Surface-Only Dynamic Deformables using a Boundary Element Method

R. Sugimoto† C. Batty† T. Hachisuka†
University of Waterloo, Canada

Abstract
We propose a novel surface-only method for simulating dynamic deformables without the need for volumetric meshing or volumetric integral evaluations. While based upon a boundary element method (BEM) for linear elastodynamics, our method goes beyond simple adoption of BEM by addressing several of its key limitations. We alleviate large displacement artifacts due to linear elasticity by extending BEM with a moving reference frame and surface-only fictitious forces, so that it only needs to handle deformations. To reduce memory and computational costs, we present a simple and practical method to compress the series of dense matrices required to simulate propagation of elastic waves over time. Furthermore, we explore a constraint enforcement mechanism and demonstrate the applicability of our method to general computer animation problems, such as frictional contact.

CCS Concepts
- Computing methodologies → Physical simulation;

1. Introduction
Physical simulation of dynamically deforming elastic objects (elastodynamics) is now widely deployed for computer animation. Most such simulation methods rely on discretizing an object into a volumetric mesh of tetrahedral or cubic elements. The simulation then processes both the exterior (surface) and the interior of the object. Computer animation, however, is often concerned only with the visible motion of the object’s surface. We propose a practical surface-only dynamic deformables simulation method (Fig. 1) suitable for computer animation. Being a surface-only method, our method does not need any volumetric discretization.

Our method builds upon advances in the boundary element method (BEM) [LMN*12]. BEM can solve partial differential equations (PDEs) while discretizing only the boundary (surface) of the domain. James et al. [JP99] applied BEM for the first time to computer animation based on linear elastostatics. Although BEM is becoming increasingly popular in computer animation [HW15; HW16; DHB*16], we are the first to solve elastodynamics simulation with BEM in computer animation.

BEM for elastodynamics is an active area of research even in computational mechanics [LMN*12]. The intuition behind elastodynamics BEM is that we can express deformation at a point on a surface by considering a superposition of elastic waves propagated from all surface points over time. Mathematically, we achieve this by solving a boundary integral equation (BIE) derived from the Navier-Cauchy equations of linear elasticity. Since this BIE is an integral over the object’s surface, only the surface needs to be discretized. Unlike the BIE for elastostatics [JP99], this BIE for elastodynamics involves convolution over time to account for the influence of the past motion, leading to a recursive matrix equation after discretization.

† email: {rsugimoto|c2batty|thachisu}@uwaterloo.ca

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.
We observed that simply employing BEM for elastodynamics would not result in a practical method. The first problem is that directly solving the BIE in a static frame results in artifacts due to the limitations of linear elasticity, such as volume gain under large rotations. We introduce a **moving frame** of reference and additional fictitious forces in BEM. This approach ensures that BEM only needs to handle local deformations, enabling large translation and rotation and leading to better numerical stability. Even compared to elastostatic methods that incorporate a moving rigid frame [HW16; JP02], adding dynamics requires us to pay additional attention to ensure the method’s stability. The second problem is that, unlike sparse matrices in volumetric simulations, the matrices in BEM are **dense**. In general, the computation and storage costs for dense matrices exhibit worse scaling compared to sparse matrices. We propose a compression method tailored to the dense matrices in BEM for elastodynamics. This compression method significantly reduces both the computational and storage costs by exploiting the smoothness of the integrands in the BIE. Lastly, we incorporate a constraint enforcement strategy that enables frictional contact and domain decomposition in order to support more complex scenarios. Our method thus goes beyond a simple application of BEM to elastodynamics animation. To summarize, our contributions are:

- an application and adaptation of elastodynamics BEM to computer animation problems,
- support for large global displacements for elastodynamics BEM using a moving body frame and fictitious forces,
- a simple matrix compression technique to reduce memory cost and computational cost,
- a constraint-based treatment of joints, frictional contact, and domain decomposition for our BEM scheme.

## 2. Related Work

**Physics-based deformable body simulation.** Physics-based simulations of deformable bodies typically solve partial differential equations with volumetric spatial discretizations. The pioneering work of Terzopoulos et al. [TPBF87] uses a finite difference method to discretize the space, whereas Teran et al. [TBNF03] applied the finite volume method. The finite element method (FEM) [SB12] later became increasingly popular due to its generality and strong theoretical foundations. Smoothed particle hydrodynamics [KBST19] and the material point method [JST+16; HZG19] have also been applied to deformable body simulation. All of the above methods require discretization of and computation over the interior of the volumetric domain; we avoid this interior discretization via BEM.

**Geometry-based deformable body animation.** Geometry-based (or position-based) methods are common alternatives to physics-based simulations. These methods often have attractive properties for computer animation, such as computational efficiency and unconditional stability. Early work in this direction includes shape matching [MHTG05] and position-based dynamics [MHHR07]. These methods have been extended in many ways, and we refer to the survey by Bender et al. [BMM17] for an overview. In their basic forms, such methods often lack the accuracy and realism of physics-based methods. Furthermore, these approaches also often rely on volumetric meshes. Our BEM method is physics-based and does not assume having any volumetric meshes.

**BEM in graphics.** In contrast to the methods above, BEM solves linear partial differential equations with only a **surface discretization**. The potential benefits of BEM in computer animation were first recognized by James et al. [JP99] for elastostatics simulation. Several applications in geometry processing and physics-based simulation have subsequently been explored. For example, James et al. [JPB06] and Umetani et al. [UPSW16] used BEM for acoustic transfer problems by solving the Helmholtz equation. Several authors have used elastostatics BEM to simulate brittle fractures [ZBG15; HW15; HW16]. Da et al. [DHB+16] and Huang et al. [HM20] explored BEM for surface-only liquid animation. They solve the Laplace equation with a mixed boundary condition to perform the pressure solve. A range of applications of BEM to geometry processing tasks have also been studied [SVB17; WSSK13; LW16]. A key distinction of our work compared to prior graphics work is that we consider, for the first time, solving a **dynamic** problem with BEM. For example, while the surface-only liquids method [DHB+16] is overall solving a dynamic problem, its BEM step is applied only to solve the static Laplace problem. Our method directly solves a hyperbolic PDE with BEM, whereas prior work uses BEM to solve only elliptic PDEs.

