Efficient block pivoting for multibody simulations with contact

Andreas Enzenhöfer
McGill University and
CM Labs Simulations, Inc.
Montreal, Canada

Nicolas Lefebvre
École de technologie supérieure
Montreal, Canada

Sheldon Andrews
École de technologie supérieure
Montreal, Canada

Figure 1: Examples simulated using our efficient block pivoting method: mobile crane lifting a concrete pipe (top left); winch spooling a chain (top center); light armored vehicle driving over obstacles (top right); knuckle boom crane on a ship lifting a subsea module (bottom right); forklift lifting a crate and driving around cones (bottom center); forwarder picking up a log (bottom right). Each one is modeled as a multibody system with hundreds of constraints, stiff contacts, and large mass ratios.

ABSTRACT

Simulating stiff physical systems is a requirement for numerous computer graphics applications, such as VR training for heavy equipment operation. However, iterative linear solvers often perform poorly in such cases, and direct methods involving a factorization of the system matrix are typically preferred for accurate and stable simulations. This can have a detrimental impact on performance, since factorization of the system matrix is costly for complex simulations. In this paper, we present a method for efficiently solving linear systems of stiff physical systems involving contact, where the dynamics are modeled as a mixed linear complementarity problem (MLCP). Our approach is based on a block Bard-type algorithm that applies low-rank downdates to a Cholesky factorization of the system matrix at each pivoting step. Further performance improvements are realized by exploiting low bandwidth characteristics of the factorization. Our method gives up to 3.5× speed-up versus recomputing the factorization based on the index set. Various challenging scenarios are used to demonstrate the advantages of our approach.

CCS CONCEPTS

• **Computing methodologies** → **Real-time simulation**; *Physical simulation.*

KEYWORDS

physics simulation, pivoting, LCP, multibody dynamics, Cholesky

I3D '19, May 21–23, 2019, Montreal, QC, Canada

ACM Reference Format:
Andreas Enzenhöfer, Nicolas Lefebvre, and Sheldon Andrews. 2019. Efficient block pivoting for multibody simulations with contact. In *Symposium on Interactive 3D Graphics and Games (I3D '19), May 21–23, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3306131.3317019

1 INTRODUCTION

Physical simulation is a key aspect of many interactive computer graphics applications, such as videos games, virtual reality (VR) training, and hapto-visual simulations. Typically, the application designer must make decisions regarding numerical methods and physical models used for the simulation based on the required fidelity and performance. The trade-off between accuracy and computational performance is one of the central conflicts here, and compromises can often be attributed to the method used to solve equations for the underlying physical model.

In this paper, we are primarily motivated by constrained multibody simulations involving contact. Specifically, when the modeled physical system contains large mass ratios between bodies and high stiffness (low compliance) in the joints. Examples include heavy vehicle simulation and robotics. These simulations often result in a poorly conditioned linear systems, which in turn restricts the class of algorithms that may be used to solve the constrained equations of motion. Often, direct linear solvers involving a Cholesky or LU decomposition of the system matrix must be used in order to achieve reliable and stable simulations.

When a linear complementarity problem (LCP) is used to model frictional contacts between bodies, as is the case in many rigid body simulators, requirements are further complicated since these models call for a special class of solvers. Iterative fixed-point methods, such as the Gauss-Seidel and Jacobi algorithms, are popular choices for video game developers due to their ability to produce approximate solutions quickly. Principal pivoting methods, such as Lemke's algorithm and Bard-type algorithms, are more popular when accurate solutions are desired. These methods systematically estimate which variables are known and unknown, and then compute the LCP solution by solving a linear system based on a partitioning of variables. If a direct method is being used to solve the linear system, this means that the lead matrix must be formed and refactorized for each change to variable partitioning, which is costly.

Our proposed method significantly reduces the time required to solve constrained multibody dynamical simulations using a pivoting-based direct solvers. We specifically target interactive training simulations, such as the examples shown in the teaser image, and based our approach on observations of the index sets during typical scenarios. We identify the matrix factorization step as a bottle neck in algorithms where direct linear solvers are required, and develop a modified version of the block principal pivoting algorithm which makes efficient re-use of an initial Cholesky factorization. Further performance improvements are realized by using advanced data structures, specifically a *skyline* coding of the factorized matrix.

2 RELATED WORK

2.1 Constrained multibody dynamics

Simulation of multibody systems is commonplace for many modern computer graphics applications. If the system consists only of unilateral constraints, such as those used to model direct contact interactions between bodies, this naturally leads to a *linear complementarity problem* (LCP) formulation of the constrained dynamics [Baraff 1994; Moreau 1988]. However, simulations typically include both bilateral and unilateral constraints, as well as friction at the contacts and in the joints. Bilateral constraints model articulations between bodies, such as hinge and prismatic joints. This leads to the more general mixed linear complementarity problem (MLCP) formulation [Stewart and Trinkle 1996]. Unfortunately, the solution of an MCLP is more involved than solving a linear system and does not permit a direct application of linear solvers, and instead requires a special class of solvers.

The course material by Erleben [2013] provides an extensive overview of the numerical methods used to solve LCPs (and its variants) in the context of physical simulation. Recent work in computer graphics has compared the performance of various MLCP solvers [Enzenhöfer et al. 2018], including both iterative and pivoting methods. There have been similar evaluations in the robotics literature [Drumwright and Shell 2012]. We briefly present some of the pertinent work in this area in the following sections.

2.2 Iterative methods

Off-the-shelf physics engines, such as Bullet [Coumans 2018] and Box2D [Catto 2018], model the constrained dynamics as an MLCP. These rely extensively on iterative fixed-point methods, such as the Projected Gauss-Seidel (PGS) method [Erleben 2007] or Sequential impulses [Guendelman et al. 2003], due to their ability to quickly produce approximate solutions that meet minimum error requirements. However, previous work has noted that fixed-point algorithms often converge slowly when the lead matrix is ill-conditioned, as is the case for many of our target examples, or the initial solution poorly initialized [Erleben 2004]. Poorly conditioned linear systems are typically due to large mass ratios between bodies coupled by constraints, but may also be caused by numerical singularities when the regularization term used to model compliance is very small.

