Real-time Large-deformation Substructuring

Jernej Barbič Yili Zhao

University of Southern California

Figure 1: Model reduction with a large number of localized degrees of freedom: *Left: nonlinear reduced simulation of an oak tree (41 branches ($r = 20$), 1394 leaves ($r = 8$), $d = 1435$ domains, $\hat{r} = 11,972$ total DOFs) running at 5 fps. Right: simulation detail.*

Abstract

This paper shows a method to extend 3D nonlinear elasticity model reduction to open-loop multi-level reduced deformable structures. Given a volumetric mesh, we decompose the mesh into several subdomains, build a reduced deformable model for each domain, and connect the domains using inertia coupling. This makes model reduction deformable simulations much more versatile: localized deformations can be supported without prohibitive computational costs, parts can be re-used and precomputation times shortened. Our method does not use constraints, and can handle large domain rigid body motion in addition to large deformations, due to our derivation of the gradient and Hessian of the rotation matrix in polar decomposition. We show real-time examples with multi-level domain hierarchies and hundreds of reduced degrees of freedom.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physicallybased modeling

Keywords: model reduction, domain decomposition, FEM, nonlinear elasticity

Links: DL PDF WEB VIDEO CODE

1 Introduction

Fast simulation of deformable models is an important problem in computer graphics, with applications in film industry, CAD/CAM, surgery simulation and video games. Model reduction is a popular method for deformable model simulation, mainly because it can approximate complex physical systems at a low computational cost. The key idea of model reduction is to project the high-dimensional equations of motions to a suitably chosen low-dimensional space where the dynamics have properties similar to the original system, but can be timestepped much more quickly [Krysl et al. 2001]. Real-time projection-based model reduction for deformable objects has, however, suffered from an important limitation: the reduction basis is global in space and time. Such bases require a large number of modal vectors to capture local deformations. More importantly, because nonlinear modal elasticity requires implicit integration for stability, and because all global basis vectors overlap in space, each timestep requires (at least) solving a $\hat{r} \times \hat{r}$ dense linear system costing $O(\hat{r}^3)$, where $\hat{r}$ is the number of basis vectors. In practice, this has limited real-time nonlinear reduced simulations to less than (approximately) one hundred degrees of freedom [An et al. 2008].

In this paper, we present an approach to make model reduction adaptive in space, by decomposing the deformable object into several components (the *domains*, see Figure 2). We pre-process the reduced dynamics of each domain separately, and then couple the domains using inertia forces. Assuming a decomposition free of loops, the resulting system supports large deformation dynamics both globally and locally within each domain (e.g., oak leaves in Figure 1). For the geometrically nonlinear FEM material model, the resulting nonlinear system can be timestepped at rates independent of the underlying geometric or material complexity. With exact reduced internal force evaluations on d domains with r degrees of freedom each, the running time of one timestep of our method is $O(dr^4) \ll O(\hat{r}^4)$, for $\hat{r} = dr$, and could be further decreased to $O(dr^3)$ using approximate reduced forces [An et al. 2008].

The idea of decomposing a deformable object for efficient simulation has been previously extensively explored in the engineering community, usually under the names of *domain decomposition* and *substructuring*. However, previous methods either did not pursue reduction in each domain, or limited the domains to small deformations. Our method is related to the well-known Featherstone's algorithm for linked rigid body systems, but differs from it by simulating large deformations involving large interface rotations, combined with model reduction. While the Featherstone's algorithm supports kinematic chains of arbitrary length, we assume shallow hierarchies (five or less in most of our examples), which is sufficient in several computer graphics applications. We approximate subtree inertia using mass lumping, which gives us fast and stable real-time large deformations rich in local detail. Our method supports *in-*