**Elastodynamics BEM.** The first time-domain elastodynamics BEM for 3D transient problems was proposed by Banerjee et al. [BAM86] and Manolis et al. [MB88]. They solve the boundary integral equation using analytical integration over time, assuming constant nodal displacements and tractions over each time step. This early work suffered from “intermittent instability” effects [PS97], meaning that the solution becomes unstable in a manner that depends on the time step size in an erratic and unpredictable way. Several researchers proposed approaches to alleviate this issue, for example by employing different fundamental solutions or using higher order interpolation functions. Among them, Schanz and Antes [SA97; Sch01] claim that their formulation called *convolution quadrature BEM* (CQBEM) is less sensitive to the choice of time step size compared to the classical counterpart. Our initial experiments supported this claim, and we base our method upon CQBEM. Li et al. [LZ13] give a summary on this topic and also conclude that CQBEM numerically outperforms other representative options in terms of stability and accuracy. There exist more recent methods [BMS12; ADFG12], but with increased implementation complexities and we leave exploring them as future work.

**Fundamental solutions.** For a time-dependent differential equation, a fundamental solution is an analytical solution to the problem in the case of a source (or load) that is concentrated in both space and time. A fundamental solution assumes an infinite domain without any specific boundary conditions. De Goes et al. [DJ18] use elastodynamics fundamental solutions directly on the infinite ambient volume containing an object in order to approximate simple secondary motions. This approach, however, cannot handle the boundary conditions necessary for general elastodynamics simulations. Our method instead relies on BEM, which uses fundamental solutions in its derivation, but supports finite domains and general boundary conditions needed for elastodynamics simulations.
Linear elasticity and rotation effects. Currently, elastodynamics BEMs that avoid the need for computation over volumetric domain are based only on linear elasticity and thus cannot naturally handle large displacements. This fact has limited the type of dynamic motions one can simulate with BEM. James et al. [JP99] did not address this problem, nor has the computational mechanics community. In computer graphics, similar problems have been addressed (for non-BEM schemes) by coupling linear elasticity and rigid body dynamics [TW88] or a corotated linear strain measure for FEM [MG04]. Hahn et al. [HW16] augment a rigid body simulator with fracture effects due to collisions by solving instantaneous elastostatic BEM problems upon contact, assuming that deformations due to elasticity are almost negligible. The BEM traction field is computed from rigid body contact impulses. This method is not applicable to elastodynamics because it does not consider time-history effects. We address this issue by solving an elastodynamics BEM problem on a moving rigid frame.

3. Surface-Only Dynamic Deformables

Our surface-only dynamic deformables formulation begins by augmenting an elastodynamics BEM with a moving frame of reference undergoing rigid body dynamics. This moving reference frame can absorb effects due to large global rotational and translational displacements and the BEM needs to handle only the remaining local deformations. The motion of the frame imposes fictitious forces in the frame, which we can include as external forces in the BEM. In the following, we first introduce basic elastodynamics BEM along with our new fictitious forces (Section 3.1), and then describe how to apply the BEM within a moving frame (Section 3.2).

With this framework laid out, we introduce two further extensions to handle diverse animation scenarios. We discuss how to compress the dense matrices in our BEM method to reduce its computation and storage costs (Section 3.3). We then propose our constraint enforcement method to support joints, frictional contact effects, and domain decomposition for even larger deformations (Section 3.4). Fig. 2 shows the relation between the building blocks of our method. Our method consists of precomputation steps, which amortize costs for the simulation loop, and runtime steps, which carry out the simulation itself. We use a bold face font (i.e., \( \mathbf{u} \)) to denote vectors and matrices in the continuous setting and use a sans serif font (i.e., \( u \)) to denote vectors and matrices after discretization.

3.1. Elastodynamics BEM

Below we describe our chosen baseline time-domain elastodynamics BEM [SA97; Sch01], emphasizing differences with respect to elastostatics [JP99]. The most fundamental of these differences is the introduction of temporal convolutions. The original work by Schanz et al. [SA97] does not explain how to handle body forces such that the resulting BIE contains only surface integrals. If volume integrals remain, they might necessitate interior discretization which would defeat the purpose of BEM. We therefore explain how gravity and two types of fictitious forces from a moving frame can be incorporated into BEM as body forces using only surface integrals.

3.1.1. Boundary integral equation

Suppose that we have a homogeneous, isotropic, linearly elastic material with undeformed volumetric domain \( \Omega \) having boundary \( \Gamma = \partial \Omega \). The displacement field for the material, \( \mathbf{u}(\mathbf{x}, t) \), follows the Navier-Cauchy equations,

\[
\rho \nabla^2 \mathbf{u} + (\mu + \lambda) \nabla(\nabla \cdot \mathbf{u}) + \mathbf{b} = \rho \ddot{\mathbf{u}},
\]

where \( \lambda \) and \( \mu \) are Lamé parameters, \( \rho \) is a constant mass density, and \( \mathbf{b}(\mathbf{x}, t) \) is the external forces applied to the body, e.g., gravity. We omit the dependency on variables \( \mathbf{x} \) and \( t \) for brevity in the equation above. We can also derive the traction (i.e., force per unit area) applied on the surface, \( \mathbf{p}(\mathbf{x}, t) \), from \( \mathbf{u}(\mathbf{x}, t) \) by Cauchy’s theorem [Sch01]. The presence of the second order time derivative of displacement (indicated by double overdots), absent for elastostatics, makes the equation a hyperbolic PDE.