Erleben [2007] developed a specialized PGS algorithm to specifically treat large stacks. The projected Jacobi method is a related algorithm and it has been popularized due to ease of parallelization, but is infamously known for its poor convergence when compared to PGS. Recent work has used Chebyshev polynomials [Wang 2015] and a Nesterov momentum term [Mazhar et al. 2015] to accelerate the convergence of this class of algorithms. Whereas Tonge et al. [2012] introduced a mass splitting scheme for large scale simulations on the GPU. Fratarcangeli et al. [2016] use a graph coloring scheme to not only improve convergence, but parallelize the solve and thus dramatically improve performance.

2.3 Pivoting methods

Pivoting methods try to find a partitioning of the system into *free* (basic) and *tight* (non-basic) variables. The free label indicates that a variable is within bounds, and thus unknown, whereas the tight label indicates the value of the variable is equal to the bound, and is therefore assumed to be known. We refer to the labeling for all variables as the *index set*. If the index set changes between iterations (or pivoting steps), the system matrix needs to be updated and its factorization must be recomputed if a direct linear solver is being used. Therefore, complex simulations requiring a large number of pivoting steps may be intractable for interactive applications.

One of the most popular pivoting methods is Lemke's algorithm [Cottle et al. 1993], which is a simplex algorithm wherein a sin-

gle variable is treated at each iteration. The algorithm is guaranteed to converge if a solution exists to the LCP, but requires many steps for complex problems. An efficient version of Lemke's algorithm was developed by Lloyd [2005] and the algorithm achieves nearly linear complexity under the assumption of a fixed size problem, although typically a complexity of $O(n^3)$ is expected. Dantzig's algorithm [Cottle and Dantzig 1968] has been previously used by interactive computer graphics work [Baraff 1994], and the algorithm is offered as an alternative solver in the Open Dynamics Engine (ODE) toolkit [Smith et al. 2005].

Baraff [1994] notes that an incremental factorization can be used for the inner pivoting loop, and thus an overall complexity close to $O(n^3)$ is achievable. They hint that low-rank modifications of the LU factorization of the lead matrix are possible, but provide few details or analysis. However, in our work, we perform an in-depth analysis regarding performance and accuracy for a variety of complex 3D simulations, and motivate our approach using empirical evidence. Furthermore, our approach applies only a series of low-rank downdates to the Cholesky factorizaton (rather than both updating and downdating), which we found improves numerical stability.

Murty and Yu [1988] proposed a Bard-type pivoting method that preserves complementarity between variables at all times and changes the index sets for a single pair of complementary variables at each step (single pivoting). This algorithm is proven to converge for LCPs with positive definite lead matrices, although convergence is slow. Keller's method [Keller 1973] preserves complementarity and nonnegativity of the LCP solution variables and is said to be more efficient for LCPs with large positive semi-definite lead matrices [Júdice 1994]. Júdice and Pires [1994] introduced a block version of Murty's method which is much more efficient for large problems since it allows changing many index sets in a single algorithm iteration. This block version is proven to converge as it switches to single pivoting (Murty's method) if the number of variables out of bounds does not decrease in a user-defined number of iterations. However, in practice the algorithm mostly performs block pivots and switches to single pivoting only for ill-conditioned problems. An extensive survey of the most common pivoting methods can be found in Júdice [1994].

2.4 Hybrid methods

Direct linear solvers, involving a Cholesky or LU decomposition of the system matrix, are heavily utilized by pivoting methods when exact solutions are required. However, direct solvers have received much less attention from the computer graphics community. This is due mainly to the higher computational overhead associated with these solvers compared to iterative techniques. Lacoursière [2007] proposed a splitting method that iterates between a direct block pivoting solve for the combined sub-set of articulation and non-interpenetration constraints, followed by an iterative solve for non-interpenetration and friction constraints. This method guarantees accurate solutions for stiff joint and contact normal forces while compromising the accuracy of the friction forces in order to reach better performance than direct solvers. However, Enzenhöfer et al. [2018] noted problems with this approach for problems where accurate friction forces are required.

2.5 Low-rank matrix updates

Our work relies on efficient low-rank modifications of a Cholesky factorization, specifically a row deletion corresponding to a rank-1 downdate. Davis and Hager [2005] provide a throughout treatment of low-rank modifications for both dense and sparse Cholesky factorizations, and indeed we used theirs as a starting point for our work. Seeger [2004] also demonstrated that low-rank modifications to a Cholesky factorization can improve stability versus modifications to the system matrix using the Sherman-Morrison-Woodbury formula.

A method for low-rank modifications of the Cholesky factorization of a 3D mesh has recently been proposed Herholz and Alexa [2018]. They re-use the factorization of the full mesh to efficiently perform operations on sub-meshes. Similar to their work, we use the so-called *left-looking algorithm* to compute our Cholesky factorizations, although in our case the domain is the constraints of a multibody system rather than a mesh.

3 SOLVING CONSTRAINED DYNAMICS

Here we briefly present the constrained multibody dynamics formulation used in our work, and describe how the resulting MLCP is solved using a block principal pivoting method. This then leads to the modified version of the algorithm that we propose in this paper.

Physical simulations for computer graphics are typically performed using a discrete numerical integration scheme with time step h. Using the velocity-level constrained dynamical equations and adopting a single-step implicit Euler integration scheme [Baraff and Witkin 1998], for an n-body system with m constraint equations this gives the linear system

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \frac{1}{h^2}\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{v}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{M}\mathbf{v} + h\mathbf{f} \\ -\frac{1}{h}\boldsymbol{\phi} \end{bmatrix} , \tag{1}$$

with mass matrix $\mathbf{M} \in \mathbb{R}^{6n\times 6n}$, momentum $\mathbf{M}\mathbf{v}$, constraint Jacobian $\mathbf{J} \in \mathbb{R}^{m\times 6n}$, constraint impulses $\boldsymbol{\lambda} \in \mathbb{R}^m$, generalized velocities $\mathbf{v} \in \mathbb{R}^{6n}$, applied forces $\mathbf{f} \in \mathbb{R}^{6n}$, constraint violations $\boldsymbol{\phi} \in \mathbb{R}^m$, and diagonal constraint compliance matrix $\mathbf{C} \in \mathbb{R}^{m\times m}$. All variables carrying the superscript $\square^+$ are implicit quantities, meaning they are computed at the end of the time step. Forming the Schur complement of the upper left block gives the reduced system

$$\underbrace{\left[\frac{1}{h^2}\mathbf{C} + \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right]}_{\mathbf{A}} \boldsymbol{\lambda}^+ = \underbrace{-\frac{1}{h}\boldsymbol{\phi} - \mathbf{J}\mathbf{M}^{-1}(\mathbf{M}\mathbf{v} + h\mathbf{f})}_{\mathbf{b}} , \tag{2}$$

which is the form used by many open source and commercial rigid body physics engines. Note that the block diagonal form of $\mathbf{M}$ makes it trivial to invert, and since $\mathbf{C}$ is a diagonal matrix with only positive values, the matrix $\mathbf{A}$ is positive definite and symmetric.