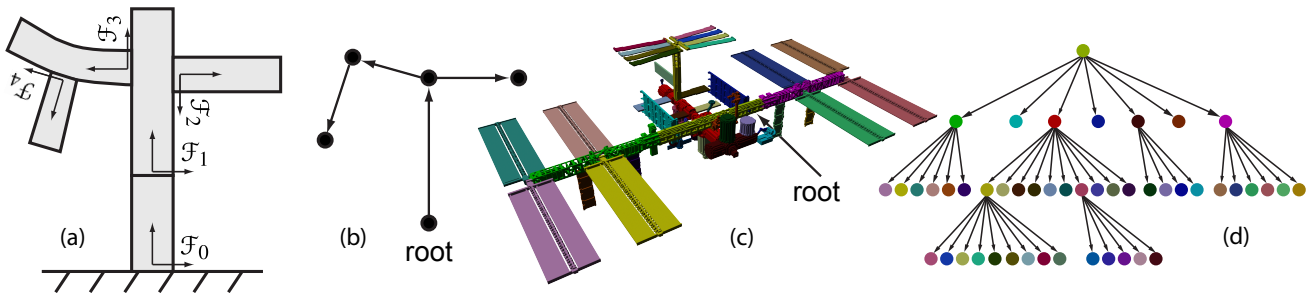

Figure 2: Domain decomposition: *(a) domains, (b) domain graph, (c) 48 domains (space station), (d) domain graph (space station).*

stancing: a single object can be pre-processed once and replicated many times (e.g., the branches and leaves in Figure 1), saving on precomputation and runtime costs. Rigid domains ($r = 0$) are supported, and can be arbitrarily mixed with reduced-deformable domains. The method also supports unanchored objects undergoing free-flight motion.

2 Related Work

The idea to decompose a deformable object into several interconnected components (domains), each of which can be simulated separately, is well-known in science. It is usually referred to as *domain decomposition* [Toselli and Widlund 2004], or, especially in case of repetitive geometry, also *substructuring* [Dodds and Lopez 1980]. Many such methods do not employ reduction, but merely divide the object so that each domain can be assigned to a different processor node, or data for repetitive substructures can be reused [Ryu and Arora 1985]. Perhaps the simplest form to add reduction to domain decomposition is to compute the static equilibrium of each domain, under x, y, z perturbations of each interface vertex, and then restrict the domain deformations to a linear combination of those shapes (*static condensation* [Storaasli and Bergan 1987]). With complex geometry, however, interfaces themselves can be high-dimensional, leading to a large number of basis vectors and slow simulation times. Alternatively, one can compute the linear vibration modes of each structure, under the boundary condition that the interfaces are held fixed [Craig and Bampton 1968]. This method, called *component mode synthesis*, has been popular with simulations of deformations of superstructures, most notably in aerospace engineering (e.g., airplanes, space satellites) [Patnaik et al. 1994]. These previous methods, however, only simulated small, linearized deformations of each domain. It is not straightforward to extend them to large deformations because the resulting large interface rotations seemingly require modes to rotate, which invalidates precomputation. In our paper, we show how these obstacles can be avoided, yielding a component mode synthesis method supporting (1) large deformations within each domain, and (2) large (finite) rotations of the domain interfaces. This makes substructuring much more useful in computer graphics applications requiring large deformations. Domain decomposition for deformable models has also been previously applied in computer graphics, but only for small domain deformations and with running times dependent on the number of domain and interface vertices. For example, a linear quasi-static application using Green's functions has been presented in [James and Pai 2002b], whereas Huang and colleagues [2006] exploited redundancy in stiffness matrix inverses to combine linear FEM with domain decomposition.

Deformable object simulation is a well-studied problem in computer graphics. We review approaches for interactive FEM and model reduction; please see [Nealen et al. 2006] for a general survey. FEM simulations with complex geometry do not run at interactive rates. Interactivity can be achieved using multi-resolution geometric constructions [Capell et al. 2002; Debunne et al. 2001; Grinspun et al. 2002], employing co-rotational elasticity [Müller and Gross 2004; Chao et al. 2010], the multigrid method [Georgii and Westermann 2005], or by coarsening of meshes and their material properties [Kharevych et al. 2009; Nesme et al. 2009]. In our work, we employ model reduction, and demonstrate that whenever the object can be decomposed into natural components, this can provide deformation-rich real-time simulations. In computer graphics, model reduction of nonlinear systems has been used for fast simulation of deformable solids [Metaxas and Terzopoulos 1992; Barbič and James 2005; Kaufman et al. 2008; An et al. 2008; Kim and James 2009] and fluids [Treuille et al. 2006], and for fast control of such systems [Barbič et al. 2009]. One drawback of these systems has been that the reduction basis is global in space. Wicke and colleagues [Wicke et al. 2009] extended Treuille's fluid reduction method to several inter-connected reduced domains. Their approach is similar in spirit to ours, but works for fluids and does not directly apply to nonlinear elasticity. Our open-loop domain structures are related to recursive algorithms for articulated rigid body structures [Featherstone 1987]. Featherstone's algorithm has been extended to small-deformation simulations [Changizi and Shabana 1988; Sharf and D'Eleuterio 1988], and deformable rods [Bertails 2009]. Our work, however, applies to 3D solid deformable objects with irregular interfaces undergoing large deformations.

