QLB: Collision-Aware Quasi-Newton Solver with Cholesky and L-BFGS for Nonlinear Time Integration

Bethany Witemeyer
Texas A&M University
USA

Nicholas J. Weidner
Texas A&M University
USA

Timothy A. Davis
Texas A&M University
USA

Theodore Kim
Yale University
USA

Shinjiro Sueda
Texas A&M University
USA

Figure 1: Our method accelerates the convergence of nonlinear implicit time integration schemes, and works with complex materials, frictional contact, and self collisions.

ABSTRACT

We advocate for the straightforward applications of the Cholesky and the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithms in the context of nonlinear time integration of deformable objects with dynamic collisions. At the beginning of each time step, we form and factor the Hessian matrix, accounting for all internal forces while omitting the implicit cross-coupling terms from the collision forces between multiple dynamic objects or self collisions. Then during the nonlinear solver iterations of the time step, we implicitly update this Hessian with L-BFGS. This approach is simple to implement and can be readily applied to any nonlinear time integration scheme, including higher-order schemes and quasistatics. We show that this approach works well in a wide range of settings involving complex nonlinear materials, including heterogeneity and anisotropy, as well as collisions, including frictional contact and self collisions.

CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**; *Simulation by animation.*

KEYWORDS

Physical simulation, deformation, finite elements

ACM Reference Format:
Bethany Witemeyer, Nicholas J. Weidner, Timothy A. Davis, Theodore Kim, and Shinjiro Sueda. 2021. QLB: Collision-Aware Quasi-Newton Solver with Cholesky and L-BFGS for Nonlinear Time Integration. In *Motion, Interaction and Games (MIG '21), November 10–12, 2021, Virtual Event, Switzerland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3487983.3488297

MIG '21, November 10–12, 2021, Virtual Event, Switzerland

1 INTRODUCTION

Physics-based simulation has become an important tool in computer animation. Starting with the seminal work by Terzopoulos et al. [1987], many sophisticated methods have been proposed to improve various aspects of these simulations. An early example of this is the *linearly* implicit Euler integration method of Baraff and Witkin [1998], which is still in use today in a production environment due to its favorable blend of efficiency, stability, and visual fidelity [Kim and Eberle 2020]. More recently, *nonlinear* integration methods have become increasingly popular in physics-based animation, with examples including BDF1 (1st-order Backward Differentiation Formula) [Hairer et al. 2006], BDF2 (2nd-order Backward Differentiation Formula) [English and Bridson 2008; Geilinger et al. 2020], TR-BDF2 (Trapezoidal-BDF2) [Xu and Barbič 2017], and SDIRK2 (2nd-order Singly-Diagonal Implicit Runge-Kutta) [Löschner et al. 2020]. Quasistatic simulations with time-varying boundary conditions, which require nonlinear solves, have also been extensively studied [Smith et al. 2018; Modi et al. 2021; Brown and Narain 2021].

We accelerate the convergence of these nonlinear time integration schemes by combining the frozen factorization approach [Bank and Rose 1981; Deuflhard 2011] and the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [Nocedal 1980]. With the frozen factorization approach, the Hessian matrix is kept constant during the nonlinear solve, which can be advantageous since it is often expensive to form and factor the Hessian. Like other quasi-Newton methods, as long as the gradient vector is

computed exactly, using an approximate Hessian only affects the convergence rate and not the final solution. Unfortunately, with the frozen factorization approach, the number of iterations can become too large, even though each iteration is quick. We therefore use the L-BFGS algorithm to prevent the frozen Hessian from going stale. At the beginning of each nonlinear solve, we form and factor the Hessian matrix at the current state of the system. Then during each step of the nonlinear solve, we use the latest system information to store and apply the implicit updates to the frozen Hessian.

Our method is a general, practical method to accelerate nonlinear time integration for deformable object simulations. The efficiency advantage provided by our approach is the greatest when the required number of nonlinear iterations is high. In Sec. 4 we show this experimentally by changing various scene parameters (*e.g.,* anisotropy, heterogeneity, collisions, time step) and measuring the corresponding change in the wallclock time. In some cases, we obtain a speedup of 7.9x compared to the baseline method.

2 RELATED WORK

There are many existing methods for improving the efficiency of simulation-based techniques.

Subspace dynamics [Pentland and Williams 1989; Barbič and James 2005; Choi and Ko 2005; Fulton et al. 2019; Lan et al. 2020] or condensation [Guyan 1965; Teng et al. 2015; Mitchell et al. 2016; Weidner et al. 2020] can achieve impressive speedups, but they inevitably introduce errors in the resulting motion. The focus of our paper is on improving computational efficiency without changing the model of the underlying system. Furthermore, unlike some of these existing techniques, our method does not require any precomputation of modes or basis functions. The recent work by Fulton et al. [2019] uses an autoencoder to compute the latent space, and they also use a combination of matrix factorization and L-BFGS to accelerate the nonlinear solve. We use a similar idea, but in the full, unreduced deformation space. Li et al. [2019] also use the same idea for domain-decomposed time integrator with impressive results. We do not use domains, but we also introduce a simple way to make the approach collision-aware.

Unlike these reduction based methods, multigrid methods are capable of producing massive speedups for the full, non-reduced problem [McAdams et al. 2011; Tamstorf et al. 2015; Xian et al. 2019]. However, non-trivial extensions are necessary to support complex materials involving heterogeneity and anisotropy [Chen et al. 2019]. Also, no previous work has simultaneously demonstrated collisions and higher-order time integration schemes.