Since our simulations involve MLCPs with limits on the constraint impulses $\boldsymbol{\lambda}$, we introduce feasibility and complementarity conditions such that

$$\mathbf{A}\boldsymbol{\lambda}^+ - \mathbf{b} = \mathbf{w} = \mathbf{w}_+ + \mathbf{w}_- \tag{3a}$$

$$\begin{cases} \mathbf{0} \le \mathbf{w}_+ \perp \boldsymbol{\lambda}^+ - \boldsymbol{\lambda}_{\text{lo}} \ge \mathbf{0} \\ \mathbf{0} \le \mathbf{w}_- \perp \boldsymbol{\lambda}_{\text{hi}} - \boldsymbol{\lambda}^+ \ge \mathbf{0} \end{cases} , \tag{3b}$$

where $\boldsymbol{\lambda}_{\text{hi}}$ and $\boldsymbol{\lambda}_{\text{lo}}$ are upper and lower bounds on the constraint impulses, respectively. For example, the lower and upper impulse limits of a Coulomb friction cone are often computed as $\boldsymbol{\lambda}_{\text{lo}} = -\mu\boldsymbol{\lambda}_{\text{N}}$ and $\boldsymbol{\lambda}_{\text{hi}} = \mu\boldsymbol{\lambda}_{\text{N}}$, where μ is the static coefficient of friction and $\boldsymbol{\lambda}_{\text{N}}$ is the normal impulse which perpendicular to the collision surface. Note that we introduce the residual vector $\mathbf{w}$ and divide it into non-negative complementary components, i.e. $\mathbf{0} \leq \mathbf{w}_+ \perp \mathbf{w}_- \geq \mathbf{0}$. The solution of the constraint impulses $\boldsymbol{\lambda}^+$ is then substituted into the first line of Eq. 1 to compute the generalized velocities.

3.1 Block Principal Pivoting

We use a Bard-type pivoting method to find a solution to the MLCP in Eq. 3. Specifically, the block principal pivoting (BPP) method proposed by Júdice and Pires [1994]. It combines a fast block strategy with Murty's single principal pivoting algorithm, with the main difference being that BPP exchanges multiple variables simultaneously during changes to the index set.

At each iteration in the algorithm, the constraint variables are labeled as belonging to either the free ($\mathbb{F}$) or tight ($\mathbb{T}$) index set. Tight variables have reached a lower or upper bound, and therefore their value is defined by the feasibility conditions, whereas free variables have not exceeded these bounds, and therefore $\mathbf{w}_i = 0, \forall\ i \in \mathbb{F}$. For instance, when a frictional contact is "sticking", the constraint impulse is within bounds, and thus free. However, once the contact begins sliding, the constraint impulse is determined by the limits of the friction model, and thus the index set changes to tight.

A *pivoting step* is the process of proposing a candidate solution and determining if the index set is correct. Based on an assumption of the index set, a candidate solution of the linear system in Eq. 3 is computed. We then verify if the assumption is correct, i.e. all variables satisfy feasibility and complementarity conditions. If not, the index set of all variables i which do not satisfy the conditions are changed such that

$$i \in \begin{cases} \mathbb{F}, & \text{if } \boldsymbol{\lambda}_{\text{lo}} < \boldsymbol{\lambda}_i < \boldsymbol{\lambda}_{\text{hi}} \text{ and } \mathbf{w}_i = 0 \\ \mathbb{T}_{\text{lo}}, & \text{if } \boldsymbol{\lambda}_i \leq \boldsymbol{\lambda}_{\text{lo}} \text{ and } \mathbf{w}_i > 0 \\ \mathbb{T}_{\text{hi}}, & \text{if } \boldsymbol{\lambda}_i \geq \boldsymbol{\lambda}_{\text{hi}} \text{ and } \mathbf{w}_i < 0 \end{cases} . \tag{4}$$

The algorithm terminates with success if the index sets between two consecutive pivoting steps do not change.

Note that in Eq. 4 the tight set is decomposed into both lower ($\mathbb{T}_{\text{lo}}$) and upper ($\mathbb{T}_{\text{hi}}$) bounded variables, and together these form the complete tight set, such that $\mathbb{T} = \mathbb{T}_{\text{lo}} \cup \mathbb{T}_{\text{hi}}$. The system matrix and right-hand side vector in Eq. 3 can be regrouped according to the index set, such that

$$\begin{bmatrix} \mathbf{A}_{\mathbb{F}\mathbb{F}} & \mathbf{A}_{\mathbb{F}\mathbb{T}} \\ \mathbf{A}_{\mathbb{T}\mathbb{F}} & \mathbf{A}_{\mathbb{T}\mathbb{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_{\mathbb{F}} \\ \boldsymbol{\lambda}_{\mathbb{T}} \end{bmatrix} - \begin{bmatrix} \mathbf{b}_{\mathbb{F}} \\ \mathbf{b}_{\mathbb{T}} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_{\mathbb{F}} \\ \mathbf{w}_{\mathbb{T}} \end{bmatrix} . \tag{5}$$

We observe that for tight variables the value of $\boldsymbol{\lambda}_{\mathbb{T}}$ is known, and that for free variables the residual velocity $\mathbf{w}_{\mathbb{F}} = 0$. This means that we only need to solve for $\boldsymbol{\lambda}_{\mathbb{F}}$ at each step in the algorithm

$$\mathbf{A}_{\mathbb{F}\mathbb{F}}\boldsymbol{\lambda}_{\mathbb{F}} = \mathbf{b}_{\mathbb{F}} - \mathbf{A}_{\mathbb{F}\mathbb{T}}\boldsymbol{\lambda}_{\mathbb{T}}. \tag{6}$$

A Cholesky factorization may be used to solve the linear system in Eq. 6 where $\mathbf{A}_{\mathbb{F}\mathbb{F}} = \mathbf{L}_{\mathbb{F}\mathbb{F}}\mathbf{L}_{\mathbb{F}\mathbb{F}}^{\mathsf{T}}$ and $\mathbf{L}_{\mathbb{F}\mathbb{F}}$ is a lower triangle matrix. Pseudo-code for the BPP algorithm using a Cholesky factorization is given in Alg. 1. We observe that the matrix $\mathbf{A}_{\mathbb{F}\mathbb{F}}$ must be refactored whenever the index sets change (i.e., at each step of the algorithm). This can be costly, since the Cholesky factorization can have a worst-case complexity of $O(m^3)$.