3 Kinematics

Our method uses reduction to simulate geometrically nonlinear FEM deformations of a 3D volumetric mesh. Let a volumetric mesh be decomposed into d connected and mutually disjoint sets of elements $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d$ (the *domains*, see Figure 2). The common surface where two domains i and j meet is called the *interface*, I_{ij}. Although in principle an object could be decomposed arbitrarily, domain decomposition is most effective when the domains form a natural decomposition of the object, such as, for example, separating the space station modules, panels and antennas into separate domains (Figure 2). In such cases the interfaces are often small or deform mostly rigidly (see Figure 3, right), which we exploit to define our low-dimensional kinematic model. We first form the *domain graph*, where each domain is one node, and nodes are connected if they share a common interface (see Figure 2). We assume that there are no cycles in the domain graph (graph is a tree) and that the tree depth is shallow, but place no restriction on maximum node degree. We direct the graph by picking one domain as the root node, which then uniquely directs all edges by traversing the domains from the root to the leaves. We set the root domain to the domain rooted to the ground for fixed objects, or a central/largest domain for free-flying objects, which tends to minimize tree depth.

We simulate each domain as a nonlinear reduced-deformable object, with its own specific reduced basis U_i of size $r_i \geq 0$. All our reduced models are precomputed under the boundary condition that the domain is held fixed at the interface to the parent, which is con-

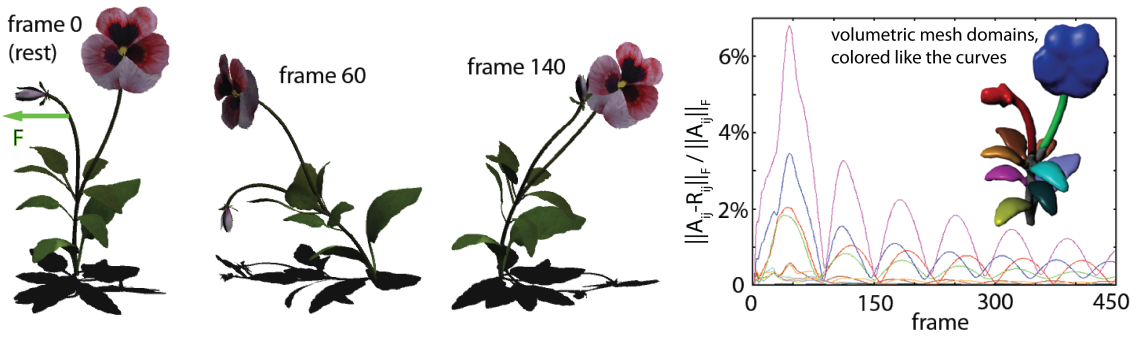

Figure 3: Effect of system and interface forces: *Left: Pulling on the small stem causes large secondary motion (bloom, leaves). Right: relative deviation of the interface transformation A_{ij} from rotation R_{ij} (Frobenius norm, for all the 11 interfaces). The deviations are small.*

sistent with the interface rigidity assumption. The deformation of each domain is given as a linear combination of the modes of domain i, $u_i = U_i q_i$, where u_i contains the 3D deformations of all the vertices of domain i, for the reduced-coordinate vector $q_i \in \mathbb{R}^{r_i}$. Deformations u_i are expressed in a local frame of reference $\mathcal{F}_i$ of each domain, which we define below. Frames are necessary because parts of the mesh may undergo large rotations, whereas modal models are poorly suited to represent large rotations. At first, it may seem possible to avoid frames by including all affine transformations into each basis U_i. However, the remaining vectors in U_i (the non-rigid deformations) would then need to be rotated synchronously with the domain rotation at runtime, leading to a time-dependent basis and a significant additional computational cost.