The problem is to solve Eq. (1) for \( \mathbf{x} \in \Omega, t \geq 0 \) under time-dependent traction boundary conditions and zero initial displacements and velocities:

\[
\begin{align*}
\mathbf{p}(\mathbf{x}, t) &= \overline{\mathbf{p}}(\mathbf{x}, t) \quad \text{for} \quad \mathbf{x} \in \Gamma, t \geq 0 \\
\mathbf{u}(\mathbf{x}, 0) &= \overline{\mathbf{u}}(\mathbf{x}, 0) = \mathbf{0} \quad \text{for} \quad \mathbf{x} \in \Omega,
\end{align*}
\]

where the overline indicates the prescribed boundary function. Other boundary conditions, such as displacements specified at vertices, can be handled using a constraint solver (see Section 3.4) or penalty springs [MZS*11]. A fundamental solution for displacements \( \mathbf{u}^*(\mathbf{x}, t) \) and traction \( \mathbf{p}^*(\mathbf{y}, t) \) to Eq. (1) can be derived analytically as responses at \( (\mathbf{x}, t) \) to a unit impulsive load at \( (\mathbf{y}, t) \) in an infinite medium.

With zero initial displacements and velocities, these fundamental solutions allow us to transform Eq. (1) into a BIE in the form of

\[
\begin{align*}
\mathbf{c}(\mathbf{x}) \mathbf{u}(\mathbf{x}, t) &= -\int_0^t \int_\Omega \mathbf{p}^*(\mathbf{y}, t - \tau) \mathbf{u}(\mathbf{y}, \tau) \, d\mathbf{y} \, d\tau \\
&\quad + \int_0^t \int_\Omega \mathbf{u}^*(\mathbf{y}, t - \tau) \mathbf{p}(\mathbf{y}, \tau) \, d\mathbf{y} \, d\tau \\
&\quad + \int_\Omega \mathbf{u}^*(\mathbf{y}, t) \mathbf{b}(\mathbf{y}, t) \, d\mathbf{y},
\end{align*}
\]

where \( \mathbf{c}(\mathbf{x}) \) is called the integral free term and is associated with the smoothness of the boundary in the undeformed configuration.

The main difference from the elastostatic case [JP99] is the introduction of a time integral from the beginning of the simulation to the current time. Intuitively, this integral expresses that elastic waves that propagate through the body from any boundary point \( \mathbf{y} \) are superposed in space and time and affect the displacement at \( \mathbf{x} \) at the current time. The last term in Eq. (3) expresses body force

![Figure 2: Overview of our method.](image-url)
effects and it involves a volume integral. We need to avoid discretizing this volume integral to have a surface-only method. This term is typically ignored in the computational mechanics literature.

When we use this boundary integral equation within a moving, non-inertial frame of reference, such a frame introduces additional forces on objects, called *fictitious forces*. Let us denote the translational acceleration of the frame by \( \mathbf{a} \), the rotational acceleration of the frame by \( \alpha \), the gravitational acceleration by \( \mathbf{g} \), expressed in the moving frame, respectively. We further denote a skew-symmetric cross-product matrix by \([ \cdot ]_c\), and the center of mass of the body in the undeformed configuration by \( \bar{x} \). With this notation, we can introduce forces due to translational acceleration \( -\rho \mathbf{a}(\tau) \) and rotational acceleration (the latter known as the Euler force) \( -\rho[y - \bar{x}]_c \alpha(\tau) \), omitting the centrifugal and Coriolis forces for simplicity. We explain more details regarding fictitious forces in a later subsection. We further include the gravitational force \( \rho \mathbf{g}'(\tau) \).

We incorporate these three forces into the BEM system as body force terms, i.e. \( \mathbf{b}(y, \tau) = -\rho \mathbf{a}(\tau) - \rho[y - \bar{x}]_c \alpha(\tau) + \rho \mathbf{g}'(\tau), \) computed based on the body’s undeformed shape. Fortunately, we can analytically convert the volume integral in these body force terms into integral free terms at the \( k^{th} \) time step, respectively. We observed that piecewise linear interpolation offers better stability than piecewise constant interpolation.

We then apply CQM (Eq. (6)) for temporal discretization and integration, and piecewise linear interpolation for spatial discretization (Eq. (7)) to each term in Eq. (3) with the body force terms described in Eq. (4). At the \( i^{th} \) vertex, we get

\[
e^{(t)}(x)u_{0,i} = \sum_{k=0}^{n_{max}} \left( -\int_0^T \left[ \int_\Gamma \left( \psi_k(\tilde{\mathbf{p}}'^*)(\mathbf{x}, \tau) \Phi(\mathbf{y}) \Phi(\mathbf{y}) d\Gamma \right) u_{0,k} - \int_0^T \left( \int_\Gamma \left( \psi_k(\tilde{\mathbf{u}}^* - \mathbf{q}')(\mathbf{x}, \tau) \Phi(\mathbf{y}) \Phi(\mathbf{y}) d\Gamma \right) p_{n,k} \right) \right) dt \right.
\]

where \( \tilde{\mathbf{p}}'^* \) is a Laplace transformed function, \( \psi_k \) is the nodal displacement 3-vector for the \( k^{th} \) vertex, and \( u_{0,i} \) is the nodal displacement 3-vector for the \( i^{th} \) vertex at time \( t = n\Delta t \). The ordering of terms in Eq. (3) after substitution of Eq. (4) is retained in Eq. (8) to expose how the CQM and interpolation were applied. We provide expressions for \( \tilde{\mathbf{p}}'^* \), \( \tilde{\mathbf{u}}^* \), \( \mathbf{d}^* \) and \( \mathbf{q}' \) in Appendix A.