Algorithm 1 Block principal pivoting

```
procedure BPP(A, b, λ_hi, λ_lo)
    initialize 𝔽 and 𝕋
    k = 0
    do
        L_𝔽𝔽 ← Cholesky(A_𝔽𝔽)
        ∀ i ∈ 𝕋_lo,  λ_i ← λ_lo,i
        ∀ i ∈ 𝕋_hi,  λ_i ← λ_hi,i
        solve L_𝔽𝔽 L_𝔽𝔽^T λ_𝔽 = b_𝔽 − A_𝔽𝕋 λ_𝕋
        w = Aλ − b
        update 𝔽, 𝕋 according to Eq. 4
        k = k + 1
    while (k < max steps) and (index sets 𝔽, 𝕋 changed)
end procedure
```

However, if the tight set is small, a more efficient approach would be to modify the factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$ using a series of low-rank updates to obtain $\mathbf{L}_{\mathbb{F}\mathbb{F}}$. This is precisely the approach we take. The subsequent subsections justify our approach and demonstrate how the BPP algorithm can be modified to support low-rank downdates.

3.2 Downdating the Cholesky factorization

In a left-looking Cholesky factorization, a change to the ith column of the factorization affects only the sub-block $\tilde{\mathbf{L}}$, formed by the rows and columns with indices in the range $i + 1 \ldots m$ (see Fig. 2). When a constraint variable pivots to the tight set $\mathbb{T}$, a downdate of the original factorization $\mathbf{L}$ can be performed by modifying only the entries in $\tilde{\mathbf{L}}$. In order to remove the row and the column corresponding to the ith variable, we downdate using the vector σ_i, and this process is repeated for each variable $i \in \mathbb{T}$ sequentially. The cost of recomputing the Cholesky factorization can therefore be reduced by copying the original factorization $\mathbf{L}$ and updating in-place the sub-matrix $\tilde{\mathbf{L}}$, eventually giving $\mathbf{L}_{\mathbb{F}\mathbb{F}}$ for indices in $\mathbb{F}$.

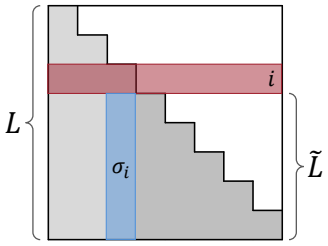

Figure 2: A change to the ith column of the factorization will only affect the entries of $\tilde{\mathbf{L}}$

3.2.1 Preliminary analysis of index sets. We inspected the behavior of the index sets across many time steps for the six scenarios shown in the teaser. The results (see Fig. 3) suggest that the size of $\mathbb{T}$ is often small compared to the total number of variables, even for complex scenarios. This indicates that, beginning from an initial factorization, only a small number of low-rank modifications may be used to obtain $\mathbf{L}_{\mathbb{F}\mathbb{F}}$. As a next step, we performed a number of experiments on randomly generated symmetric positive-definite matrices of various sizes using a dense matrix storage in order to determine the threshold for which the downdating scheme becomes

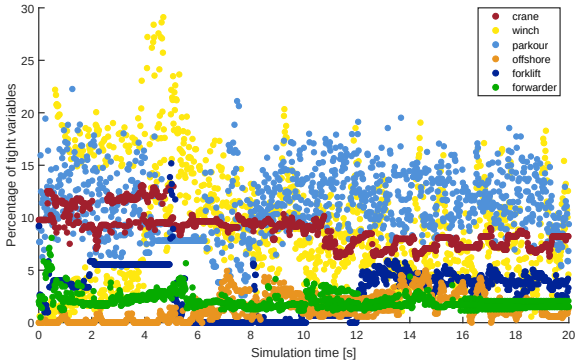

Figure 3: Variation of the percentage of variables in matrix A which pivot into the tight set $\mathbb{T}$

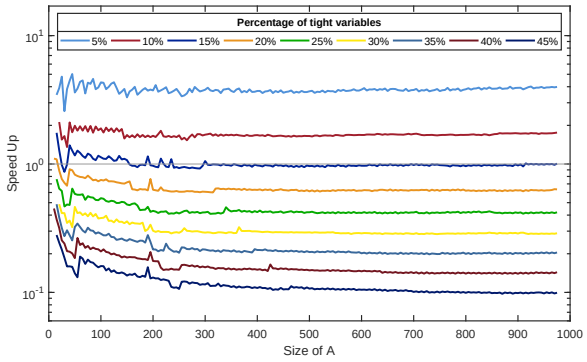

Figure 4: Speed-up using low-rank downdates instead of recomputing the Cholesky factorization of $\mathbf{A}_{\mathbb{FF}}$ with respect to the size of A. Each curve corresponds to a different percentage of tight variables.

inefficient. We computed the Cholesky factorization $\mathbf{L}$ for the full matrix and randomly select variables to be pivoted into the tight set according to a pre-defined percentage of tight variables with respect to all variables. We then compared the time required to downdate $\mathbf{L}$ versus the time spent to recompute the factorization $\mathbf{A}_{\mathbb{FF}} = \mathbf{L}_{\mathbb{FF}}\mathbf{L}_{\mathbb{FF}}^{\mathsf{T}}$. Fig. 4 illustrates that a speed-up can be realized when the percentage of tight variables is < 15%. Revisiting Fig. 3, we observe that this is the case for most of the simulation frames in our target examples. We also note that the speed-up factor remains nearly constant for a fixed percentage of tight variable even if the overall matrix size increases or decreases. Encouraged by these results, we proceeded to develop a modified version of the BPP algorithm incorporating low-rank downdates.

