A Parallel Block Iterative Method for Interactive Contacting Rigid Multibody Simulations on Multicore PCs

Claude Lacoursière

HPC2N/VRlab and Department of Computing Science,
Umeå University, SE-901 87 Umeå, Sweden
claude@cs.umu.se
http://www.cs.umu.se/~claude

Abstract. A hybrid, asynchronous, block parallel method to approximately solve complementarity problems (CPs) in real-time on multicore CPUs is described. These problems arise from interactive real-time simulations of systems of constrained, contacting rigid bodies, which are useful in virtual operator training systems for instance. A graph analysis phase identifies components which are weakly coupled using simple heuristics. Each component is then solved in parallel using either a block principal pivot or a projected block Gauss-Seidel method running in separate threads. Couplings which generate forces between the subsystems are handled iteratively using a Gauss-Seidel process which communicates updates between the interacting subsystems asynchronously. Preliminary results show that this approach delivers good performance while keeping overhead small.

1 Introduction

Interactive real-time simulation of systems of constrained rigid bodies and particles is an essential component of several commercially relevant applications, such as virtual environment heavy machinery operator training systems. Numerical integration of multibody systems needed for these simulations involves solution of systems of equations with several hundred variables. The best known strategy to simulate frictional contacts—ubiquitous in real-life systems—requires solving mixed linear complementarity problems (MLCP)s as described below. In the real-time interactive simulation context, solutions of these MLCPs must be delivered in less than 10ms to keep screen refresh rates above 60 frames per second. Direct solution methods for MLCP are slow for large systems because they rely on single pivot operations [3] for which sparsity exploitation is difficult. For this reason, iterative methods based on pairwise interatcions—which are fast but much less accurate—have dominated the 3D graphics and interactive games literature [8] [4]. Parallelization has also been generally overlooked.

The present proposes a parallel hybrid solution method which uses either a direct or an iterative solver on blocks of strongly coupled equations and Gauss-Seidel (GS) iterations to approximate the coupling forces between those blocks.

B. Kågström et al. (Eds.): PARA 2006, LNCS 4699, pp. 956–965, 2007.

This scheme is easily implemented on multicore PCs using simple threads and synchronization. The test application consists of a wheeled vehicle toting a long tether. Such vehicles are used for various remote operations from de-mining to underwater inspections. Virtual training systems have proved useful and commercially relevant. The proposed hybrid scheme delivers good performance on this example.

The rest of the article is organized as follows. Section 2 briefly introduces the problem being solved in the context of interactive simulations. Section 3 presents the splitting strategy as well as the iterative algorithm which computes approximate coupling forces between subsystems. Section 4 covers heuristics used to split a physical system into interacting blocks, and Section 5 describes the asynchronous execution framework implementing the GS scheme to compute the approximate coupling forces. Results and conclusions are presented in Section 6 in which a simple system is analyzed to provide preliminary data. The present report is part of work in progress.

2 Problem Formulation

Consider a general mechanical system with n-dimensional generalized velocity vector $v \in \mathbb{R}^n$. The system has a square, real, positive definite and block diagonal $n \times n$ mass matrix M, with easily computed inverse $U = M^{-1}$, and is subject to a variety of constraints which have Jacobian matrix G of size $m \times n$. The essential computation of a large family of stepping schemes involves the solution of a *mixed linear complementarity problem* (MLCP):

$$\begin{gathered} Sy + q = w = w^{(+)} - w^{(-)} \\ 0 \le y - l \perp w^{(+)} \ge 0, \qquad 0 \le u - y \perp w^{(-)} \ge 0, \end{gathered} \tag{1}$$

at each time step. The real, square $m \times m$ matrix S has the form $S = GUG^T$. In addition, $q \in \mathbb{R}^m$ is a real vector, $w^{(+)}, w^{(-)}$ are the real positive and negative components of the residual vector $w \in \mathbb{R}^m$, respectively. The lower and upper bound vectors $l, u \in \bar{\mathbb{R}}^m$, with $\bar{\mathbb{R}} = \mathbb{R} + \{\pm\infty\}$, are extended real vectors. The inequality and orthogonality signs in (1) are understood componentwise so that $a, b \in \mathbb{R}^m$, $a \ge b$ means $a_i \ge b_i, i \in \{1, 2, \ldots, m\}$, $a \le b$ means $a_i \le b_i, i \in \{1, 2, \ldots, m\}$, and $a \perp b$ means $a_i b_i = 0$ for all $i \in \{1, 2, \ldots, m\}$, whenever $a, b \ge 0$. This problem of solving (1) is abbreviated as MLCP(S, q, l, u). The solution of this MLCP is the vector $y \in \mathbb{R}^m, y = \mathrm{SOL}(\mathrm{MLCP}(S, q, l, u))$, which is unique as long as S is symmetric and positive definite [2]. The solution vector y produces the constraint force vector $G^T y$ from which the updated velocities and coordinates can be computed directly given a time-integration strategy.