Since our method is based on factoring a global matrix and reusing its decomposition, it is similar in flavor to Projective Dynamics (PD), ADMM, and other global-local methods [Bouaziz et al. 2014; Narain et al. 2016; Liu et al. 2017; Peng et al. 2018; Zhang et al. 2019; Brown and Narain 2021]. In particular, the quasi-Newton approach of Liu et al. [2017] is highly related to our paper, as they also use L-BFGS to accelerate the convergence. In the context of PD, their method is clearly the fastest method around, since they require only one factorization at the beginning of the time step, and during the rest of the simulation, they only require forward/backward solves and L-BFGS updates. Furthermore, their L-BFGS updates allow them to efficiently account for any nonlinear material models. Outside of the PD framework, however, their initial Hessian becomes not applicable. They also did not show any results involving collisions between multiple dynamic objects, self collisions, or friction.

Position-based Dynamics (PBD) is an attractive option for real-time multi-physics simulation that is capable of modeling everything from rigid and elastic bodies to fluids [Müller et al. 2007]. Many features have been added over the years, including new material models, new constraints, and more sophisticated integrators [Bender et al. 2017]. However, PBD is a fundamental departure from classical methods, and cannot be readily deployed without changing the core of an existing simulator.

Similar to our work, Cholesky factorizations have been re-used or updated in geometry processing, but they either apply to local solutions [Herholz et al. 2017; Herholz and Alexa 2018], or update based on changing boundary conditions [Herholz and Sorkine-Hornung 2020]. These methods are efficient for nonlinear solves involving a subset of vertices but are inefficient for nonlinear solves involving all of the vertices of the deformable objects. They further assume that the underlying energy is Laplacian. Several simulation works have also attempted to use updated factorizations, but either only apply to linear [James and Pai 1999] or co-rotational materials [Hecht et al. 2012; Mitchell et al. 2016].

3 METHODS

In this section, we describe our approach, which we call QLB: a **Q**uasi-Newton approach that uses the Cholesky factor **L** and L-**B**FGS updates.[1] First, in §3.1, we describe the general formulation of the nonlinear solver. Then in §3.2, we describe how we use Cholesky and L-BFGS to efficiently compute the search direction. Finally, in §3.3, we describe how we handle dynamic collisions within this framework.

3.1 Background

Our approach is applicable to a wide range of nonlinear integrators, but for brevity we use BDF1 [Hairer et al. 2006] as a concrete example throughout this section. Let $\mathbf{x}$ and $\mathbf{v}$ be the nodal positions and velocities of the volumetric solid, and $\mathbf{M}$ be the constant mass matrix. Furthermore, let $\mathbf{f}(\mathbf{x}, \mathbf{v})$ be the force vector, and derivatives $\mathbf{D} = d\mathbf{f}/d\mathbf{v}$ and $\mathbf{K} = d\mathbf{f}/d\mathbf{x}$ be the damping and stiffness matrices, respectively. We use a trailing superscript with parenthesis to denote the time step. Thus, the goal of each integrator is to compute the nodal positions $\mathbf{x}^{(k+1)}$ given $\mathbf{x}^{(k)}$. After computing $\mathbf{x}^{(k+1)}$, we compute the nodal velocities $\mathbf{v}^{(k+1)}$ using the discretization scheme of the particular integrator.

With BDF1 (see supplemental material for other integrators), we solve a nonlinear system to advance the state from step k to $k+1$:

$$\mathbf{M}\mathbf{v}^{(k+1)} = \mathbf{M}\mathbf{v}^{(k)} + h\mathbf{f}^{(k+1)} \tag{1a}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + h\mathbf{v}^{(k+1)}, \tag{1b}$$

where h is the time step. We substitute Eq. 1a into Eq. 1b to arrive at the following equation:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + h\left(\mathbf{v}^{(k)} + h\mathbf{M}^{-1}\mathbf{f}^{(k+1)}\right). \tag{2}$$

[1] QLB can also mean "Quick Like a Bunny."

We now solve for a root $\mathbf{x}^{(k+1)}$ that satisfies the nonlinear system. After some rearranging (see supplemental material), the Newton search direction is then $\Delta\mathbf{x}_i = -\mathbf{H}_i^{-1}\mathbf{g}_i$, with

$$\begin{aligned} \mathbf{g}_i &= \mathbf{M}\left(\mathbf{x}_i^{(k+1)} - \mathbf{x}^{(k)} - h\mathbf{v}^{(k)}\right) - h^2\mathbf{f}_i^{(k+1)} \\ \mathbf{H}_i &= \mathbf{M} - h\mathbf{D}_i^{(k+1)} - h^2\mathbf{K}_i^{(k+1)}, \end{aligned} \tag{3}$$

where i indicates the current iteration of the nonlinear solve.