3.3 Modified BPP algorithm

Pseudo-code for our proposed modification to the BPP algorithm is provided in Alg. 2. Our primary contribution is that the Cholesky factorization is computed only once per simulation step, rather than once per iteration of the pivoting algorithm. Instead, at each pivoting step, we copy the initial factorization and apply a sequence

Algorithm 2 Our efficient block principal pivoting

```
1:  procedure BPP_DOWNDATE(A, b, λ_lo, λ_lo)
2:      L_0 ← CHOLESKY(A)
3:      s ← SKYLINE(L_0)
4:      initialize index sets 𝔽 and 𝕋
5:      k = 0
6:      do
7:          L = L_0
8:          for each i ∈ 𝕋 do
9:              σ = L_{i+1…m,i}
10:             L ← DOWNDATE(i, L, σ, s)
11:         end for
12:         ∀ i ∈ 𝕋_lo,  λ_i ← λ_{lo,i}
13:         ∀ i ∈ 𝕋_hi,  λ_i ← λ_{hi,i}
14:         λ_𝔽 ← CHOLSOLVE(L, b_𝔽 − A_𝔽𝕋 λ_𝕋, 𝔽, s)
15:         w = Aλ − b
16:         update 𝔽, 𝕋 according to Eq. 4
17:         k = k + 1
18:     while (k < max steps) and (index sets 𝔽, 𝕋 changed)
19: end procedure
```

of low-rank downdates to the matrix $\mathbf{L}$, one for each tight variable in the system. Constraint impulses $\boldsymbol{\lambda}_{\mathbb{T}}$ are known and may be transferred to the right-hand side of Eq. 6. We therefore need to remove the rows and columns in $\mathbf{L}$ associated with these variables.

3.3.1 Downdating. The computation of the downdate vector is based on the dense row deletion algorithm given by Davis and Hager [2005], where vector $\boldsymbol{\sigma}_i = \mathbf{L}_{i+1\ldots m,i}$ consists of the rows below the diagonal of the ith column. Vector $\boldsymbol{\sigma}_i$ is then used to perform a rank-1 downdate of the lower right sub-block of the Cholesky factorization. Note that $\mathbf{L}$ is not resized during this process. Matrix elements are modified in place when the downdate is performed, and during the Cholesky solve in line 14 in Alg. 2; rows and columns corresponding to tight variables are skipped. The DOWNDATE routine in Alg. 3 gives pseudo-code to downdate $\mathbf{L}$ for a single variable at index i. This routine is similar to low-rank modification routines which are commonplace in linear algebra frameworks, e.g. Eigen [2018]. However, here we modify only the sub-matrix. Furthermore, we optimize the routine to take advantage of matrix sparsity using a *skyline* coding of the factorization, which we describe in the next section.

3.4 Skyline coding

As a pre-process, we reorder constraint variables according to a traversal of the constraint graph using the *Cuthill-McKee* (CM) algorithm [Cuthill and McKee 1969]. This reduces the envelope of the system matrix and, thus, reduces fill-in for $\mathbf{L}$ giving an overall reduced bandwidth (see Fig. 5). Furthermore, we note that since we only remove, but never add rows and columns, the bandwidth of $\mathbf{A}_{\mathbb{FF}}$ never increases relative to $\mathbf{A}$. The bandwidth of the matrix may therefore be efficiently encoded using a skyline data structure $\mathbf{s}$, which stores the bandwidth at each column of $\mathbf{L}$ relative to the diagonal. In other words, each $\mathbf{s}_i$ stores the row index of the last non-zero entry of column $\mathbf{L}_i$ (see Fig. 6).

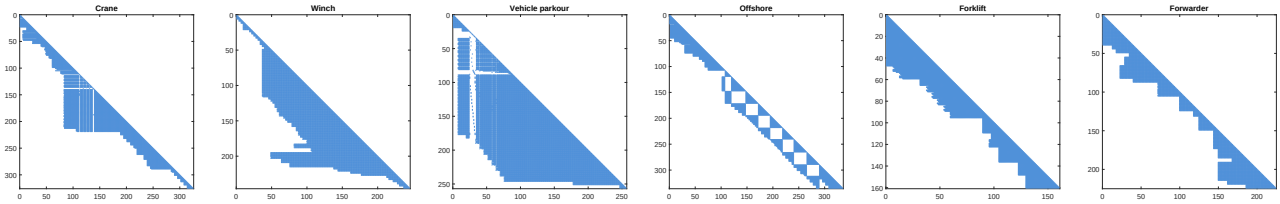

Figure 5: Reordering constraint variables of the system matrices creates a fill-in pattern with smaller bandwidth. The plots show the system matrices for the largest connected island in each simulation.

We use the skyline to limit the amount of work done in the DownDate routine, since it helps to avoid unnecessary updates on zero sub-blocks of the Cholesky factorization. We also use the skyline to reduce the amount of work during the forward/backward substitution of the Cholesky solve (see CholSolve in Alg. 3).

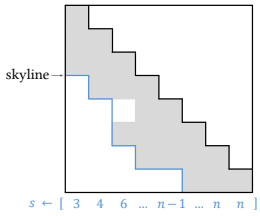

Figure 6: The skyline vector is composed of the last non-zero entry of each column

4 RESULTS

Our implementation uses the Vortex physics engine [CM Labs Simulations 2018] for collision detection, computing constraint information, and dynamics integration. The solver is implemented in C++ using double precision and the Eigen linear algebra library [Eigen 2018] for matrix multiplications and storage. We choose a column-wise dense matrix storage format to have full control over the matrix traversal and skyline coding in our modified BPP algorithm. The memory for the $m \times m$ matrix is allocated, but for the skyline version of our downdating algorithm we only iterate over elements within the envelope. We initialize all index sets as free when starting all versions of the BPP algorithm.

4.1 Examples

Fig. 1 shows the six examples we use to evaluate our modified BPP algorithm. We summarize the simulation and modeling parameters for these examples in Table 1. Table 2 contains additional information, such as the maxima for mass ratio, number of constraints and condition number for each example. Note that unconnected systems are split into multiple islands, and an MLCP is formulated and solved for each such island. The supplementary video also shows a side-by-side comparison of our modified BPP algorithm versus using the full Cholesky decomposition. The results are qualitatively identical. Below, we highlight characteristics of each example.

4.1.1 Forklift. A 3,000 kg warehouse forklift picks up a 100 kg crate, accelerates then makes a 360 degree turn around a traffic cone while frictional contact ensures that the crate remains stable on the fork. The wheels start skidding during the turn which leads to a increase of variables in the tight set to up to 5 % of all variables.

4.1.2 Mobile crane. A 30,000 kg mobile crane hoists a 2,000 kg concrete pipe segment with a 70 kg steel cable attached to a 70 kg hook. Contacts are created when the hook touches the pipe, which causes the solve time to momentarily spike.