For jointed mechanical systems, Jacobians have the simple block structure

$$G^T = \left[{G^{(1)}}^T \; {G^{(2)}}^T \; \ldots \; {G^{(n_c)}}^T \right], \tag{2}$$

where each block row $G^{(i)}$ is of size $n_i \times n$, with $\sum_i n_i = m$. The integer n_c is the number of constraints. Each block row $G^{(i)}$ contains only a few non-zero column blocks, $c_i(1), c_i(2), \ldots, c_i(p_i)$ where p_i is usually 1 or 2, as is the

case for common joints and contact constraints. In this format, each column block usually corresponds to a single physical body. This produces the first level of partitioning of the system where the velocity vector v is decomposed into blocks $v^{(i)}, i = 1, 2, \ldots, n_b$, where n_b is the number of *generalized bodies* in the system, which may or may not have the same dimensionality. Each block here corresponds to a given physical body.

The connectivity structure at this level of partitioning is a bipartite graph, where nodes are either physical bodies or constraints. Any common graph traversal algorithm can identify the connected components. Processing these is an embarrassingly parallel problem.

The solution of MLCP (1) for each connected component can be performed using, e.g., a block pivot [3] (BP) method, which is equivalent to applying the Newton-Raphson method on the nonsmooth formulation, $\mathrm{mid}((y - l)_i, (Sy + q)_i, (y - u)_i) = 0$, where $\mathrm{mid}(a, b, c)$ is the midpoint function [6], defined component-wise as:

$$\mathrm{mid}(a, b, c) = a + c - \sqrt{(a - b)^2} + \sqrt{(b - c)^2}. \tag{3}$$

This is called a direct solver since it can produce the exact solution with appropriate smoothing strategy [6]. Other possibilities include iterative solvers such as projected Gauss-Seidel (PGS) methods or projected Successive Overrelaxation (PSOR) [3] methods which are extremely simple to implement and quickly yield answers of moderate accuracy. The BP, PGS and PSOR can all be warm started from an approximate solution and they typically decrease the infeasibility at each stage. Warm starting is not possible for direct pivoting methods such as the Lemke, Cottle-Dantzig, or Keller algorithms [3]. In addition, direct pivoting methods do not reduce infeasibility monotonically and cannot be stopped early to yield an approximate solution. Both features are key element for splitting methods in which slightly different MLCPs (1) with the same matrix S but slightly different vectors q, l and u, are solved numerous times.

The BP method without smoothing performs well on average but it can cycle over a set of candidates, none of which exactly solve the problem, when the problem is nearly degenerate or infeasible which is common in practice due to the use of a vertex based definition of contacts, large integration step, and *a posteriori* collision detection [5]. Degeneracy problems are not considered further here.

3 Splitting Strategy and MLCP Iterative Method

To parallelize further, we consider one of the connected components described in the previous section and assume a partitioning of the bodies into two groups for example, labeled as $1, 2$, having block velocity vectors and block inverse mass matrix $v^{(i)}, U_{ii}, i = 1, 2$, respectively. Any given constraint Jacobian can then be split into blocks: $G^{(i)} = [G_1^{(i)}, G_2^{(i)}]$, where block $G_j^{(i)}$ acts only on the block coordinates v_j, $j = 1, 2$. The natural separation of constraints produces three

groups, namely, those acting only on the first group, those acting only on the second group, and those acting on both groups of bodies

$$G = \begin{bmatrix} G_{11} & 0 \\ 0 & G_{22} \\ G_{31} & G_{32} \end{bmatrix}. \tag{4}$$

After splitting the mass matrix correspondingly, with $U = \mathrm{diag}(U_{ii})$, matrix S in (1) has the form

$$S = \begin{bmatrix} S_{11} & 0 & S_{31}^T \\ 0 & S_{22} & S_{32}^T \\ S_{31} & S_{32} & S_{33} \end{bmatrix}, \tag{5}$$

where $S_{3i} = G_{3i}U_{ii}G_{ii}^T, i = 1, 2$, and $S_{33} = \sum_{i=1}^{2} G_{3i}U_{ii}G_{3i}^T$. Obviously, if the constraints in group 3 are chosen to be mass orthogonal to those in groups 1 and 2, there are no off-diagonal terms. Minimizing these coupling terms is likely to improve the convergence rate and is subject of future work.

