4 On Modeling the Constraints

4.1 Collision Constraints

Here, we assume that the reader is well-versed in applying any one of the suitable collision detection techniques. We encourage the interested readers to go through the survey papers presented in §2. However, we would like to make a brief point on detecting collisions at discrete and continuous time intervals. The former approach which is quite the norm checks for collisions at each time step, $\mathbf{x}(t), \mathbf{x}(t+dt)$, etc. The latter on the other hand detects collisions *between* the time steps checking for intersections in the trajectory between $\mathbf{x}(t)$ and $\mathbf{x}(t+dt)$, by taking into account of the intermediate velocity $\Delta\mathbf{v}(t+dt)$.

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.

4.2 Vertex-Triangle Collision

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 $\hat{\mathbf{n}}$. The above are the instantaneous values at time t. Our objective is to find the new $\Delta\mathbf{v}$ at time $t+dt$. 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:

$$(\mathbf{v}_{ij} + \Delta\mathbf{v}_{ij}) \cdot \hat{\mathbf{n}} \; dt \leq g - (r_i + r_j) \tag{11}$$

It can be rearranged as:

$$(w_0\Delta\mathbf{v}_{j0} + w_1\Delta\mathbf{v}_{j1} + w_2\Delta\mathbf{v}_{j2} - \Delta\mathbf{v}_i).\hat{\mathbf{n}} \leq c \tag{12}$$

where $c = (g - (r_i + r_j))/dt + (w_0\mathbf{v}_{j0} + w_1\mathbf{v}_{j1} + w_2\mathbf{v}_{j2} - \mathbf{v}_i) \cdot \hat{\mathbf{n}}$. Each such constraint corresponds to one row of the $\mathbf{J}$ matrix of (4). The structure of the k^{th} row $\mathbf{J}_k$ will then look as follows:

$$\begin{pmatrix} \cdots & & j_0 & j_1 & j_2 & & \cdots & & i & & \cdots \\ \cdots & 0 & w_0\hat{\mathbf{n}} & w_1\hat{\mathbf{n}} & w_2\hat{\mathbf{n}} & \cdots & 0 & \cdots & \hat{\mathbf{n}} & 0 & \cdots \end{pmatrix} \tag{13}$$

4.3 Edge-Edge Collision

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). 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 equations constraining the relative velocity is same as (11). It can be further elaborated as:

$$((1-b)\Delta\mathbf{v}_{j0} + b\Delta\mathbf{v}_{j1} - (1-a)\Delta\mathbf{v}_{i0} - a\Delta\mathbf{v}_{i1}) \cdot \hat{\mathbf{n}} \leq c \tag{14}$$

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

$$\begin{array}{ccccccccc} \cdots & j_0 & j_1 & \cdot & \cdot & \cdot & i_0 & i_1 & \cdots \\ \end{array}$$
$$\begin{pmatrix} 0 & (1-b)\hat{\mathbf{n}} & b\hat{\mathbf{n}} & \cdots & 0 & \cdots & -(1-a)\hat{\mathbf{n}} & -a\hat{\mathbf{n}} & 0\cdots \end{pmatrix} \tag{15}$$

4.4 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 the iteration and is thus treated as an equality all along.

4.5 Strain Constraints

Some times we would like to control the strain values during the simulation typically to avoid excessive elongation or compression of spring. 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 restlength 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), this can be written as:

$$\parallel \mathbf{x}_i + dt(\mathbf{v}_i + \Delta\mathbf{v}_i) \ - \mathbf{x}_j + dt(\mathbf{v}_j - \Delta\mathbf{v}_j) \parallel < l_{\max} \tag{16}$$

The equation can be linearized with first order approximation and can be expressed as:

$$(\Delta\mathbf{v}_i - \Delta\mathbf{v}_j) \cdot \hat{\mathbf{n}} \leq l(l_{\max} - l)/dt \tag{17}$$

where $\hat{\mathbf{n}}$ is the normalized direction of the spring elongation $(\mathbf{x}_i - \mathbf{x}_j)/ \parallel \mathbf{x}_i - \mathbf{x}_j \parallel$. A similar equation can be written so that the spring does not compress below a certain length $l_{\min}$.

5 On Using the Conjugate Gradient Algorithm

5.1 Without Inverting K

We generally assume that the $\mathbf{K}$ matrix of (3) is symmetric and positive-definite. Hence the composite matrix $\mathbf{A}$ in (3) is also symmetric and can be solved with a standard *Conjugate Gradient* algorithm [PFTV92]. We also would prefer $\mathbf{A}$ 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 §7.1. Otherwise, both the $\mathbf{K}$ and $\mathbf{J}$ 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 $\mathbf{A}$ matrix. Assuming that there are n equations and m active constraints, the multiplication operators required for computing the left hand side of (8) are:

$$\mathbf{y}_i = \mathbf{K} \cdot \Delta\mathbf{v} - \mathbf{J}^T \cdot \lambda, i \in [1, \cdots, n] \tag{18}$$

$$\mathbf{y}_i = -\mathbf{J} \cdot \Delta\mathbf{v}, i \in [n+1, \cdots, n+m] \tag{19}$$

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

5.2 Inverting K

In certain cases, it is possible quickly invert $\mathbf{K}$ matrix using techniques such as LU inversion. When $\mathbf{K}^{-1}$ exists, from (8) we find the solution λ and $\Delta\mathbf{v}$ as follows:

$$\mathbf{J}\mathbf{K}^{-1}\mathbf{J}^T\lambda = \mathbf{c} - \mathbf{J}\mathbf{K}^{-1}\mathbf{b} \tag{20}$$

$$\Delta\mathbf{v} = \mathbf{K}^{-1}(\mathbf{b} + \mathbf{J}^T\lambda) \tag{21}$$

(20) reduces the maximum CG iterations to m.