Table 1: Simulation parameters

Friction coefficient	$\mu = 1.0$
Gravity [m/s^2]	$g = 9.81$
Constraint compliance [N/m]	10^{-6} to 10^{-10}
Step size [s]	$h = 1/60$
Simulation duration [s]	$t = 20$
Max solver iterations	$k = 35$

4.1.3 Vehicle parkour. A simulation involving a 15,000 kg light armored vehicle (LAV) traversing various obstacles (stairs, small rectangular bumps, hills). The power train of the vehicle is modeled using multiple unilateral constraints connecting the vehicle shafts, wheels, differentials, and engine, which creates a highly coupled mechanical system (see sparsity pattern in Fig. 5).

4.1.4 Offshore. A 50,000 kg offshore knuckle boom crane is rigidly attached to a 5,500,000 kg ship hoisting a 15,000 kg subsea module with lightweight steel cable. Sliding friction (tight) occurs between the ship hull and the subsea module, as well as cable bodies.

4.1.5 Forwarder. A forwarder with a 16-link 1,900 kg gripper arm is used to grasp a 400 kg wooden log and lift it in the air by raising its boom. This scenario is challenging since a stable grasp without sliding is required, and a non-negligible friction impulse is acting between log and claw due to arm rotation. The hydraulics, hydraulics cables, and the vehicle are solved independently from the main mechanical components of the gripper arm.

4.1.6 Winch. A chain, consisting of 50×1 kg links modeled using capsules and connected by spherical joints, is attached to a 400 kg winch and wound at a constant angular velocity. As the chain is dragged along the ground, many of the frictional constraints transition to a sliding mode (tight).

4.2 Performance comparison

Table 3 shows the average solve time per frame for the original BPP algorithm performing a full Cholesky factorization of $\mathbf{A}_{\mathbb{FF}}$, alongside our modified BPP algorithm performing low-rank downdates to $\mathbf{L}$ with and without taking into account the skyline. Fig. 7 shows the CPU time required to solve Eq. 3 at each time step for the examples used in our experiments. All simulations were performed on an Intel Core i7-5820K (3.3 GHz) with 16 GB of RAM. Our modified BPP algorithm with low-rank downdates and skyline gives the best

Algorithm 3 Downdating and solve with skyline coding

procedure DOWNDATE($i, \mathbf{L}, \boldsymbol{\sigma}, \mathbf{s}$)
  $\beta = 1$
  for $j \leftarrow 1$ to $(m - i - 1)$ **do**
    $k = i + j + 1$
    $x = \boldsymbol{\sigma}_j$
    $y = \mathbf{L}_{k,k}$
    $z = y^2 + \frac{x^2}{\beta}$
    $\gamma = \beta\, y^2 + x^2$
    $\mathbf{L}_{k,k} = \sqrt{z}$
    $\beta = \beta + \frac{x^2}{y^2}$
    if $(\mathbf{s}_k - k - 1) > 0$ **then** ▷ skyline check
      $j_1 = j + 1$ and $j_2 = j_1 + (\mathbf{s}_k - k - 1)$
      $k_1 = k + 1$ and $k_2 = k_1 + (\mathbf{s}_k - k - 1)$
      $\boldsymbol{\sigma}_{j_1 \ldots j_2} = \boldsymbol{\sigma}_{j_1 \ldots j_2} - \frac{x}{y}\mathbf{L}_{k_1 \ldots k_2, k}$
      $\mathbf{L}_{k_1 \ldots k_2, k} = \mathbf{L}_{k_1 \ldots k_2, k} + \frac{\sqrt{z}\, x}{\gamma}\boldsymbol{\sigma}_{j_1 \ldots j_2}$
    end if
    $j = j + 1$
  end for
  return $\mathbf{L}$
end procedure

procedure CHOLSOLVE($\mathbf{L}, \mathbf{b}_{\mathbb{F}}, \mathbb{F}, \mathbf{s}$)
  $\boldsymbol{\lambda}_{\mathbb{F}} = \mathbf{b}_{\mathbb{F}}$
  for each $i \in \mathbb{F}$ **do** ▷ forward substitution $\mathbf{L}\mathbf{y} = \mathbf{b}_{\mathbb{F}}$
    $\boldsymbol{\lambda}_i \leftarrow \frac{\boldsymbol{\lambda}_i}{\mathbf{L}_{i,i}}$
    $j \leftarrow$ NEXTINSET($\mathbb{F}, i$) ▷ next index in $\mathbb{F}$ following i
    while $j < \mathbf{s}_i$ **do**
      $\boldsymbol{\lambda}_j \leftarrow \boldsymbol{\lambda}_j - \mathbf{L}_{j,i}\boldsymbol{\lambda}_i$
      $j \leftarrow$ NEXTINSET($\mathbb{F}, j$)
    end while
  end for
  for each $i \in \mathbb{F}$ **do** ▷ backward substitution $\mathbf{L}^{\mathsf{T}}\boldsymbol{\lambda}_{\mathbb{F}} = \mathbf{y}$
    $j \leftarrow$ NEXTINSET($\mathbb{F}, i$)
    while $j < \mathbf{s}_i$ **do**
      $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i - \mathbf{L}_{j,i}\boldsymbol{\lambda}_j$
      $j \leftarrow$ NEXTINSET($\mathbb{F}, j$)
    end while
    $\boldsymbol{\lambda}_i \leftarrow \frac{\boldsymbol{\lambda}_i}{\mathbf{L}_{i,i}}$
  end for
  return $\boldsymbol{\lambda}_{\mathbb{F}}$
end procedure

performance for all test cases, followed by the algorithm using low-rank downdates without skyline information (except for the winch simulation). For simplicity, we do not parallelize the computations of multiple unconnected islands, i.e. the presented solve times are the sum of the dynamics solve times of all islands. Note that Table 3 and Table 4 exclude frames for which there are no tight variables, i.e. no pivoting, since the BPP algorithms using full Cholesky or downdating are equivalent in this case.

The highest speed-up is experienced for the mobile crane, in particular when the crane hook is in contact with the pipe in the beginning of the simulation (0-5 s). The offshore simulation, which resembles the mobile crane example, also achieves a significant performance improvement. In practice, one would only perform low-rank downdates if the number of tight variables is sufficiently small in order to guarantee to be always at least as fast as the original algorithm applying the full Cholesky factorization. In our tests, we obtain a slow-down for the winch example since we always use the downdating strategy.