We can precompute the boundary integrals in Eq. (8) by taking the sum of integrals over triangles. See Section 4 for implementation details. By assembling the equations for all vertices \( \mathbf{x} \) into a matrix equation and reordering terms, we get

\[
H_0u_{0,i} = G_0p_0 + D_0(a_{0,n} - g_{0,n}) + Q_0\alpha_n + \sum_{k=1}^{n_{max}} \left( -H_ku_{n-k} + G_kp_{n-k} + D_k(a_{n-k} - g_{n-k}) + Q_k\alpha_{n-k} \right) \left( \begin{array}{c} \cdot \end{array} \right) \left( \text{time history effects} \right)
\]

The matrices \( H_k, G_k, D_k, Q_k \) correspond to the terms with \( \tilde{\mathbf{p}}'^*, \tilde{\mathbf{u}}^*, \mathbf{d}^* \) and \( \mathbf{q}' \) in Eq. (8), respectively. The integral free term \( e^{(t)}(x) \) is absorbed into \( H_0 \). Unlike matrices in FEM, these matrices are dense.

This equation shows that, as a result of discretizing the convolution integrals, finding the unknown current displacement \( u_0 \) requires data both from past steps and the current one. From the current time step, we require the traction and body acceleration vectors; from the past \( n_{max} \) time steps, we require the displacement, traction, and body acceleration vectors. Thus it is a recursive matrix
equation. This structure lets us store information in a surface-only manner, in contrast to volumetric elasticity formulations. Unlike elastostatics BEM [HW16], the $H_0$ matrix is full-rank in general (i.e., no null space) implying that we can straightforwardly solve problems with pure traction boundary conditions.

### 3.2. Local Deformation within a Moving Frame

One of our technical contributions is to use BEM within a moving frame of reference to alleviate the limitations due to linear elasticity, inspired by prior work outlined in Section 2. We approximate the motion of the reference frame with rigid body dynamics based on the linear elasticity assumption of modest deformations. The traction applied on the elastic body’s undeformed shape, based on the linear elastostatics BEM [HW16], the equation. This structure lets us store information in a surface-only manner over consecutive time steps, and we find the following simple geometry-based based approach greatly improves the stability of the method.

We find the translation remaining in the displacement vector by computing the difference between the current centroid and the initial centroid. We find the remaining rotation by applying shape matching [MHTG05] with the mass weighting $w \in \mathbb{R}^N$ for each vertex defined as $w = \int y \cdot \Phi(y) \, d\gamma$, motivated by the surface integral to derive the volume of the body [Mir96].

We subtract the translational error from all elements of $\mathbf{u}$ and solve Eq. (9) for $\mathbf{a}$ in the least squares sense. We then remove the rotational error from $\mathbf{a}$ by $\mathbf{a}_n = R^{-1}(x + \mathbf{u}_n) - x$, where $x$ is a 3N-vector for undeformed vertex positions and $R$ is a rotation matrix for the extracted rotational error. We solve Eq. (9) for $\mathbf{a}$ in the least squares sense with the updated $\mathbf{a}_n$ and $\mathbf{a}_n$. As we only solved Eq. (9) in the least squares sense, the equality is not satisfied exactly in general. Therefore, we solve Eq. (9) for $\mathbf{u}_n$ with the updated $\mathbf{a}_n$ and $\mathbf{u}_n$ as a final step. The removed global translation and rotation are then reintroduced to the system as rigid body frame motions by updating the frame (Eq. 11) using the updated accelerations.

### 3.3. Compression of Matrices

Dense matrices in general require larger storage and computational costs than sparse matrices. In Eq. (9), all $H$ and $G$ matrices are dense, including the time history terms needed for dynamic effects. Each has $3N \times 3N$ entries, so the computational cost is quadratic in the vertex count and thus, unfortunately, scales worse than FEM, which scales linearly in the number of volumetric elements, in practice; we ignore the matrix inversion costs here since we perform inversion in the precomputation step. The BEM community has noticed this issue and proposed compression methods [LSSW12]. The existing methods [KS15; MS10] for elastodynamics simulations do not yet show test cases with large deformations and high compression ratios that we aim for. James et al. [JP03] used lifted wavelet transforms on the input mesh for elastostatics BEM assuming a multiresolution mesh as its input. We present a simple yet effective compression method based on wavelet transforms, which works on a general triangle surface mesh.

The time-domain fundamental solutions $\mathbf{u}'$ and $\mathbf{p}'$ are spatially smooth except at the wavefronts, and they decay smoothly as the source point $\mathbf{y}$ and target point $\mathbf{x}$ move apart. $H$ and $G$ matrices in Eq. (9) generally retain this property; the $(i,j)$ entries of the matrices express how displacement or traction at the $i^{th}$ vertex affect the $j^{th}$ vertex’s displacement. If the $i^{th}$ and $k^{th}$ vertices are spatially close
to each other and have similar normals, the norm of the difference between columns \( j \) and \( k \) will be small, and the same holds for rows. We exploit this property in designing our compression scheme.

First, we reorder the vertices so their indices are more spatially smooth across the surface, using cache-oblivious mesh layouts [YLPKM08]. We also order the matrix elements so that the \( x \) coordinates of all vertices come first, followed by all \( y \) and then all \( z \) coordinates, forming \( 3 \times 3 = 9 \) subblocks. The reordered matrices exhibit several smooth dense blocks (Fig. 3). We then compress the matrices by applying the 2D (for row and column directions) Haar wavelet transform [BCR91] on the matrix elements. Because the transformation is linear, the transformed matrices support efficient evaluations of operations such as general matrix-vector multiplication and multiplication between a row of the matrix and a vector in the compressed form. We also tested other wavelet bases, but among those tested, the Haar basis offered the best balance between the compression ratio and the cost for multiplication operations.