6 Overall Algorithm

The overall algorithm is described here:

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

Algorithm 1 QP Collide

1: Solve (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 (8)
14: **for all** $\lambda_q^{(k)}$, $q \in \mathcal{A}$ **do**
15:     **if** $\lambda_q^{(k)} \geq 0$ **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 $\mathbf{v}(t + dt) \leftarrow \mathbf{v}(t) + \Delta\mathbf{v}^*$
34: Compute final position $\mathbf{x}(t + dt) \leftarrow \mathbf{x}(t) + dt \cdot \mathbf{v}(t + dt)$

cause numerical errors while computing the solution to the conjugate gradient algorithm. We propose the following techniques to tackle against commonly occurring numerical problems.

7.1 Linearly Dependant Constraints

There can be certain 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 (Fig. 1). 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 corresponding rows of $\mathbf{J}$ will be:

$$\begin{pmatrix} \cdots & 0 & n_x & n_y & n_z & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & n_x & n_y & n_z & 0 & \cdots \\ \cdots & 0 & a_1 n_x & a_1 n_y & a_1 n_z & a_2 n_x & a_2 n_y & a_2 n_z & 0 & \cdots \end{pmatrix} \tag{22}$$

This is a common occurrence in mechanical simulations and creates a singular 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 $\mathbf{A}$ with a 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:

$$\mathbf{A} = \begin{bmatrix} \mathbf{K} & -\mathbf{J}^T \\ -\mathbf{J} & \mathbf{L} \end{bmatrix} \tag{23}$$

The corresponding matrix-vector multiplication routine in (19) is appropriately modified as:

$$\mathbf{y}_i = -\mathbf{J} \cdot \Delta\mathbf{v} - \mathbf{L} \cdot \lambda, i \in [n+1, \cdots, n+m] \tag{24}$$

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

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 $\mathcal{A}'$ causing the QP iteration in an endless 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).

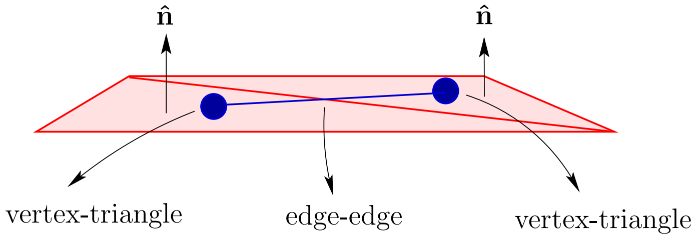
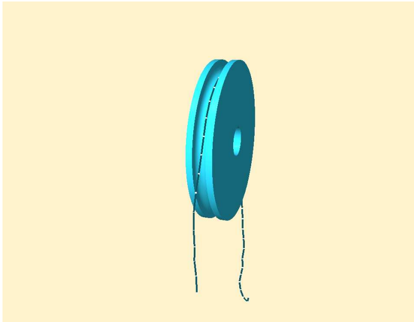


Figure 1: A commonly occurring collision scenario which results in linearly dependant constraints

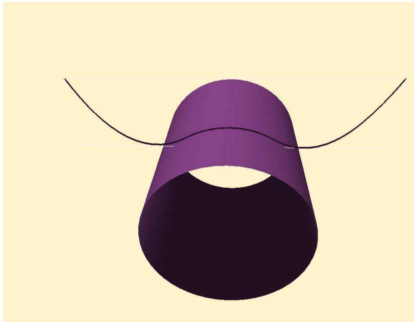
8 Results & Conclusions

We have presented an elegant and novel solution for simultaneously treating multiple collisions and contacts. We now present some results of our algorithm (Fig. 2). We note that the algorithm handles both the collisions and fixed constraints well for the case of a mechanical cable. More experiments are needed to validate this method for other complex scenarios such as cloth, deformable organs, etc.

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 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 will not handle the case of multiple collisions such as a cloth colliding with itself - we might need go through multiple QP passes which is computationally expensive in order to deal with that. Nonetheless, we hope that this method be further explored to solve the problem in hand.



(a)



(b)

Figure 2: Snapshots of simulation using our algorithm. A stiff cable (a) falling off a pulley and (b) suspended (constrained at the end points) over a cylinder

References

[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.

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

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

[GMS02] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. on Optimization*, 12(4):979–1006, 2002.

[JTT01] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers and Graphics*, 25(2):269–285, April 2001.

[LG98] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proc. IMA Conference on Mathematics of Surfaces*, pages 33–52, 1998.

[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.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C. The Art of Scientific Computing*, pages 83–89. Cambridge University Press, 1992.

[PNTSD96] P.Volino, N.Magnenat-Thalmann, S.Jianhua, and D.Thalmann. The evolution of a 3d system for simulating deformable clothes on virtual actors. *IEEE Comp. Graph. & Appl.*, 16(5):42–50, 1996.

[Pro97] X. Provot. Collision and self collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation '97*, pages 177–189, 1997.

[RGF+04] L. Raghupathi, L. Grisoni, F. Faure, D. Marchal, M.-P. Cani, and C. Chaillou. An intestinal surgery simulator: Real-time collision processing and visualization. *IEEE Trans. Vis. Comput. Graph.*, 10(6):708–718, 2004.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Proc. Eurographics '02*, 2002.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, March 2005.

[VT00] P. Volino and N. M-. Thalmann. Accurate collision response on polygonal meshes. In *Proc. Computer Animation*, pages 154–163. IEEE Computer Society, 2000.