Table 2: Maxima of mass ratio, number of constraints per contiguously solved system and condition number of A

Scenario	Mass ratio	Number of constraints	Condition number
Crane	6,300:1	430	10^9
Winch	400:1	370	10^8
Vehicle parkour	20,000:1	260	10^{19}
Offshore	4,400,000:1	350	10^9
Forklift	2,700:1	300	10^8
Forwarder	6,500:1	250	10^{11}

Table 3: Average dynamics solve time and speed-up for each scenario and algorithm.

Scenario	Full Cholesky	Downdating	Downdating Skyline
Crane	12.4 ms	4.7 ms (2.7 ×)	3.4 ms (3.6 ×)
Winch	2.0 ms	2.4 ms (0.9 ×)	1.6 ms (1.3 ×)
Vehicle parkour	3.0 ms	2.2 ms (1.4 ×)	2.1 ms (1.4 ×)
Offshore	6.7 ms	4.0 ms (1.7 ×)	2.9 ms (2.3 ×)
Forklift	1.2 ms	1.0 ms (1.3 ×)	0.8 ms (1.5 ×)
Forwarder	3.1 ms	2.5 ms (1.2 ×)	2.0 ms (1.6 ×)

Table 4: Numerical error introduced by downdating L versus computing $\mathbf{L}_{\mathbb{F}\mathbb{F}}$. The two rows show the error when downdating is performed without (first row) and with (second row) skyline information.

Scenario	Min	Max	Mean	Median
Crane	6.74e-12	3.00e-10	4.77e-11	3.80e-11
	6.76e-12	2.50e-10	4.84e-11	3.88e-11
Winch	0.00e+00	7.29e-11	6.60e-13	1.31e-13
	0.00e+00	7.34e-11	6.07e-13	1.28e-13
Vehicle parkour	3.17e-13	1.81e-09	4.93e-12	7.18e-13
	2.71e-13	7.99e-10	1.85e-12	7.36e-13
Offshore	0.00e+00	2.00e-08	1.50e-09	6.55e-11
	0.00e+00	1.00e-08	1.30e-09	5.55e-11
Forklift	0.00e+00	5.74e-11	1.50e-12	1.46e-12
	0.00e+00	1.17e-11	1.35e-12	1.36e-12
Forwarder	1.11e-10	7.00e-08	4.04e-09	3.22e-09
	1.11e-10	7.00e-08	4.04e-09	3.22e-09

4.3 Discussion

Table 4 shows the numerical error introduced by downdating $\mathbf{L}$ versus recomputing the full Cholesky factorization. This error is determined by the Frobenius matrix norm between the lower triangular matrix obtained with and without low-rank downdates.

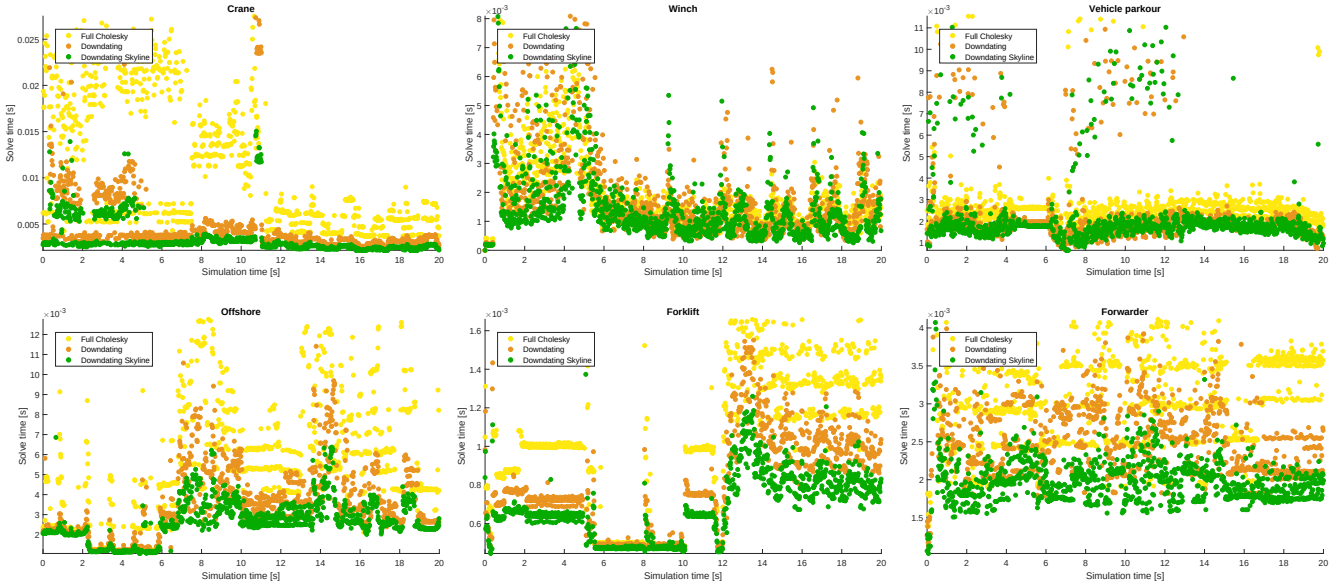

Figure 7: Time to solve the constrained dynamics equations for the examples shown in the teaser. From left to right, top row: mobile crane, chain and winch, vehicle parkour. Bottom row: offshore, forklift, logging forwarder.

The first line for each example represents downdating without considering the skyline, whereas for the second line the skyline information is used. We observe that the errors remain reasonably low for all scenarios (10^{-8} at worst) and that the fluctuations between minimum and maximum errors are low (difference of 10^{-4} at worst). The skyline does not significantly impact the error.

We have evaluated an alternate version of our algorithm in which both low-rank updates and downdates are used to incrementally modify the factorization at each pivoting step. Updates correspond to variables pivoting from the $\mathbb{T} \rightarrow \mathbb{F}$, and downdates $\mathbb{F} \rightarrow \mathbb{T}$. However, we find that after just a few pivoting steps, numerical errors quickly accumulate and result in unstable simulations, specifically for ill-conditioned system. Instead, we find it much more stable to simply copy the initial factorization of $\mathbf{A}$ at each pivoting step and apply downdates only.

A skyline encoding is used to keep track of the envelope of $\mathbf{A}$ for which we typically get fill-ins in the lower triangular matrix $\mathbf{L}$. This information is used by our proposed method to iterate only over nonzero element and reduce computational effort during the low-rank downdates and the solve of the linear system. We allocate enough memory to store the entire $m \times m$ matrix in dense format even if the skyline information is considered. It is likely that further performance improvements could be realized by allocating only enough memory space to store elements in the non-zero envelope. This would also allow larger matrices to be stored in cache.