To formulate an iterative method for solving this problem, we need a definition for the infeasibility error $e \in \mathbb{R}^m$. Consider a candidate solution vector x and residual $w = Sx + q$. For each component i, first check that it is within range, i.e., $l_i < x_i < u_i$. If so, the residual should vanish so we set $e_i = w_i$. If the bounds are finite and either $x_i = l_i$ or $x_i = u_i$, we use $e_i = \min(x_i - l_i, w_i)$ or $e_i = \min(u_i - x_i, w_i)$, respectively. If x_i is out of bound, we use the midpoint function $e_i = \max(x_i - u_i, \min(w_i, z_i - l_i))$.

A projected block GS solution of MLCP(S, q, l, u) is formulated in Algorithm 1.

Algorithm 1. Serial block projected Gauss-Seidel for solving partitioned MLCP(S, q, l, u).

1: Set $\nu \leftarrow 1$, choose $\tau > 0$ and $\nu_m > 0$
2: Initialize $y_1^{(1)}, y_2^{(1)}, y_3^{(1)}$.
3: **repeat**
4: Solve: $y_1^{(\nu+1)} \leftarrow \mathrm{SOL}(\mathrm{MLCP}(S_{11}, q_1 + S_{31}^T y_3^{(\nu)}, l_1, u_1))$
5: Solve: $y_2^{(\nu+1)} \leftarrow \mathrm{SOL}(\mathrm{MLCP}(S_{22}, q_2 + S_{32}^T y_3^{(\nu)}, l_2, u_2)))$
6: Solve: $y_3^{(\nu+1)} \leftarrow \mathrm{SOL}(\mathrm{MLCP}(S_{33}, q_3 + S_{31} y_1^{(\nu+1)} + S_{32} y_2^{(\nu+1)}, l_3, u_3))$
7: $\nu \leftarrow \nu + 1$
8: Compute infeasibility $e^{(\nu)}$
9: **until** $\|e^{(\nu)}\| < \tau$ OR $\nu > \nu_m$

This GS process reduces the infeasibility monotonically for a positive definite matrix, even in the case of MLCP [3] but the convergence is linear at best and can be stationary in case the matrix S is degenerate or nearly so. This algorithm can be parallelized with or without synchronization (see [1] for descriptions of chaotic asynchronous schemes) by using threads so that solutions to each of the subsystems are computed in parallel. Overhead can be kept small by only having

mutex and condition variables to control block reading from and block writing to the shared data, namely, the kinematic variables in subsystems $1, 2$.

4 Heuristics for Splitting of Connected Components

Splitting of connected components into primary groups of bodies and constraints and secondary groups of constraints should be done so as to minimize the coupling between the primary groups. The theoretical and algorithmic treatment of this minimization problem merits further investigation. Only a simple heuristic has been implemented which is now described.

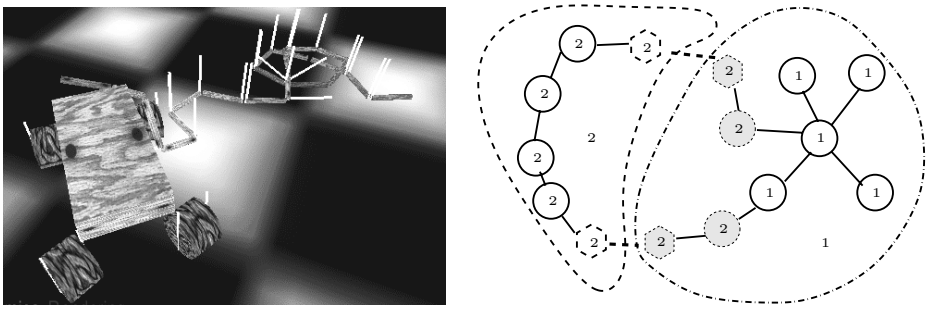
Each body is first labeled with a *grouping* identifier. It is expected that bodies with the same grouping index are interacting strongly with each other, at least when they are collected into the same connected component. This grouping index also carries a priority so that bodies in grouping g_1 are expected to be more important than those in grouping g_2 if $g_1 < g_2$.