3.2 Accelerated Nonlinear Solver

One way to accelerate the nonlinear solver is the frozen Cholesky approach [Bank and Rose 1981; Deuflhard 2011]. This approach improves the efficiency of nonlinear solvers by forming and factoring the system matrix $\mathbf{H}$ only periodically and reusing the frozen factorization $\mathbf{L} = \text{chol}(\mathbf{H})$ multiple times, even when $\mathbf{H}$ changes during the nonlinear solver iterations. To compute the search direction, only forward/backward solves are required: $\Delta\mathbf{x}_i = -\mathbf{L}^{-\top}\mathbf{L}^{-1}\mathbf{g}_i$. The main difficulty with the frozen Cholesky approach, however, is that the $\mathbf{L}$ factor can become stale over time—*i.e.,* $\mathbf{L}\mathbf{L}^\top$ is no longer a good approximation of $\mathbf{H}$.

The L-BFGS algorithm is an ideal candidate to overcome this difficulty [Nocedal 1980]. Since we are performing a root-finding operation, we can view the Newton solve as optimizing some function with gradient $\mathbf{g}$ and Hessian $\mathbf{H}$. L-BFGS applies corrections to the inverse Hessian implicitly, so that the computation of the line search takes into account the curvature information obtained during the nonlinear iterations. While this approach has also been used previously for Projective Dynamics by Liu et al. [2017] and for reduced dynamics by Fulton et al. [2019], in this paper, we apply L-BFGS in the context of classical nonlinear integrators for scenes with collisions between dynamic objects, self collisions, and frictional contact.

We use the standard, two-loop recursion version of the L-BFGS algorithm [Nocedal 1980; Nocedal and Wright 2006]. For clarity, we divide the algorithm into two stages: LBFGSUPDATE and LBFGSSOLVE, shown in Alg. 1 and Alg. 2, respectively. The parameter m controls the number of past steps to use to compute the correction to the Hessian. Given m, we store a pair of vectors $\{\mathbf{s}_i, \mathbf{y}_i\}$ and a scalar ρ_i from each of the past m iterations. In LBFGSUPDATE, we take the current and previous values of $\mathbf{x}$ and $\mathbf{g}$, and then update $\mathbf{s}$, $\mathbf{y}$, and ρ, throwing away old values from m iterations ago. In LBFGSSOLVE, we use the stored values to compute $\mathbf{z} = \bar{\mathbf{H}}^{-1}\mathbf{b}$, where $\bar{\mathbf{H}}^{-1}$ is the implicitly updated inverse Hessian. Computationally, the only bottleneck is the forward/backward solve in line 5, since the two for-loops contain only vector dot products and scalar operations. Importantly, unlike with the original BFGS algorithm, the actual rank-1 updates to the inverse Hessian are never constructed, which would be prohibitively expensive since they would be fully dense. If $m = 0$, this algorithm reverts back to the frozen Cholesky approach, since the two for-loops would not be executed.

Algorithm 1 Stores the necessary information for implicitly updating $\mathbf{H}^{-1}$ with L-BFGS. Both $\mathbf{s}$ and $\mathbf{y}$ are lists of vectors $\in \mathbb{R}^n$, and ρ is a list of scalars. All three lists have length m. Here we assume that elements of these lists have been filled up to $m-1$.

1: **procedure** LBFGSUPDATE($\mathbf{x}, \mathbf{x}_{\text{prev}}, \mathbf{g}, \mathbf{g}_{\text{prev}}$)
2: $\quad\mathbf{s}_m = \mathbf{x} - \mathbf{x}_{\text{prev}}$
3: $\quad\mathbf{y}_m = \mathbf{g} - \mathbf{g}_{\text{prev}}$
4: $\quad\rho_m = (\mathbf{s}_m^\top \mathbf{y}_m)^{-1}$

Algorithm 2 Computes $\mathbf{z} = \bar{\mathbf{H}}^{-1}\mathbf{b}$, where $\bar{\mathbf{H}}^{-1}$ is the implicitly updated inverse Hessian.

1: **procedure** LBFGSSOLVE($\mathbf{b}, \mathbf{L}$) → ($\mathbf{z}$)
2: $\quad$**for** $i = m, \cdots, 1$ **do**
3: $\quad\quad\alpha_i = \rho_i(\mathbf{s}_i^\top \mathbf{b})$
4: $\quad\quad\mathbf{b} \mathrel{-}= \mathbf{y}_i\alpha_i$
5: $\quad\mathbf{z} = \mathbf{L}^{-\top}\mathbf{L}^{-1}\mathbf{b}$
6: $\quad$**for** $i = 1, \cdots, m$ **do**
7: $\quad\quad\beta = \rho_i(\mathbf{y}_i^\top \mathbf{z})$
8: $\quad\quad\mathbf{z} \mathrel{+}= \mathbf{s}_i(\alpha_i - \beta)$

Algorithm 3 QLB solver for stepping from $\mathbf{x}^{(k)}$ to $\mathbf{x}^{(k+1)}$.