To define the frames, we first collect all individual vectors q_i into a global vector $q \in \mathbb{R}^{r_1+\ldots+r_d}$. The frame computation then proceeds from the tree root to the leaves. Frame $\mathcal{F}_0$ is the world coordinate frame for fixed objects, and the global rigid body motion frame for free-flying objects. For each child domain j of domain i, we define frame $\mathcal{F}_j$ as *the best fitting frame to the interface* I_{ij}. We do so by specifying its position $x_{ij} \in \mathbb{R}^3$ and rotation $R_{ij} \in \mathbb{R}^{3\times3}$, relative to frame $\mathcal{F}_i$, *and* expressed in the coordinate axes of frame $\mathcal{F}_i$, by fitting the best rigid transformation that transforms vertices of interface I_{ij} from their rest positions to current positions given by q_i (see Figure 4). Let $v^1_{ij},\ldots v^m_{ij}$ be the vertices of domain i that are on the interface to child domain j. We can weight the vertices according to the surface area (or mesh volume) locally belonging to each vertex, arriving at weights $w^1_{ij},\ldots w^m_{ij}$. In domain i, each vertex k deforms according to a $3\times r_i$ submatrix of U_i, denoted by U^k_{ij}. We make x_{ij} track the centroid of I_{ij} :

$$x_{ij} = \frac{1}{\sum_{k=1}^m w^k_{ij}} \sum_{k=1}^m w^k_{ij}(X^k_{ij} + U^k_{ij} q_i) \equiv \hat{x}_{ij} + \hat{a}_{ij} q_i, \qquad (1)$$

where X^k_{ij} is the rest position (in $\mathcal{F}_i$) of vertex v^k_{ij}, and the matrix $\hat{a}_{ij} \in \mathbb{R}^{3\times r_i}$ can be precomputed. In order to fit the rotation, we need to align the interface vertices to their deformed positions as best as possible using a rigid transformation (after subtracting centers). We do so by first computing the covariance matrix [Müller et al. 2005]

$$A_{ij}(q_i) = B_{ij}(q_i) C^{-1}_{ij} \hat{R}_{ij} \equiv \hat{R}_{ij} + \sum_{\ell=1}^{r_i} A^\ell_{ij} q^\ell_i, \quad \text{where} \qquad (2)$$

$$B_{ij}(q_i) = \sum_{k=1}^m w^k_{ij}\big((X^k_{ij} - \hat{x}_{ij}) + (U^k_{ij} - \hat{a}_{ij}) q_i\big)\big(X^k_{ij} - \hat{x}_{ij}\big)^T, \qquad (3)$$

$$C_{ij} = \sum_{k=1}^m w^k_{ij}\big(X^k_{ij} - \hat{x}_{ij}\big)\big(X^k_{ij} - \hat{x}_{ij}\big)^T, \qquad (4)$$

and the fixed matrices $A^\ell_{ij} \in \mathbb{R}^{3\times3}$ can be precomputed. Here, $\hat{R}_{ij}$ is the rotation of interface I_{ij} in the rest configuration, relative to rest frame $\mathcal{F}_i$. We then perform polar decomposition to extract the best-fitting rotation R_{ij}:

$$A_{ij}(q_i) = R_{ij}(q_i) S_{ij}(q_i), \qquad (5)$$

where S_{ij} is a symmetric 3×3 matrix. We note that rigid transformations cannot be made to fit deformable interfaces in general, leading to discrepancies in the domain meshes at the interface. In practice, however, the interface transformation are often very close to rigid and the discrepancies are small (Figures 3, 8). Any interface artifacts in embedded triangle meshes can be removed using MLS surfaces previously proposed for discontinuous FEM [Kaufmann et al. 2009], and we address this rendering issue in Section 5. If interface vertices lie in the same plane in the rest configuration, matrix C_{ij} becomes degenerate. Let $0 \le \lambda_1 \le \lambda_2 \le \lambda_3$ be the eigenvalues of C_{ij}. If $\lambda_1 < \varepsilon\lambda_3$ (we use $\varepsilon = 0.001$), we modify the precomputation of A^ℓ_{ij} by adding an extra point at $\hat{x}_{ij} + \eta N$, where N is the eigenvector for λ_1 (interface normal). The value $\eta = \sqrt{\varepsilon\lambda_3 - \lambda_1}$ increases the smallest eigenvalue of C_{ij} to $\varepsilon\lambda_3$, and gave stable planar interfaces in our examples.