We apply this compression method to the precomputed matrices \( H_1, H_2, H_3 \) and \( H_4 \) for \( k = 1, 2, ..., n_{\text{max}} \), and use them for all BEM solves, including constraint solves (Section 3.4). Note each \( D \) and \( Q \) matrix in Eq. (9) has only \( 3N \times 3 \) entries, making compression unnecessary.

### 3.4. Constraints, Friction, and Domain Decomposition.

To further extend the applicability of our method beyond prior BEM work, we implemented a Gauss-Seidel-based iterative position-level constraint solver. The previous methods solved contact problems by either simply specifying displacements of vertices [JP99] or using a rigid body contact method [HW16]. The computational mechanics literature [GS18] does not handle global motion of objects, to our knowledge. By contrast, we couple both the global rigid frame motion and deformation to solve the constraints and can handle fixed-position constraints, non-penetration constraints, friction constraints, and point-joint constraints.

We linearize the vertex position expressed in terms of the traction vector \( p_0 \) at the current step as follows. The vertex position \( V_k \) at the \( k^{\text{th}} \) time step can be written as

\[
V_k = R_k(x^i + u_k) + T_k, \quad (13)
\]

where \( (\cdot)^i \) denotes a 3-vector for the \( i^{\text{th}} \) vertex. Notice that \( R_k \) (Eq. (11g)) depends nonlinearly on \( p_0^i \), while \( u_k \) (Eq. (12)) and \( T_k \) (Eq. (11i)) depend linearly on \( p_0 \) with the approximations Eq. (11b) and Eq. (11c). We linearize this equation by considering the update of rotation by the rotational velocity from the last time step and the additional rotational velocity of the global rigid frame motion and deformation to solve the constraint equations 

\[
\Delta \omega = \omega_k - \omega_{k-1}, \quad (14)
\]

With this approximation, \( V_k \) depends linearly on \( p_0 \).

To handle positional constraints, we adopt a force-based constraint solve. For each constrained point, we distribute the force to the triangle vertices with barycentric weights, and the force applied at each vertex is converted to traction by dividing the force by the Voronoi area of the vertex. Then, for non-penetration and friction constraints, we formulate complementarity problems with Eq. (14) and solve for \( p \) with the projected Gauss-Seidel method similarly to Duriez et al. [DAK04; DDKA06] with staggered projections [L84; KJ08]. When a fixed-position constraint is imposed, we solve Eq. (14) for \( p \) with the standard Gauss-Seidel method. When a point-joint constraint is imposed, we couple two equations from Eq. (14) by specifying the positions of two points to be the same and the force to be applied in the opposite directions with the same magnitude, and solve them similarly with Gauss-Seidel iterations. Formally, convergence of Gauss-Seidel is only guaranteed if the matrix is symmetric positive definite, and the matrices for our constraint solver are not positive definite because the matrices \( H_0 \) and \( G_0 \) are asymmetric. However, we empirically observed that the solver converges to a solution with constraints satisfied for all the problems we considered.

Using the point-joint constraints, we can implement a domain decomposition approach [BZ11; KJ11; JP02], allowing simulation of larger local displacements in addition to large global displacements. Each subdomain is simulated independently with our method, and point joint constraints are applied between adjacent subdomains to allow simulation of one large object. Since each subdomain must be bounded by a closed surface, this approach requires the addition of triangulated interfaces between the subdomains, which may be undesirable. However, it makes each subdomain’s matrix and cutoff time step size \( n_{\text{max}} \) smaller, making it an interesting alternative.

### 4. Implementation Details

In this section, we describe our choice of quadrature methods and other details for implementation.

**CQM weights and time history size.** The time-domain fundamental solutions, \( a^* \) and \( p^* \), vanish after time \( r_{\text{max}}/c_2 \), where \( r_{\text{max}} \) is the maximum length between any two points on the surface mesh and \( c_2 = \sqrt{\mu/\rho} \) is the shear wave speed. This is because the fundamental solutions vanish after the propagation of all elastic waves. Since the numerical integration weights of the CQM represent discretization of the corresponding time-domain functions, they also vanish after a certain number of steps, leaving only \( n_{\text{max}} + 1 \) terms each in Eq. (8). In our implementation, we set the maximum number of time history steps to be \( n_{\text{max}} = \lceil r_{\text{max}}/(c_2 \Delta t) \rceil + 2 \). Then, we
let \( L = r_{max} \) and \( \mathcal{R} = \epsilon \mathcal{R}_{\text{max}} \) where \( \epsilon = 10^{-10} \), following the work of Schanz et al. [SA97], and define the integration weight as

\[
\phi_k(f) = \frac{\mathcal{R}^k}{L} \int_{\Gamma} \frac{y((\mathcal{R}^k)^2/\Delta)}{e^{ik\nabla} y} \, d\Gamma,
\]

where \( y \) is a characteristic function for the linear multistep method.

We tested two different underlying linear multistep methods for CQM: the backward differentiation formulas of first order (BDF1) and second order (BDF2). We observed BDF2 gives too much oscillation, so we use BDF1 for all examples presented in this paper. The characteristic function for BDF1 is \( y(s) = 1 - s \).