The heuristic is to collect all connected components belonging to the same grouping id, g, as well as all bodies and constraints which are connected to bodies in g, have a grouping id higher than g, and are connected within depth k of a body with grouping index g. An illustration of this process is found in Fig. 1(b).

Connected components are constructed by performing a breadth first search in the rigid body system. After sorting the bodies in increasing order of grouping index, the first untouched body in that list is selected and from it, connected bodies are found by first listing attached constraints and then listing the bodies found for each constraint. If one of these bodies has a different grouping index, breadth first search is re-initiated at that body, going down to depth k. The bodies found during this search necessarily have a higher grouping index than current. Once this search terminates at a body with depth k, all the uncounted constraints attached to this body are added to the secondary partition pool containing all secondary constraints. This corresponds to the G_{3i} blocks in the previous section.

When this search terminates, we have a forest of primary partitions and a secondary partition with all coupling constraints. This secondary partition could be refined further but that is left for future work.

Consider the example of a tethered vehicle illustrated in Fig. 1. First assign grouping index one for the vehicle and two for the tether, and set the maximum depth to two. The partitioner will collect the main vehicle and the first two cable segments into the first partition. The imported bodies are filled in circles and octagons in Fig. 1(b). If the tether does not otherwise touch the vehicle or its wheels, the tether is in a primary partition by itself and the secondary partition contains a single constraint. If the vehicle rolls over the cable though, the first primary partition contains the main vehicle, the first two segments of the tether, and at least two more cable segments near each contact, symmetrically distributed about the contact location. This is illustrated in Fig. 1(b) where the end of the cable touches one of the vehicle parts. This introduces two more imported bodies in primary partition 1 illustrated with the gray bodies. This leads



(a) A still from a simulation of a tethered vehicle. Vertical white lines indicate normals at contact points.

(b) A sample graph and its partitioning.

Fig. 1. An example application involving a tethered vehicle. A still from the simulation is shown in (a) and a simplified graph analysis is shown in (b).

to four bodies, drawn with octagons instead of circles, which are influenced both by their primary partition and the forces computed by the secondary partition which contains the dashed constraints. The rest of the tether is then segmented and cut into two or more primary partitions, and the secondary partition contains two or more coupling constraints.

5 Asynchronous Execution

The shared variables in the system are the constraint forces applied on the rigid bodies. Indeed, some of the bodies are listed both in a primary partition (once only) and in secondary partitions (potentially several times).

When working on any given partition, a solver must first read the current net forces from the global data, solve for updated forces, and add the updated contributions back to the global data. To do this efficiently, the rigid body force data is segmented along the primary partitions. One mutex and one global time stamp is created for each of these, the stamp being incremented each time the data is updated by one of the parallel solver. Secondary partitions must acquire locks for all the primary partitions they are connected with but primary ones only acquire one lock.

In order to know if a solver should compute a new solution, local time stamps are kept for all the data sets used in a given partition. Reference counting the global time stamps allows to know if a data set is coupled to others.

Partitions and associated local data are packaged as a work unit and put in a priority queue so that partitions with lower iteration count have higher priority. A thread pool, designed much as in Ref [7], Ch. 3, is started during program initialization and each of the threads waits after a semaphore on which a thread manager broadcast when there is fresh data. Threads report upon completion to the manager via another semaphore by signaling.

The thread manager first configures all the data for a new solve process, fills the priority queue with the work units, and broadcast the workers in the thread pool. Each work thread then picks up a work item from the top of the priority queue and executes the solve operations on it. When this is done, a check is made to see whether or not that work unit needs more computations done to it and it is requeued accordingly. The work thread then picks up another work item from the top of the queue or goes idling if that is empty. Meanwhile, the thread manager listens for signals from the work threads and marshals them back to work as long as processing is not completed.

For each work unit, the program first checks if it is coupled to others and whether there is fresh data to consider by measuring the difference between local and global time stamps. If there is fresh data, locks are acquired, data is cached locally, locks are then released, and computations are performed. When the results are ready, the global data set is locked while the local data is added to it. Time stamps are then incremented and all locks are released. Local iteration count is then increased and the work unit is requeued. If there are no couplings when a work unit corresponding to a primary partition is processed, the final integration stage is performed using the available global data.