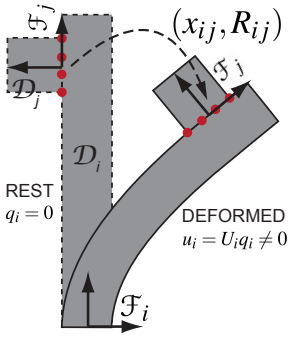

Figure 4: Fitting the best interface transformation.

Once the root frame $\mathcal{F}_0$ and transformations (x_{ij}, R_{ij}) are known for all interfaces I_{ij}, we can easily compute world-coordinate expressions for all frames $\mathcal{F}_i$, $i = 0,\ldots,d-1$. We note that the frames $\mathcal{F}_i$ are completely determined by q and $\mathcal{F}_0$; i.e., the frames are not separate independent simulation parameters. An alternative would be to keep them separate and simulate the combined frame-deformable system, but doing so would require constraints to keep the domains connected, turning the system into a differential-algebraic equation and leading to standard problems of constraint drift. Our construction, in turn, avoids constraints. In the next section, we shall derive the dynamics that govern q, using a mass-lumped formulation running in time linear in the number of domains.

For dynamics, it is necessary to compute linear velocity v_{ij}, linear acceleration a_{ij}, angular velocity ω_{ij}, and angular acceleration α_{ij} of each frame $\mathcal{F}_j$, relative to $\mathcal{F}_i$ and expressed in $\mathcal{F}_i$, based on $q_i, \dot{q}_i, \ddot{q}_i$. Linear quantities are simply $v_{ij} = \hat{a}_{ij}\dot{q}_i$ and $a_{ij} = \hat{a}_{ij}\ddot{q}_i$. To compute the angular quantities, first differentiate Equation 2:

$$\dot{A}_{ij} = \sum_{\ell=1}^{r_i} A^\ell_{ij} \dot{q}^\ell_i, \qquad \ddot{A}_{ij} = \sum_{\ell=1}^{r_i} A^\ell_{ij} \ddot{q}^\ell_i. \qquad (6)$$

Next, we compute the first and second derivatives of the polar decomposition rotation matrix R_{ij} with respect to q_i. We derived a

general formula for $\dot{R}(t)$ and $\ddot{R}(t)$, where $R(t)$ is the rotation in polar decomposition of a 3×3 matrix $A(t)=R(t)S(t)$ that depends on a scalar parameter $t\in\mathbb{R}$ (usually time, but can be any parameter):

$$G=\big(\mathrm{tr}(S)I-S\big)R^T\in\mathbb{R}^{3\times3},\ \omega=G^{-1}\Big(2\,\mathrm{skew}(R^T\dot{A})\Big)\in\mathbb{R}^3, \quad (7)$$

$$\dot{R}=\tilde{\omega}R,\qquad \dot{S}=R^T(\dot{A}-\dot{R}S),\qquad \ddot{R}=\tilde{\dot{\omega}}R+\tilde{\omega}^2R, \quad (8)$$

$$\dot{\omega}=G^{-1}\Big(2\,\mathrm{skew}\big(R^T(\ddot{A}-\tilde{\omega}\dot{A})\big)-\big(\mathrm{tr}(\dot{S})I-\dot{S}\big)R^T\omega\Big). \quad (9)$$

Here, $\tilde{\omega}$ denotes the 3×3 skew-symmetric matrix corresponding to a vector $\omega\in\mathbb{R}^3$, i.e., $\tilde{\omega}x=\omega\times x$ for all $x\in\mathbb{R}^3$. Similarly, $\mathrm{skew}(A)$ denotes the unique skew-vector $\omega\in\mathbb{R}^3$ so that $\tilde{\omega}=(A-A^T)/2$. The derivation of Equations 7-9 is given in Appendix A. Evaluation of $\omega,\dot{R},\dot{S},\dot{\omega},\ddot{R}$ requires solving the 3×3 nonsymmetric linear system given by matrix G. Because this system is nonsingular whenever A is nonsingular, which is the case for interfaces that deform mostly rigidly, computing G^{-1} is stable. We can now apply Equations 7-9 to $A_{ij}(t)=A_{ij}(q_i(t))$ and its time derivatives as given by Equations 2 and 6, yielding ω_{ij} and α_{ij}. We note that such *matrix decomposition gradients* have been previously explored for singular value decomposition [Twigg and Kačić-Alesić 2010; Mathai 1997]. With SVD, singularities in the decomposition gradient occur whenever two singular values are equal, e.g., even if A is identity, and would, unlike polar decomposition, require additional treatment.