**Spatial quadrature method on the surface.** For each vertex \( \mathbf{x}_i \), in order to evaluate all the surface integrals in Eq. (8), we evaluate the integrals over each triangle on the surface and take a sum of them, i.e., \( \int_{\Gamma} f(x) \, d\Gamma = \sum_{m=1}^{M} \int_{\Gamma_m} f(x) \, d\Gamma \), where \( \Gamma_m \) denotes \( m \)-th triangle. The Laplace domain fundamental solutions \( \hat{u}^*(x, y, s) \) and \( \hat{p}^*(x, y, s) \) and the Laplace domain functions for body force terms \( \hat{q}^*(x, y, s) \) have singularities at \( y = x \). Thus, for each \( \Gamma_m \), if the vertex \( \mathbf{x}_k \) corresponds to one of the vertices of the triangle, the integrand is singular there. We evaluate all non-singular integrals with a 9-point Gaussian quadrature formula that is exact up to fifth degree polynomials [Cow73], after dividing the triangle into four subtriangles. We observed that the use of lower degree formula causes instabilities. We evaluate the singular integrals in different manners depending on the order of singularity. The terms \( \hat{u}^* \), \( \hat{d}^* \) and \( \hat{q}^* \) behave like \( \hat{u}^* \sim 1/r \) as \( r \to 0 \), and we can remove their singularities by performing integration in polar coordinate. The term \( \hat{p}^* \) behaves like \( \hat{p}^* \sim 1/r^2 \) as \( r \to 0 \) and the integral needs to be evaluated in the Cauchy principal value sense; we employ the numerical integration method of Guiggiani et al. [GG90]. We evaluate non-singular integrals of \( \hat{p}^* \) and all the singular integrals with double precision, and use single precision for the rest.

**Evaluation of the integral free term.** The integral free term \( c(\mathbf{x}_i) \) in Eq. (8) is a \( 3 \times 3 \)-matrix associated with the smoothness of the surface. If the surface is smooth at \( \mathbf{x}_i \), \( c(\mathbf{x}_i) = 1/4 \), where \( I \) is an identity matrix. Since \( \mathbf{x}_i \) corresponds to the positions of vertices of input mesh, the surface is not necessarily smooth at \( \mathbf{x}_i \) in general. Therefore, we employ a direct computation method proposed by Mantic [Man93] to evaluate the integral free terms.

**Precomputation of matrices.** We can precompute the matrices in Eq. (9) because they depend only on the undeformed input mesh and the input parameters. The integration weights of Laplace domain functions. Therefore, we can first perform numerical integration of the Laplace domain functions, multiplied by the interpolation matrix if necessary, in space with all possible complex parameters to get Laplace domain matrices and form the final time-domain matrices by taking weighted sums of them. Note we can compute about half of the Laplace domain matrices by taking the conjugate of the other half of matrices. Also, we exploit the symmetry of \( \hat{u}^* \) to reduce the precomputation time. The precomputation of all necessary matrices took between a few minutes and a little under an hour in total for the examples we tested using a GPU implementation; we have not included the exact numbers in this paper simply because our test machine had only 16GB of RAM and disk I/O to store some temporary data was the bottleneck for large meshes.

**Traction discontinuity.** We use a piecewise linear interpolation function over the surface, and therefore special care must be paid to handle traction discontinuities (e.g., at geometric corners where the applied tractions may differ on one side versus the other). We use double nodes [BTW84] to handle this problem: we duplicate the vertices when there are traction discontinuities and construct matrices based on the modified mesh with discontinuous parts. With this method, \( G_0 \) becomes singular but \( H_0 \) does not, which allows us to use the constraint solver without modifications. For the sake of compression, any duplicated vertex is given a vertex index next to that of the original vertex. We apply this technique to compute the matrices for the objects in Fig. 5 and Fig. 9.

**5. Results**

We implemented our method using C++ and performed benchmarks on a desktop computer with an AMD Ryzen 2700X processor and 16GB of RAM using a single thread, except the scene in Fig. 1, for which we exploit object-level parallelism and a simple batch processing for frictional contact solves. Further parallelization such as matrix-vector multiplications for time history terms using a low priority thread is possible. We use a simple spatial hashing-based collision detection method. Other collision detection methods or a penalty springs approach [MZS*11] could be easily combined with our method.

Table 1 shows the computation time, memory consumption for compressed matrix storage, and relevant parameters for the scenes in this paper, and videos for all examples are available in the supplementary video.

**Elastostatics vs. elastodynamics.** Elastodynamic simulation produces more realistic and lively animations compared to elastostatic simulation. Fig. 4 shows how we simulate the secondary motions using an elastodynamic formulation, making our method more visually plausible and suitable for general computer animation. The elastostatic simulation is based on the work by James et al. [JP99] and we replaced its interpolation function with linear interpolation for consistency.

**Figure 4:** Comparison between elastostatics and elastodynamics. A torus is squished from both sides by specifying displacements. Even after the boundary conditions become fixed, we observe the dynamic secondary motions with our method. For consistency, the body frame update is disabled for this scene because a similar method is not available for elastostatics.
**Table 1**: Performance and parameters. \( \mu \) is the shear modulus (=Lame’s second parameter), and \( \nu \) is Poisson’s ratio. CR is the compression ratio applied to matrices computed with (#entries before compression)/#entries kept after compression), and MR is the total memory size required to store the original or compressed matrices. When the compression ratio is not listed, the original dense matrices without compression are used. The times listed are the average time in seconds for one simulation step. UNCNST is the time to compute unconstrained compression are used. The times listed are the average time in seconds for one simulation step. UNCNST is the time to compute unconstrained motion, COLL Desire is the time taken for collision detection, CONST is the time required for the constraint solve, and ERRCR is the time taken for correcting errors and for processing data for the next iterations.