Work stops when a maximum iteration count is reached though a strategy based on global error would presumably be more accurate and potentially slower. The latter is future work.

6 Results and Conclusions

An implementation of Algorithm (1) was realized using Vortex, a commercial rigid multibody dynamics simulation library not originally designed to handle couplings between partitions (see `http://www.cm-labs.com`). The reference platform was Linux and the standard pthreads library. As a preliminary test, a simple example of a tethered car was constructed. This consists of a simplified vehicle with a chassis and four wheels, dragging a long cable made of rigid body sections attached with ball joints. A still of that simulation is shown in Fig. 1(a). The graphics rendering is minimal here because the main goal was to test the performance of the splitting scheme but a similar setup is used in a commercial trainer application for remotely operated tethered vehicles. Such an application demands hundreds of rigid bodies for the tether simulation because of the combination of long cable length and tortuous paths in typical scenarios.

The natural partitioning here is to explicitly assign the rigid bodies composing the vehicle in group 1 and those composing the tether in group 2. Further partitioning can be realized by segmenting the cable so that each segment contains 10 rigid bodies, say. With this strategy, we can then use a direct BP or iterative PGS solver on each primary partition, making the overall algorithm hybrid. Since the main vehicle becomes unstable when using the PGS solver, the direct solver is always used on that group. Therefore, without splitting, it is necessary to use a direct solver for the entire system and this gives the reference baseline for performance.

Simulations were run on an Athlon 64 X2 Dual Core Processor 3800+ with 512MB secondary cache, 2.0 GHz clock speed, and 2GB of RAM. The vehicle was set on a circular path so that it would eventually roll over the tether. Simulations ran for 10000 steps which corresponds to nearly 3 minutes of real time. The time step was set to $h = 1/60 = 0.01667$ which produces real-time for refresh rates of 60 frames per second, with one integration step per frame. For cases where splitting is used, we performed five (5) coupling iterations based on visual inspection of the simulation. Global error estimate was not computed and not used as a stopping criterion for these runs. This is left for future work. When the iterative PGS was used, the stopping criteria was an absolute local error of less than 10^{-3} or an iteration count exceeding 20—parameters selected by trial and error. The direct solver used here is a BP method [3] with protection against cycling and restart and the iterative solver is a PGS method. Both are part of the Vortex toolkits.

Results are collected in Table 1 where the direct solver was used on each subgroup, and Table 2 where the PGS solver was used for cable segments and the BP solver for the main vehicle. In both these tables, the first column contains the number of rigid body segment in the tether of the example illustrated in Fig. 1(a). The second column is the number of parallel threads. Only cases with 1 or 4 threads were constructed. The third column indicates how many different groups where used in the simulation. This number is two (2) if we only split the tether in a group and the vehicle in the other but we also show results of segmenting the cable into many more groups using segmenting. The next column is the percentage CPU utilization which goes near 100% on single CPU and remains less than 160% when both cores are used with four (4) parallel threads. This is because other sections of the software are not multithreaded. The fifth column is the average time used for solving the MLCP, isolated from the other computations and the sixth column is the speedup of the solver time as compared to the single thread, single group setting. The last two column contain the total time taken by the simulation for one frame and the speedup, also with respect to the single group, single thread configuration.

The first striking observation is that splitting alone produces a speedup of a factor of 10 to 30. This is explained by the fact that we are now only computing an approximation of the MLCP (1). In other words, time is saved by decreasing the accuracy of the result. That would not be possible using a direct method on the full problem for instance. Using four separate threads of execution and enough groupings to saturate the thread pool, the speedup is up to 40 for small systems, and 120 for large ones. This speedup is sufficient to make the application truly interactive, both in speed and stability.

Since using the iterative method on the entire system produces unsatisfactory results for the vehicle dynamics, the baseline in Table 2 is computed with two groups, namely, one for the tether and one for the vehicle, both in the same thread. A speedup of 1.5 is observed on average when using four (4) parallel threads of execution as opposed to just one. This is close to what one would expect since other components of the software take much time.

Table 1. Timing results using a direct solver for each group. The baseline refers to a single thread simulation with a simple group containing all bodies and constraints.