1: **procedure** QLB($\mathbf{x}^{(k)}$) → ($\mathbf{x}^{(k+1)}$)
2: $\quad$Compute $\mathbf{H}_{\text{P}}(\mathbf{x}^{(k)})$ ▷ Hessian depends on integrator
3: $\quad\mathbf{L}_{\text{P}} = \text{chol}(\mathbf{H}_{\text{P}})$ ▷ persistent Cholesky factor
4: $\quad\mathbf{x} = \mathbf{x}_{init}$ ▷ initial guess depends on integrator
5: $\quad$**while** not converged **do**
6: $\quad\quad$Compute $\mathbf{g}(\mathbf{x})$ ▷ gradient depends on integrator
7: $\quad\quad$LBFGSUPDATE($\mathbf{x}, \mathbf{x}_{\text{prev}}, \mathbf{g}, \mathbf{g}_{\text{prev}}$)
8: $\quad\quad\Delta\mathbf{x} = -$LBFGSSOLVE($\mathbf{g}, \mathbf{L}_{\text{P}}$) ▷ search direction
9: $\quad\quad$Line search for λ
10: $\quad\quad\mathbf{x} \mathrel{+}= \lambda\Delta\mathbf{x}$
11: $\quad\mathbf{x}^{(k+1)} = \mathbf{x}$

Alg. 3 shows our QLB nonlinear solver that combines Cholesky and L-BFGS, which can be applied to a variety of nonlinear time integration schemes. In line 2, we compute the Hessian matrix using the current vertex positions. (In §3.3, we explain the subscript 'P', which is for 'persistent'.) Then in line 3, we compute the Cholesky factor of the Hessian, which is passed into the L-BFGS solve function in line 8.

3.3 Dynamic Collisions

One of the bottlenecks in the QLB procedure is the Cholesky factorization (Alg. 3, line 3). And within the sparse Cholesky factorization routine, the two major steps are symbolic analysis and numeric factorization. With our implementation (which uses Cholmod) and our examples, we found that symbolic analysis can take up to 20% of the time of numerical factorization. Therefore, it is highly advantageous to keep the sparsity the same if at all possible throughout the course of the simulation.

If a dynamic object collides only with a kinematic object (or a kinematically moving object), then the sparsity pattern remains constant. This is because the stiffness matrix entries from the penalty collision forces already exist in the stiffness matrix of the internal deformation forces. More formally, let $\mathbf{K}_{\text{ext}}$ and $\mathbf{K}_{\text{int}}$ be the stiffness matrices of external (collision) and internal (deformation) forces. With dynamic-static collisions, the non-zero entries of $\mathbf{K}_{\text{ext}}$ are a subset of the non-zero entries of $\mathbf{K}_{\text{int}}$. However, when there are dynamic-dynamic collisions (*e.g.,* between two dynamic objects or

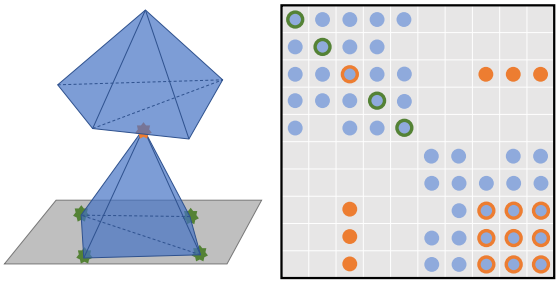


Figure 2: Collision between two bodies with 5 vertices each. The blue circles represent the non-zero entries from the elastic forces. (Each circle corresponds to a 3x3 block for a vertex.) Collisions between a body and a kinematic object (*e.g.,* ground) add entries only to existing blue entries (green outline). Dynamic collisions between two bodies (or self contact) add entries to existing blue entries (orange outline) and to new locations (orange solid). We call solid orange entries the *transient* matrix $\mathbf{H}_\mathrm{T}$ and all the other entries (solid blue, green outline, and orange outline) the *persistent* matrix $\mathbf{H}_\mathrm{P}$.

self collisions), then the non-zero entries of $\mathbf{K}_{\text{ext}}$ are no longer a subset of the non-zero entries of $\mathbf{K}_{\text{int}}$.

As a simple example, consider a collision between a vertex from one object and a triangle from another object, as shown in Fig. 2. The two objects have five vertices each, and the non-zero entries in the system matrix corresponding to these vertices are shown as blue circles, with each circle representing a 3x3 block of a vertex. One of the objects is colliding with a kinematic object (*e.g.,* ground), and the resulting non-zero entries are shown in green. For these kinematic collisions, no new non-zero entries are required. However, new non-zero entries are required for collisions between two dynamic objects or self collisions. The entries corresponding to these types of collisions are shown in orange. Some of the entries are added to existing non-zero entries (orange outline), but the cross-coupling terms are added to new locations in the matrix (orange solid). We partition $\mathbf{H}$ into *persistent* and *transient* matrices: $\mathbf{H} = \mathbf{H}_\mathrm{P} + \mathbf{H}_\mathrm{T}$. The transient matrix contains just the cross-coupling entries (orange solid), and the persistent matrix contains all of the other entries (blue solid, green outline, and orange outline).

Through experimentation, we found that ignoring $\mathbf{H}_\mathrm{T}$ and simply using $\mathbf{H}_\mathrm{P}$ gives the best overall result in terms of wallclock time. Even though using $\mathbf{H} = \mathbf{H}_\mathrm{P} + \mathbf{H}_\mathrm{T}$ results in fewer iterations, the added overhead of symbolic factorization cannot be overcome. For the Ducks scene (§4.4), using $\mathbf{H} = \mathbf{H}_\mathrm{P} + \mathbf{H}_\mathrm{T}$ resulted in a wallclock time 11% slower than simply using $\mathbf{H}_\mathrm{P}$. Additionally, ignoring $\mathbf{H}_\mathrm{T}$ leads to a simpler implementation.