<table>
<thead>
<tr>
<th>scene</th>
<th>Fig.</th>
<th>N</th>
<th>( \Delta t(s) )</th>
<th>( n_{\text{max}} )</th>
<th>( \rho(\text{kg/m}^3) )</th>
<th>( \mu(\text{Pa}) )</th>
<th>CR</th>
<th>MR(GB)</th>
<th>UNCNST(s)</th>
<th>COLL(s)</th>
<th>CONST(s)</th>
<th>ERRCR(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
<td>1</td>
<td>2843 \times 25</td>
<td>0.001</td>
<td>3</td>
<td>10^3</td>
<td>10^0</td>
<td>0.4</td>
<td>5</td>
<td>1.4</td>
<td>4.78 \times 10^{-4}</td>
<td>2.42 \times 10^{-4}</td>
<td>1.82 \times 10^{-4}</td>
</tr>
<tr>
<td>torus</td>
<td>4</td>
<td>1152</td>
<td>0.001</td>
<td>32</td>
<td>5 \times 10^{-5}</td>
<td>5 \times 10^{-3}</td>
<td>0.48</td>
<td>–</td>
<td>6.3</td>
<td>2.37 \times 10^{-1}</td>
<td>–</td>
<td>2.03 \times 10^{-2}</td>
</tr>
<tr>
<td>beam (with frame update)</td>
<td>5</td>
<td>2818</td>
<td>0.005</td>
<td>8</td>
<td>10^{-3}</td>
<td>3 \times 10^{-3}</td>
<td>0.5</td>
<td>2</td>
<td>9.0</td>
<td>5.28 \times 10^{-1}</td>
<td>–</td>
<td>3.05 \times 10^{-2}</td>
</tr>
<tr>
<td>beam (w/o frame update)</td>
<td>5</td>
<td>2818</td>
<td>0.005</td>
<td>8</td>
<td>10^{-3}</td>
<td>3 \times 10^{-3}</td>
<td>0.5</td>
<td>2</td>
<td>9.0</td>
<td>5.30 \times 10^{-1}</td>
<td>–</td>
<td>2.51 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny cube (uncompressed)</td>
<td>6</td>
<td>3000</td>
<td>0.01</td>
<td>5</td>
<td>10^{-3}</td>
<td>5 \times 10^{-3}</td>
<td>0.5</td>
<td>–</td>
<td>7.8</td>
<td>2.68 \times 10^{-4}</td>
<td>–</td>
<td>1.90 \times 10^{-4}</td>
</tr>
<tr>
<td>bunny cube (8x compressed)</td>
<td>6</td>
<td>3000</td>
<td>0.01</td>
<td>5</td>
<td>10^{-3}</td>
<td>5 \times 10^{-3}</td>
<td>0.5</td>
<td>8</td>
<td>1.5</td>
<td>8.55 \times 10^{-2}</td>
<td>–</td>
<td>2.72 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny cube (32x compressed)</td>
<td>6</td>
<td>3000</td>
<td>0.01</td>
<td>5</td>
<td>10^{-3}</td>
<td>5 \times 10^{-3}</td>
<td>0.5</td>
<td>32</td>
<td>0.4</td>
<td>2.20 \times 10^{-2}</td>
<td>–</td>
<td>1.71 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny cube (128x compressed)</td>
<td>6</td>
<td>3000</td>
<td>0.01</td>
<td>5</td>
<td>10^{-3}</td>
<td>5 \times 10^{-3}</td>
<td>0.5</td>
<td>128</td>
<td>0.1</td>
<td>6.75 \times 10^{-3}</td>
<td>–</td>
<td>1.15 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny (friction=0.2)</td>
<td>8</td>
<td>3485</td>
<td>0.003</td>
<td>5</td>
<td>3 \times 10^{-3}</td>
<td>10^0</td>
<td>0.3</td>
<td>10</td>
<td>1.6</td>
<td>9.12 \times 10^{-2}</td>
<td>5.23 \times 10^{-2}</td>
<td>1.25 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny (friction=0.4)</td>
<td>8</td>
<td>3485</td>
<td>0.003</td>
<td>5</td>
<td>3 \times 10^{-3}</td>
<td>10^0</td>
<td>0.3</td>
<td>10</td>
<td>1.6</td>
<td>9.06 \times 10^{-2}</td>
<td>5.55 \times 10^{-2}</td>
<td>6.96 \times 10^{-2}</td>
</tr>
<tr>
<td>bunny (friction=0.6)</td>
<td>8</td>
<td>3485</td>
<td>0.003</td>
<td>5</td>
<td>3 \times 10^{-3}</td>
<td>10^0</td>
<td>0.3</td>
<td>10</td>
<td>1.6</td>
<td>9.04 \times 10^{-2}</td>
<td>7.50 \times 10^{-2}</td>
<td>1.81 \times 10^{-2}</td>
</tr>
<tr>
<td>beam (with decomposition)</td>
<td>9</td>
<td>770.5</td>
<td>0.005</td>
<td>5</td>
<td>10^{-3}</td>
<td>10^0</td>
<td>0.45</td>
<td>–</td>
<td>0.6</td>
<td>1.14 \times 10^{-1}</td>
<td>–</td>
<td>6.64 \times 10^{-2}</td>
</tr>
<tr>
<td>beam (w/o decomposition)</td>
<td>9</td>
<td>2818</td>
<td>0.005</td>
<td>12</td>
<td>10^{-3}</td>
<td>10^0</td>
<td>0.45</td>
<td>2</td>
<td>13.0</td>
<td>7.82 \times 10^{-1}</td>
<td>–</td>
<td>3.05 \times 10^{-2}</td>
</tr>
</tbody>
</table>

**Body frame update.** Our body frame update effectively decouples the global translation and rotation from the local deformation. Fig. 5 shows the effectiveness of our method using the beam bending configuration. Our method alleviates the volume inflation problem significantly in this example.

**Matrix compression.** By applying our matrix compression technique, we can lessen the memory and computational costs. Fig. 6 gives a visual comparison between simulations with different compression ratios. The compression ratio is given by (#entries before compression)/#entries kept after compression).

Our matrix compression method successfully preserves the visual quality while significantly reducing the memory and computational costs (Table 1). For instance, applying the compression ratio of 32 reduces the memory cost by a factor of 21 and the computational cost by 22%. Fig. 7 shows that our method is most effective when both of the reordering methods for matrices are enabled, and is more efficient than naively pruning the smallest entries without wavelet transforms except when an unreasonably high compression ratio is applied.