Length	Threads	Groups	% CPU	Solver Time	Solver Speedup	Total Time	Speedup
60	1	1	99	98.3	98.9	1.0	1.0
	1	2	99	7.8	8.2	12.6	12.1
	1	7	100	6.7	7.0	14.7	14.0
	4	7	137	2.1	2.3	46.2	42.4
80	1	1	99	67.7	68.2	1.0	1.0
	1	2	100	6.8	7.1	9.9	9.6
	1	9	100	5.7	6.0	11.9	11.3
	4	9	147	2.9	3.2	23.4	21.4
100	1	1	99	186.9	187.3	1.0	1.0
	1	2	99	7.6	8.0	24.6	23.4
	1	11	99	8.7	9.1	21.5	20.6
	4	11	157	6.9	7.4	26.9	25.3
120	1	1	99	393.2	393.8	1.0	1.0
	1	2	99	11.7	12.2	33.7	32.2
	1	13	99	11.7	12.2	33.7	32.2
	4	13	151	3.2	4.7	121.3	83.2
140	1	1	97	513.9	514.5	1.0	1.0
	1	15	100	11.2	11.8	46.1	43.8
	4	15	152	3.8	4.3	134.7	120.5

Table 2. Timing results using an iterative solver for each group. The baseline here refers to a single thread simulation with two groups, the cable being processed with the PGS and the vehicle with BP solver respectively.

Length	Threads	Groups	% CPU	Solver Time	Solver Speedup	Total Time	Speedup
60	1	2	99	5.6	5.8	1.0	1.0
	4	7	149	3.5	3.7	1.6	1.5
80	1	2	100	7.8	8.2	1.0	1.0
	4	9	158	5.2	5.5	1.5	1.5
100	1	2	99	7.9	8.3	1.0	1.0
	4	11	160	5.7	6.2	1.4	1.3
120	1	2	99	9.3	9.8	1.0	1.0
	4	13	158	5.2	5.6	1.8	1.7
140	1	2	100	10.6	11.0	1.0	1.0
	4	15	163	8.7	9.3	1.2	1.2

The conclusion we draw from the data is that combining problem splitting and a direct solver is attractive in comparison to the far less accurate GS method which, unsurprisingly, does not benefit significantly from the parallelization. Further work is necessary to assess the error margin of this strategy and is work in progress. Also part of future work is the investigation of more sophisticated

partitioning schemes based on spectral analysis as well as conjugate gradient-type methods for computing inter-partition couplings.

Acknowledgments

Constructive comments by Bo Kågström, Daniel Kressner and Kenneth Bodin on earlier versions of this paper, as well as cooperation from CMLabs is gratefully acknowledged. This research was conducted using the resources of High Performance Computing Center North (HPC2N), and supported in part by the "Objective 1 Norra Norrlands" EU grant VISTA awarded to VRlab at Umeå University, and by the *Swedish Foundation for Strategic Research* under the frame program grant SSF-A3 02:128.

References

1. Bai, Z., Huang, Y.: A class of asynchronous parallel multisplitting relaxation methods for the large sparse linear complementarity problems. Journal of Computational Mathematics 21(6), 773–790 (2003)
2. Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. In: Computer Science and Scientific Computing, Academic Press, New York (1992)
3. Júdice, J.J.: Algorithms for linear complementarity problems. In: Spedicato, E. (ed.) Algorithms for Continuous Optimization. NATO ASI Series C, Mathematical and Physical Sciences, Advanced Study Institute, vol. 434, pp. 435–475. Kluwer Academic Publishers, Dordrecht (1994)
4. Kaufman, D.M., Edmunds, T., Pai, D.K.: Fast frictional dynamics for rigid bodies. ACM Trans. Graph. 24(3), 946–956 (2005)
5. Lacoursière, C.: Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results. In: Ollila, M. (ed.) Conference Proceedings from SIGRAD2003, November 20–21, 2003, Umeå Unversity, Umeå, Sweden, pp. 11–16. SIGRAD (2003)
6. Li, D., Fukushima, M.: Smoothing Newton and quasi-Newton methods for mixed complementarity problems. Comp. Opt. and Appl. 17, 203–230 (2000)
7. Nichols, B., Buttlar, D., Farrell, J.P.: Pthreads Programming. O'Reilly and Associates, Inc., Sebastopol, CA 95472 (1996)
8. Weinstein, R., Teran, J., Fedkiw, R.: Dynamic simulation of articulated rigid bodies with contact and collision. IEEE TVCG 12(3), 365–374 (2006)