4 Dynamics

We now give the equations of motion for q, under the kinematic model of Section 3. These equations simulate the coupled motion of all domains. Each domain follows the equation

$$M_i\ddot{q}_i+D_i\dot{q}_i+f_i^{\mathrm{int}}(q_i)=f_i^{\mathrm{ext}}+f_i^{\mathrm{sys}}+\sum_{j\text{ is child of }i}f_{ij}^{\mathrm{itf}}, \quad (10)$$

where M_i is the reduced mass matrix (constant matrix), $D_i=D_i(q_i)$ is the reduced damping matrix, and $f_i^{\mathrm{int}}(q_i)$ are the reduced nonlinear internal elastic forces of domain i. The terms $f_i^{\mathrm{sys}},f_{ij}^{\mathrm{itf}},f_i^{\mathrm{ext}}$ contain the reduced system forces due to motion of ancestor domains, interface forces of child domains of i arising due to subtree inertia, and external forces, respectively. Equation 10 is standard in model reduction; it is obtained by projecting a full (geometrically nonlinear) FEM deformable model to a chosen basis U_i. We use the modal derivative basis [Barbič and James 2005] because the computation is automatic and does not require any presimulation.

System forces: Equation 10 is expressed in frame $\mathcal{F}_i$ which is non-inertial (accelerates through time). An observer rigidly attached to $\mathcal{F}_i$ can correctly simulate deformations if she adds the resulting system forces f_i^{sys} to the equations of motion of her domain (see, e.g. [James and Pai 2002a]). Let X be a material point in frame $\mathcal{F}_i$. The world-coordinate velocity and acceleration of X, expressed in the coordinate frame $\mathcal{F}_i$, equal [Shabana 2005]

$$v(X)=v_i+\omega_i\times X+\dot{X} \quad (11)$$

$$a(X)=a_i+\alpha_i\times X+\omega_i\times(\omega_i\times X)+2\omega_i\times\dot{X}+\ddot{X}, \quad (12)$$

where $v_i,\omega_i,a_i,\alpha_i$ are the world-coordinate velocity, angular velocity, acceleration and angular acceleration of frame $\mathcal{F}_i$, respectively, expressed in the frame $\mathcal{F}_i$. The system forces are distributed volumetrically throughout the domain. When projected to the low-dimensional space of each domain, they are

$$f_i^{\mathrm{sys}}=-\int_{\mathcal{D}_i}\rho_i(X)U_i^T(X)a(X)dV, \quad (13)$$

where $\rho_i(X)$ is mass density at X, $U_i\in\mathbb{R}^{3\times r_i}$ are the spatially-varying modes, and $a(X)$ is acceleration at X. The ℓ-th component of $f_i^{\mathrm{sys}}\in\mathbb{R}^{r_i}$, for $\ell=1,\ldots,r_i$, can be expanded to

$$f_{i\ell}^{\mathrm{sys}}=W_{i\ell}^{1T}a_i-W_{i\ell}^{2T}\alpha_i+(W_{i\ell}^3+q_i^TW_{i\ell}^4)||\omega_i||^2-$$
$$-(\omega_i^TW_{i\ell}^5+2\dot{q}_i^TW_{i\ell}^6)\omega_i-q_i^TW_{i\ell}^6\alpha_i-(\omega_i\omega_i^T):\sum_{p=1}^{r_i}W_{i\ell}^{7p}q_i^p, \quad (14)$$

for constant precomputable coefficients $W_{i\ell}^j$ (Appendix B). Notation $A:B$ denotes component-wise matrix dot product. The evaluation of f_i^{sys} requires $O(r_i^2)$ flops, and is fast in practice.