Intuitively, what is the error introduced by ignoring the transient matrix $\mathbf{H}_\mathrm{T}$? Ignoring the transient and only using the persistent component removes the cross coupling terms from the stiffness and damping matrices, which correspond to the colliding portions of the deformable object(s). When we do not take into account the cross coupling terms, we are in fact assuming that the collision involves a static object—each body assumes that the other body is static. Note, however, that this "error" *only affects the speed of convergence and not the final solution.*

4 RESULTS

We implemented our system in C++ and ran the simulations on a desktop with an Intel Core i7-7700 CPU @ 3.6 GHz and 16 GB of RAM. We use Eigen for dense linear algebra and Cholmod with MKL for sparse linear factorizations and solves [Guennebaud et al. 2010; Chen et al. 2008]. We use the Stable Neo-Hookean material as the base isotropic material [Smith et al. 2018] with strain rate damping [Sánchez-Banderas and Otaduy 2018]. For all materials, we clamp the eigenvalues to ensure that the Hessian is semidefinite [Smith et al. 2018, 2019; Kim et al. 2019]. We parallelize the matrix fill with OpenMP and use multithreaded versions of the libraries. We compare the following solvers:

- ND (**N**ewton-**D**irect): A full Newton method where we form the exact Hessian every iteration and use a direct method (Cholmod) to solve the linear system.
- QLF (**Q**uasi Cho**L**esky **F**rozen): A quasi-Newton method where we form and factor the Hessian at the beginning of the time step, and then use this frozen factor for the duration of the time step.
- QLB (ours: **Q**uasi Cho**L**esky L-**B**FGS): A quasi-Newton method where we form and factor the Hessian at the beginning of the time step, and update the Hessian implicitly with L-BFGS.

We also tried a PCG-based Newton solver with block Jacobi preconditioner and early termination, but its performance was significantly worse than ND for the stiff armadillo example, so we did not include it in our results. However, for very large meshes with lower stiffness, a carefully crafted matrix-free PCG implementation may outperform ND. Additionally, we experimented with using a scaled identity as the Hessian approximation in L-BFGS (line 5 in Alg. 2) [Nocedal 1980], but the simulation quickly became unstable. We use 1e-6 as the relative and absolute tolerance for the Newton and quasi-Newton solvers. A detailed list of parameters is shown in Table 1. Screenshots of the scenes are shown in Fig. 1, and the animations are available in the supplemental video. We also include a supplemental document with the iteration information for the experiments. Our QLB method consistently performs the best.

	verts	tets	mass
Bar	10,125	51,744	3.00
Arma	25,317	98,486	1.37
Bunny	3,907	14,374	2.25
Ducks	12,800	36,800	1.31

4.1 Bar

Our first experiment is a bar undergoing time-varying boundary conditions. Over a period of 5 simulation seconds, the bar is compressed, stretched, bent, and twisted by rigidly moving the vertices of the free end of the bar. We vary several different parameters to see the effect on the convergence of the nonlinear solvers.

4.1.1 L-BFGS storage. First, we modify m, the number of past steps that are used for the L-BFGS update. We run the same Bar scene with the default parameters, with m set to 2, 5, 10, and 20, and the overall wallclock times are shown in Fig. 4a. As reported in the literature [Nocedal and Wright 2006], only a modest number is required, with $m = 10$ giving us the best results.

Table 1: Scene parameters. *vary*: scene parameter to vary. *material*: material model. E: Young's modulus (Pa). μ: Poisson's ratio. *damping*: damping coefficient. h: time step (s). *integrator*: time integrator scheme. m: L-BFGS corrections.

	vary	material	E	μ	damping	h	integrator	m
Bar	E	SNH	6e3, 2e4, 5e4, 1e5	0.49	5e-3	5e-2	BDF2	10
Bar	μ	SNH	2e4	0.42, 0.49	5e-3	5e-2	BDF2	10
Bar	hetero	SNH	6e4, 2e4	0.49	5e-3	5e-2	BDF2	10
Bar	aniso	aSTVK+SNH	3e4+2e4	0.49	5e-3	5e-2	BDF2	10
Bar	integrator	SNH	2e4	0.49	5e-3	5e-2	quas., BDF1, BDF2	10
Bar	m	SNH	2e4	0.49	5e-3	5e-2	BDF2	2, 5, 10, 20
Arma	damping	SNH	1e7	0.49	0, 1e-6	1e-2	BDF2	10
Arma	material	STVK, SNH	1e7	0.49	0	1e-2	BDF2	10
Arma	h	SNH	1e7	0.49	0	5e-3, 1e-2, 2e-2	BDF2	10
Bunny	-	SNH	8e4	0.49	1e-3	5e-3	SDIRK2	10
Ducks	-	SNH	5e4	0.49	4e-3	1e-2	SDIRK2	10

4.1.2 Young's Modulus. We run the simulation with the Young's modulus values of 6e3, 2e4, 5e4, and 1e5. As shown in Fig. 3a, our QLB method works well across a range of stiffness values. With the highest Young's modulus value of 1e5, our QLB is 7.9x faster than ND. When the Bar gets stiffer, the wallclock time of QLB increases more slowly than the wallclock time of both ND and QLF.

4.1.3 Poisson's Ratio. Next we change the Poisson's ratio to control the amount of volume preservation. The resulting wallclock times are shown in Fig. 3b. For all three methods, the wallclock time is shorter when the Poisson's ratio is lower. The relative performance gain by going from the default value of $\mu = 0.49$ down to $\mu = 0.42$ is modest for QLB, but it still beats the baseline methods.