**Figure 5:** Beam under gravity. Starting from an initial configuration (left), we apply gravity to the body. Without the frame update (middle), we observe a maximum of 30.4% of volume inflation due to linear elasticity’s limitations. With our frame update method, the maximum volume inflation is only 1.91%. The faces are colored by the triangle areas’ relative inflation rate. Red represents more inflation.

**Figure 6:** Matrix compression test. We pull the two sides of bumpy cube sideways. The simulation results are shown using four different compression ratios. The compression parameters and reduction in memory and computational costs are listed in Table 1.

**Frictional contact.** Unlike prior work [JP99; HW15; HW16], our surface-only dynamic deformable method can handle more general computer animation tasks. Fig. 8 shows how an object simulated with our method can interact with the floor with different friction coefficients. The friction from the floor causes additional dynamic motion of the object, and it comes to rest at different positions on the floor depending on the friction coefficient.

**Domain decomposition.** With domain decomposition, we can separately extract the global displacement for each subdomain, as illustrated in Fig. 9. Domain decomposition should be applied when one expects large local displacements, so as to circumvent the limitations of the linear model. The tradeoff is that the simulation with domain decomposition is slower due to the constraints solve iterations, and applying domain decomposition deviates from our surface-only philosophy due to the interior interfaces. Thus, all the other results in this paper are without domain decomposition.
The elastic body behaves differently according to the friction coefficients. Transparent bunnies indicate the position at equally spaced times.

**Figure 7:** Matrix reconstruction error with our compression method. The compression method is applied to the $H_1$ matrix of the bumpy cube scene in Fig. 6. We denote the vertex reordering by "v" and the coordinate reordering by "c" in the legends. The naive method prunes the smallest entries in the input matrix without wavelet transform. The compression errors are the smallest when we enable both reordering methods.

**Figure 8:** Frictional contact test. The same initial velocity is applied to the object in its initial position. The elastic body behaves differently according to the friction coefficients. Transparent bunnies indicate the position at equally spaced times.

**Figure 9:** Domain decomposition demonstration. We twist a beam from its undeformed configuration (left) by gradually rotating both ends. The simulation result without domain decomposition (top) fails to simulate the scene due to the limitations of linear elasticity model, and the object with domain decomposition consisting of five cubes (bottom) alleviates the problem. (More elaborate continuity constraints could help to smooth the transitions between subdomains.)
easy to extend the method to objects with other surface representations besides triangle meshes, including those with vaguely defined boundaries. This includes surfaces defined with NURBS, triangle meshes with holes or overlaps, triangle soup, and point clouds. Evaluation of the boundary integrals given the available information and ensuring stability would be the hurdles.

References


Appendix A: Laplace domain fundamental solutions and surface-only body force terms

The Laplace domain fundamental solutions \( \hat{u}^* \) and \( \hat{p}^* \) appear in the work of Coe et al. [CR868]. We present them here for the readers’ convenience. We use a non-bold font with subscripts to denote components of a vector or matrix in this appendix. The Laplace domain displacement fundamental solution for elastodynamics is

\[
\hat{u}^*_i(x,y,s) = \frac{1}{4\pi} \left\{ \left( \frac{3r_j}{r^3} - \delta_{ij} \right) \left( s \frac{r_j + 1}{s^2} - s \frac{r_j + 1}{s^3} e^{-\frac{r_j}{s}} \right) + \frac{r_j}{r^3} \left( e^{-\frac{r_j}{s}} - \frac{1}{c_1^2} e^{-\frac{r_j}{c_1^2 s}} \right) \right\},
\]

and the Laplace domain traction fundamental solution for elastodynamics is:

\[
\hat{p}^*_i(x,y,s) = \frac{1}{4\pi} \left\{ \frac{6r_j^2}{r^3} \left( \delta_{ij} - 3 \frac{r_j}{r^2} \right) \mathbf{r} \cdot \mathbf{n} \right\} + \frac{e^{-\frac{r_j}{r}}}{r^2 c_1^2} \left[ 2r_j + \frac{3 r_j}{r^2} - 3 \delta_{ij} \right] \mathbf{r} \cdot \mathbf{n} - \frac{2 r_j c_1^2}{r^2 c_2^2}
\]

where

\[
\hat{p}^*_i(x,y,s) = -\frac{1}{4\pi} \left\{ \frac{r_j n_j}{r^3 s^2} \left( \frac{L}{c_1} + 1 \right) e^{-\frac{r_j}{s^2}} - \left( \frac{r_j}{c_1} + 1 \right) e^{-\frac{r_j}{s^2 c_1}} \right\} + \frac{\delta_{ij} \mathbf{r} \cdot \mathbf{n}}{r^3 s^2} \left( 1 - \left( \frac{r}{c_2} + 1 \right) e^{-\frac{r_j}{s^2 c_1}} \right)
\]

The newly derived surface-only body force terms are as follows, and we provide the detailed derivations in the supplementary note. The Laplace domain function for fictitious force term due to translational acceleration is

\[
\hat{d}^*_i(x,y,s) = -\frac{1}{4\pi} \left\{ \frac{r_j n_j}{r^3 s^2} \left( \frac{L}{c_1} + 1 \right) e^{-\frac{r_j}{s^2}} - \left( \frac{r_j}{c_1} + 1 \right) e^{-\frac{r_j}{s^2 c_1}} \right\} + \frac{\delta_{ij} \mathbf{r} \cdot \mathbf{n}}{r^3 s^2} \left( 1 - \left( \frac{r}{c_2} + 1 \right) e^{-\frac{r_j}{s^2 c_1}} \right)
\]

and the Laplace domain function for the Euler force term is

\[
\hat{q}^*_i = \hat{d}^*_i(x-x',y,s),
\]