Interface Forces: We model reduced interface forces as

$$f_{ij}^{\mathrm{itf}}=-M_{ij}\ddot{q}_i+f_{ij}^0,\qquad \text{for} \quad (15)$$

$$M_{ij}=\overline{m}_j\hat{a}_{ij}^T\hat{a}_{ij},\qquad f_{ij}^0=\hat{a}_{ij}^T\Big(R_{ij}\overline{f}_j^{\mathrm{ext}}- \quad (16)$$
$$-\overline{m}_j\big(a_i+\alpha_i\times x_{ij}+\omega_i\times(\omega_i\times x_{ij})+2\omega_i\times v_{ij}\big)\Big),$$

where $\overline{m}_j$ and $\overline{f}_j^{\mathrm{ext}}$ are the total mass and net sum of external forces (expressed in $\mathcal{F}_j$) in the subtree rooted at j (call it $\overline{\mathcal{D}}_j$). Note that $\overline{m}_j$ is constant and precomputable, but could vary at runtime, e.g., if domains fracture, or are replaced (modularity). Equation 15 is similar to Featherstone's recursive terms. The terms M_{ij} and f_{ij}^0 model the mass inertia of $\overline{\mathcal{D}}_j$, by assuming that the mass is lumped at I_{ij}. The term f_{ij}^0 also models the effect (to domain i) of external forces applied in $\overline{\mathcal{D}}_j$. This approximation gives stable motion and keeps the system matrix symmetric, and we found it reasonable for examples with limited domain graph depth (discussed further in Section 5).

Integration: Equation 15 transforms Equation 10 into

$$\Big(M_i+\sum_jM_{ij}\Big)\ddot{q}_i+D_i\dot{q}_i+f_i^{\mathrm{int}}(q_i)=f_i^{\mathrm{ext}}+f_i^{\mathrm{sys}}+\sum_jf_{ij}^0. \quad (17)$$

We timestep Equation 17 in time linear in the number of domains. The algorithm first constructs the frames for all domains, then timesteps each domain using semi-implicit Newmark integration for stability [Barbič and James 2005], and finally updates relative frame kinematics (see Algorithm 1). The frame of the root domain can be the world-coordinate frame if the root domain is fixed to the ground, or its frame can be floating and affected by the forces and torques of the subdomains for free-flying objects.

Algorithm 1: Multidomain reduced dynamics

```
Procedure Simulation timestep
  Input: values of q̂^(k) = (q, q̇, q̈) and frames F^(k) at timestep k,
         external forces f^ext,(k+1) at timestep k+1, timestep size Δt
  Output: values of q̂^(k+1) and F^(k+1) at timestep k+1
begin
    for all domains i, leaves to root do Assemble f̄_i^ext, m̄_i
    for all domains i, root to leaves do
        Compute ω_i, a_i, α_i, F_i^(k+1)   (Eq. 11, 12)
        Compute f_i^sys   (Eq. 14)
        for all child domains j of i do
            Compute M_ij, f_ij^0   (Eq. 16)
        Do a reduced dynamics timestep (Eq. 17), yielding q̂_i^(k+1)
        for all child domains j of i do
            Use q̂_i^(k+1) to compute x_ij, R_ij, v_ij, ω_ij, a_ij, α_ij
end
```

Example	vol-vtx	vol-el	rend-vtx	rend-tri	d	$\hat{r}$	D	pre	rt:core	rt:StVK	rt:total	S	$u_i = U_i q_i$	fps
flower (tet)	2,713	7,602	6,675	12,868	12	240	3	0.6 min	22 %	78 %	6.5 msec	1	0.43 msec	69 Hz
SIGGRAPH (tet)	18,945	83,753	10,463	20,934	15	160	8	4.3 min	21 %	79 %	1.7 msec	1	2.5 msec	83 Hz
dragon (tet)	46,736	160,553	38,625	77,250	40	454	5	5.1 min	45 %	55 %	1.6 msec	3	7.0 msec	40 Hz
space station (voxel)	219,058	107,556	177,691	248,521	48	921	4	0.8 min	25 %	75 %	4.5 msec	3	25.5 msec	14 Hz
oak tree (tet)	262,363	626,734	578,801	838,704	1435	11,972	5	1.0 min	52 %	48 %	29 msec	1	12.2 msec	5 Hz