4.1.4 Heterogeneity. We repeat the simulation with a heterogeneous material distribution, alternating soft and stiff materials along the length of the bar. When heterogeneity is added, the wallclock times of all three methods increase due to the added nonlinearity. As shown in Fig. 3c, QLB is the fastest of the three methods despite the added nonlinearity.

4.1.5 Anisotropy. We use anisotropic fibers within the Bar to introduce nonlinearities. On top of the background SNH material, we add the anisotropic StVK material [Kim et al. 2019] to model the fibers. The added complexity makes the three methods noticeably slower across the board, however as seen in Fig. 3d, the relative slowdown for QLB is the smallest of the three methods.

4.1.6 Integrator. Finally, with the same default parameters, we switch out the time integration scheme. In addition to the default integrator of BDF2, we use BDF1 and quasistatics. The wallclock times for all three integrators are shown in Fig. 3e. The times for BDF2 and BDF1 are similar. The conditioning for the quasistatics integrator is worse, leading to slower wallclock times. QLB has the smallest relative slowdown when using the quasistatics integrator.

4.2 Arma

Next we simulate the Armadillo with BDF2 for 1 second. The stiff Arma is attached to a base by its feet, and the base is moved kinematically to induce motion.

4.2.1 Damping. We first add damping. Since we use a higher-order integrator, the added damping is much more controllable and does not depend heavily on the time step (see below). Fig. 3f shows the wallclock time for the default and damped Arma. QLB works well both with and without damping. For the Arma scene with the default parameters, QLB is 3.1x faster than ND and 3.9x faster than QLF.

4.2.2 Material. We now change the material model of the object. In addition to the default material of SNH, we use the StVK material. The wallclock times for each material are shown in Fig. 3g. For both materials used, QLB results in the fastest wallclock time.

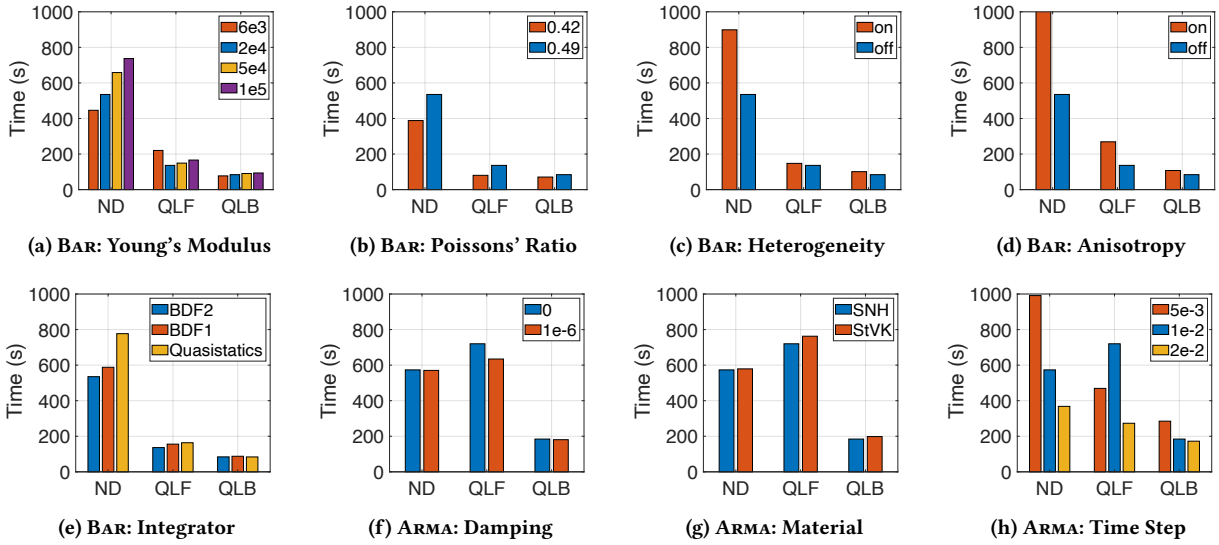
4.2.3 Time step. Using BDF2, we use half and double the default time step of h = 1e-2. At the high time step, the resulting simulation is slightly more damped, due to the small amount of numerical damping in BDF2. As seen in Fig. 3h, the speedup gained by QLB is not as large for the high time step, but QLB still results in the fastest wallclock times for all three step sizes.

4.3 Bunny

We show self collisions with a Bunny with the SDIRK2 integrator for a 3 second scene. We use the contact model by Geilinger et al. [2020] for collisions with the floor, and the contact model by McAdams et al. [2011] for self collisions. As shown in the supplemental material, the SDIRK2 integrator requires two nonlinear solves per time step. However, unlike BDF2, it is a single-step method and is potentially more suitable for simulations with contact [Löschner et al. 2020] (though Geilinger et al. [2020] do use BDF2). We use a relatively small time step of h = 5e-3 to deal with the thin ears. The overall wallclock times are shown in Fig. 4b. For this 3 second simulation, QLB has a 4.5x speedup over ND. QLF failed to converge and thus is not included for this simulation.