5 CONCLUSIONS AND FUTURE WORK

We have proposed a modification to the BPP algorithm that improves efficiency by using low-rank modifications to form the lead matrix. We further improve performance by using a skyline data structure that encodes the bandwidth of the Cholesky factorization, which is used by both the Cholesky solve and downdating processes. Our work is mainly motivated by the simulation of stiff physical systems which typically requires direct linear solvers. We have demonstrated that our method gives a significant speed-up over the standard algorithm for challenging scenarios.

Updates to a factorization using sparse storage are not considered by our approach, but this would be an interesting avenue of future work. However, it is unclear that for the size of problems targeted in the context of our work would benefit from a sparse storage since there is additional effort required to manage sparse data structures. We find a skyline encoding of the matrices to be much better trade-off in performance versus initialization overhead.

We intend to explore other strategies to further improve performance. One technique often utilized to improve the convergence of iterative solvers is to exploit temporal coherence by using the solution at the previous timestep as an initial estimate of the solution for the current time step (warmstarting). Similarly, we could assume variables in the tight set at the previous time step will continue to be tight at the current time step, and the constraint variable ordering could be permuted such that ß $\in \mathbb{T}$ are moved to the end of the ordering, and thus affect mainly the lower-right block of the factorization. This would mean few computations are required to downdate these variables, but it may also diminish the benefits of the low-bandwidth achieved through CM reordering.

ACKNOWLEDGMENTS

This work was supported by a Collaborative Research and Development (CRD) grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. We also thank Daniel Holz and Marek Teichmann at CM Labs Simulations for their feedback and stimulating discussions.

REFERENCES

Baraff, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics Proceedings, Annual Conference Series* 23-24, 23–34.

Baraff, D. and Witkin, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 43–54.

Catto, E. 2018. Box2D physics engine. https://box2d.org/. [Online].

CM Labs Simulations. 2018. Vortex Studio. http://www.cm-labs.com/. [Online].

Cottle, R., Pang, J., and Stone, R. 1993. *The Linear Complementarity Problem*. SIAM.

Cottle, R. W. and Dantzig, G. B. 1968. Complementary pivot theory of mathematical programming. *Linear Algebra Appl.* 1, 1, 103–125.

Coumans, E. 2018. Bullet physics library. https://pybullet.org/. [Online].

Cuthill, E. and McKee, J. 1969. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*. ACM, 157–172.

Davis, T. A. and Hager, W. W. 2005. Row modifications of a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.* 26, 3, 621–639.

Drumwright, E. and Shell, D. A. 2012. Extensive analysis of Linear Complementarity Problem (LCP) solver performance on randomly generated rigid body contact problems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5034–5039. https://doi.org/10.1109/IROS.2012.6385974

Eigen. 2018. Eigen C++ library for linear algebra, vector and matrix multiplications. http://eigen.tuxfamily.org/. [Online].

Enzenhöfer, A., Andrews, S., Teichmann, M., and Kövecses, J. 2018. Comparison of Mixed Linear Complementarity Problem Solvers for Multibody Simulations with Contact. In *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association. https://doi.org/10.2312/vriphys.20181063

Erleben, K. 2004. *Stable, robust, and versatile multibody dynamics animation*. Ph.D. Dissertation. University of Copenhagen, Copenhagen.

Erleben, K. 2007. Velocity-based Shock Propagation for Multibody Dynamics Animation. *ACM Trans. Graph.* 26, 2, Article 12, 20 pages.

Erleben, K. 2013. Numerical methods for linear complementarity problems in physics-based animation. In *ACM SIGGRAPH 2013 Courses*. ACM, 8.

Fratarcangeli, M., Tibaldo, V., and Pellacini, F. 2016. Vivace: A Practical Gauss-seidel Method for Stable Soft Body Dynamics. *ACM Trans. Graph.* 35, 6, Article 214, 9 pages. https://doi.org/10.1145/2980179.2982437

Guendelman, E., Bridson, R., and Fedkiw, R. 2003. Nonconvex Rigid Bodies with Stacking. *ACM Trans. Graph.* 22, 3, 871–878. https://doi.org/10.1145/882262.882358

Herholz, P. and Alexa, M. 2018. Factor Once: Reusing Cholesky Factorizations on Sub-meshes. In *SIGGRAPH Asia 2018 Technical Papers (SIGGRAPH Asia '18)*. ACM, New York, NY, USA, Article 230, 9 pages. https://doi.org/10.1145/3272127.3275107

Júdice, J. J. 1994. Algorithms for linear complementarity problems. In *Algorithms for Continuous Optimization*, Emilio Giuseppe Spedicato (Ed.). Springer, 435–474.

Júdice, J. J. and Pires, F. M. 1994. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & operations research* 21, 5, 587–596.

Keller, E. L. 1973. The general quadratic optimization problem. *Mathematical Programming* 5, 1, 311–337.

Lacoursière, C. 2007. A Parallel Block Iterative Method for Interactive Contacting Rigid Multibody Simulations on Multicore PCs. In *Applied Parallel Computing. State of the Art in Scientific Computing*, Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Waśniewski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 956–965.

Lloyd, J. E. 2005. Fast implementation of Lemke's algorithm for rigid body contact simulation. In *IEEE international conference on robotics and automation*. 4538–4543.

Mazhar, H., Heyn, T., Negrut, D., and Tasora, A. 2015. Using Nesterov's Method to Accelerate Multibody Dynamics with Friction and Contact. *ACM Trans. Graph.* 34, 3, Article 32, 14 pages. https://doi.org/10.1145/2735627

Moreau, J. J. 1988. Unilateral Contact and Dry Friction in Finite Freedom Dynamics. *Nonsmooth Mechanics and Applications* 302, 1–82.

Murty, K. G. and Yu, F.-T. 1988. *Linear complementarity, linear and nonlinear programming*. Vol. 3. Citeseer.

Seeger, M. 2004. *Low rank updates for the Cholesky decomposition*. Technical Report.

Smith, R. and others. 2005. Open dynamics engine.

Stewart, D. E. and Trinkle, J. C. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. *Internat. J. Numer. Methods Engrg.* 39, 15, 2673–2691.

Tonge, R., Benevolenski, F., and Voroshilov, A. 2012. Mass Splitting for Jitter-Free Parallel Rigid Body Simulation. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2012* 31, 4, 1–8.

Wang, H. 2015. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)* 34, 6, 246.