4.4 Ducks

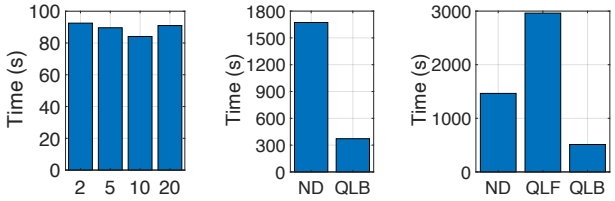
For the final scene, we simulate 100 contacting Ducks with the SDIRK2 integrator for 2 seconds. For added stability, in addition to Green damping (Table 1), we also added some velocity-based contact damping forces. Unlike the BDF1 integrator, adding damping forces is much more controllable, since changing the time step does not add significant artificial damping. We use all three methods for this scene, with a large time step of h = 1e-2. The overall wallclock



(a) Bar: Young's Modulus (b) Bar: Poissons' Ratio (c) Bar: Heterogeneity (d) Bar: Anisotropy

(e) Bar: Integrator (f) Arma: Damping (g) Arma: Material (h) Arma: Time Step

Figure 3: Wallclock times for Bar with: (a) different Young's moduli; (b) different Poisson's ratios; (c) heterogeneity on/off; (d) anisotropy on/off; (e) different integration schemes; and Arma with: (f) different damping coefficients; (g) different materials; (h) different time steps. In all the plots, the Bar and Arma with the default parameters are shown in blue.



(a) Bar: m (b) Bunny (c) Ducks

Figure 4: Wallclock times for (a) different numbers of past steps used for the L-BFGS update for the default Bar scene; (b) Bunny; and (c) Ducks.

times are shown in Fig. 4c. For this 2 second simulation, QLB is 2.9x faster than ND and 5.8x faster than QLF.

5 CONCLUSION

We showed through a variety of examples that a straightforward application of Cholesky and L-BFGS gives excellent performance when used in the nonlinear solver for higher-order time integration. We showed that our approach can handle a wide range of complex material models, including anisotropy and heterogeneity, as well as collisions between dynamic objects, self collisions, and frictional contact. In some simulations, QLF performs worse than ND. This is due to QLF not always converging quickly, and it is hard to predict when this will happen. *QLB does not suffer from this problem and performed the best in every test case.* Our approach is practical, general, and easy to implement.

5.1 Discussions & Limitations & Future Work

In our work, we only reuse the factorization within a single time step, but it can also be used across time steps. There are standard methods to detect when a re-factorization is required, such as maximum Newton iterations or the change in the norm of residual across time steps [Bank and Rose 1981]. In this work, rather than reusing the factorization, we take larger time steps when possible, since our approach supports higher-order integrators that do not significantly suffer from artificial damping.

We use a simple back-tracking strategy for our line search, which may not always produce stable L-BFGS updates [Nocedal and Wright 2006]. Though we have not encountered any issues with our examples, more sophisticated strategies may be needed in other cases.

Our approach currently does not support hard constraints. Accelerating the quadratic programming approach of Löschner et al. [2020] using our approach is an interesting avenue of future work.

One advantage of higher-order schemes is that damping becomes more controllable. It would be interesting to apply our approach to an adaptive time stepping scheme, since adaptively changing the time step would not overly introduce numerical damping.

If implemented properly, GPU solvers can often be considerably faster than their CPU counterparts. Since direct solvers have been successfully deployed on the GPU [Rennich et al. 2016], our method can also be ported to the GPU. We leave this as future work.

ACKNOWLEDGMENTS

This work was sponsored in part by the National Science Foundation (CAREER-1846368).

REFERENCES

Randolph E Bank and Donald J Rose. 1981. Global approximate Newton methods. *Numer. Math.* 37, 2 (1981), 279–295.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Annual Conference Series (Proc. SIGGRAPH)*. 43–54.

Jernej Barbič and Doug L. James. 2005. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Trans. Graph.* 24, 4 (July 2005), 982–990.

Jan Bender, Matthias Müller, and Miles Macklin. 2017. A Survey on Position Based Dynamics. In *Proceedings of the Eurographics: Tutorials.* Article 6.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014).

George E. Brown and Rahul Narain. 2021. WRAPD: Weighted Rotation-aware ADMM for Parameterization and Deformation. *ACM Trans. Graph.* 40, 4 (July 2021).

Jiong Chen, Max Budninskiy, Houman Owhadi, Hujun Bao, Jin Huang, and Mathieu Desbrun. 2019. Material-Adapted Refinable Basis Functions for Elasticity Simulation. *ACM Trans. Graph.* 38, 6, Article 161 (Nov. 2019).

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.* 35, 3, Article 22 (Oct. 2008).

Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal Warping: Real-Time Simulation of Large Rotational Deformation and Manipulation. *IEEE TVCG* 11, 1 (Jan. 2005), 91–101.

Peter Deuflhard. 2011. *Newton methods for nonlinear problems: affine invariance and adaptive algorithms.* Vol. 35. Springer Science & Business Media.

Elliot English and Robert Bridson. 2008. Animating Developable Surfaces Using Nonconforming Elements. *ACM Trans. Graph.* 27, 3, Article 66 (Aug. 2008).

Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. *Computer Graphics Forum (Proc. Eurographics)* (2019).

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. 2020. ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact. *ACM Trans. Graph.* 39, 6, Article 190 (Nov. 2020).

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Robert J Guyan. 1965. Reduction of stiffness and mass matrices. *AIAA journal* 3, 2 (1965), 380–380.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations.* Vol. 31. Springer Science & Business Media.

Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. 2012. Updated Sparse Cholesky Factors for Corotational Elastodynamics. *ACM Trans. Graph.* 31, 5, Article 123 (Sept. 2012).

Philipp Herholz and Marc Alexa. 2018. Factor Once: Reusing Cholesky Factorizations on Sub-Meshes. *ACM Trans. Graph.* 37, 6, Article 230 (Dec. 2018).

Philipp Herholz, Timothy A. Davis, and Marc Alexa. 2017. Localized Solutions of Sparse Linear Systems for Geometry Processing. *ACM Trans. Graph.* 36, 6, Article 183 (Nov. 2017).

Philipp Herholz and Olga Sorkine-Hornung. 2020. Sparse Cholesky Updates for Interactive Mesh Parameterization. *ACM Trans. Graph.* 39, 6, Article 202 (Nov. 2020).

Doug L. James and Dinesh K. Pai. 1999. ArtDefo: Accurate Real Time Deformable Objects. In *Proceedings of SIGGRAPH.* 65–72.

Theodore Kim, Fernando De Goes, and Hayley Iben. 2019. Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation. *ACM Trans. Graph.* 38, 4, Article 69 (July 2019).

Theodore Kim and David Eberle. 2020. Dynamic Deformables: Implementation and Production Practicalities. In *ACM SIGGRAPH 2020 Courses.* Article 23.

Lei Lan, Ran Luo, Marco Fratarcangeli, Weiwei Xu, Huamin Wang, Xiaohu Guo, Junfeng Yao, and Yin Yang. 2020. Medial Elastics: Efficient and Collision-Ready Deformation via Medial Axis Transform. *ACM Trans. Graph.* 39, 3, Article 20 (April 2020).

Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed Optimization Time Integrator for Large-step Elastodynamics. *ACM Trans. Graph.* 38, 4, Article 70 (July 2019), 70:1–70:10 pages.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 4, Article 116 (May 2017).

Fabian Löschner, Andreas Longva, Stefan Jeske, Tassilo Kugelstadt, and Jan Bender. 2020. Higher-Order Time Integration for Deformable Solids. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.* Article 15.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.* 30, 4, Article 37 (July 2011).

Nathan Mitchell, Michael Doescher, and Eftychios Sifakis. 2016. A Macroblock Optimization for Grid-Based Nonlinear Elasticity. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.* (Zurich, Switzerland). 11–19.

Vismay Modi, Lawson Fulton, Alec Jacobson, Shinjiro Sueda, and David I. W. Levin. 2021. EMU: Efficient Muscle Simulation in Deformation Space. *Computer Graphics Forum* 40, 1 (2021), 234–248.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Rahul Narain, Matthew Overby, and George E. Brown. 2016. ADMM ⊇ Projective Dynamics: Fast Simulation of General Constitutive Models. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.* 21–28.

Jorge Nocedal. 1980. Updating quasi-Newton matrices with limited storage. *Mathematics of computation* 35, 151 (1980), 773–782.

Jorge Nocedal and Stephen Wright. 2006. *Numerical optimization.* Springer Science & Business Media.

Yue Peng, Bailin Deng, Juyong Zhang, Fanyu Geng, Wenjie Qin, and Ligang Liu. 2018. Anderson Acceleration for Geometry Optimization and Physics Simulation. *ACM Trans. Graph.* 37, 4, Article 42 (July 2018).

Alex Pentland and John Williams. 1989. Good Vibrations: Modal Dynamics for Graphics and Animation, Vol. 23. ACM, New York, NY, USA, 207–214.

Steven C. Rennich, Darko Stosic, and Timothy A. Davis. 2016. Accelerating sparse Cholesky factorization on GPUs. *Parallel Comput.* 59 (2016), 140–150.

Rosa M. Sánchez-Banderas and Miguel A. Otaduy. 2018. Strain rate dissipation for elastic deformations. In *Computer Graphics Forum (Proc. Symposium on Computer Animation)*, Vol. 37. 161–170.

Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Trans. Graph.* 37, 2, Article 12 (March 2018).

Breannan Smith, Fernando De Goes, and Theodore Kim. 2019. Analytic Eigensystems for Isotropic Distortion Energies. *ACM Trans. Graph.* 38, 1, Article 3 (Feb. 2019).

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (Oct. 2015).

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace Condensation: Full Space Adaptivity for Subspace Deformations. *ACM Trans. Graph.* 34, 4, Article 76 (July 2015).

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically Deformable Models. In *Computer Graphics (Proc. SIGGRAPH)*, Vol. 21. 205–214.

Nicholas J. Weidner, Theodore Kim, and Shinjiro Sueda. 2020. ConJac: Large Steps in Dynamic Simulation. In *Motion, Interaction and Games.* ACM, Article 6.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-time Simulation of Deformable Objects. *ACM Trans. Graph.* 38, 6, Article 162 (Nov. 2019).

Hongyi Xu and Jernej Barbič. 2017. Example-Based Damping Design. *ACM Trans. Graph.* 36, 4, Article 53 (July 2017).

Juyong Zhang, Yue Peng, Wenqing Ouyang, and Bailin Deng. 2019. Accelerating ADMM for efficient simulation and optimization. *ACM Trans. Graph.* 38, 6, Article 163 (Nov. 2019).