Table 1: Simulation statistics *for #volumetric and rendering mesh vertices, elements and triangles (vol-vtx, vol-el, rend-vtx, rend-tri), #domains (d), total # of reduced DOFs ($\hat{r}$), tree depth (D), precomputation time (pre), #simulation steps per graphics frame (S), constructing deformations for rendering ($u_i = U_i q_i$, once per frame), frame rate (fps), simulation step time (rt:total), and its breakdown in terms of timestepping reduced dynamics of individual domains (rt:StVK) and the rest (rt:core, including polar decomposition, gradient and Hessian, system and interface forces, frames). Machine specs: Intel Core i7-980X, 6-Core, 3.33 GHz, 10 GB memory. Only a single core was used.*

5 Results

In our first example we show a complex oak tree (41 branches, 1394 leaves) deformed in the wind and undergoing large deformations (see Figure 1). This example greatly uses substructuring, as the leaves all use a single mesh (rotated and translated into the proper place). The leaf mesh is pre-processed and reduced only once. The copies use pointers to the single datastructure, leading to short precomputation times (Table 1). Similarly, the 41 branches only belong to 5 distinct classes, translated, rotated and scaled (uniformly) into their place. Scalings can be handled efficiently by observing that for a uniform scaling factor s, the basis matrix does not change, whereas the frequency spectrum scales by $1/s$. We also pursued instancing in the space station example (Figure 6).

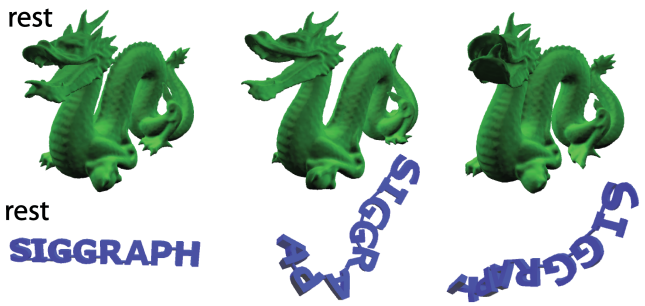

Figure 5: Our method supports localized deformations. *End-effector domains often deform most due to largest system forces.*

Our pre-process is fast (Table 1) and automatic. The number of modes in each domain can be chosen automatically, by setting a total number of modes $\hat{r}$, and assigning to each domain the number of modes proportional to its number of elements. One good choice is a logarithmic distribution: $r_i \propto \log(\#elements(i))$ (dragon example, Figure 5). The SIGGRAPH example demonstrates that our method supports free-flying motion. This is achieved by pre-processing a "free-fly" reduced deformable model [Barbič and James 2005] for the root domain, and then using the external forces to integrate the rigid body motion for the entire object. Our method supports rigid domains ($r_i = 0$), so it can combine rigid and deformable objects: the 8 letters of "SIGGRAPH" are connected by 7 rigid links.

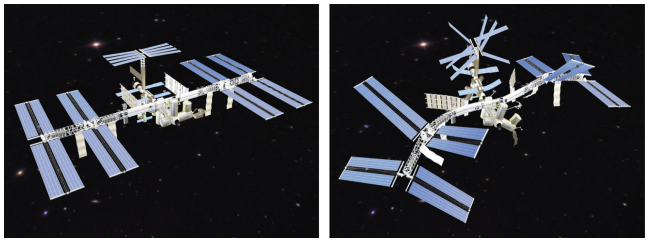

Figure 6: Instancing: *The repeated panel copies are instanced, and can all bend independently, as can all major structure parts.*

Accuracy: Figure 7 compares our method to an unreduced single-domain geometrically nonlinear FEM simulation (ground truth). We provide a comparison to two variants of our method: (L1) interface lumping (Equation 16), and (L2) center-of-mass lumping where the mass of the subtree is lumped not at the interface, but at the center of mass of the subtree. Method L2 can be implemented by adding additional terms to Equation 16, guided by Equation 12. The ground truth (G) was computed offline and is 55x slower than L1 and L2. All three simulations use the same material parameters, timestep, fixed vertices, and the initial condition: a velocity aligned with the first eigenmode of the entire mesh, sufficient to bend the flower stem by about 45 degrees. We found that the three methods give similar trajectories, but differ in oscillation frequencies: L1 and L2 gave 2x and 1.5x higher lowest natural frequency than G, respectively. In general, reduced simulations of solids lack detailed DOFs, resulting in small increases in natural frequency ("artificial stiffening"). With multidomain reduced dynamics, however, the increase is largely due to mass lumping, similar to how a pendulum with mass lumped close to the pivot oscillates at a higher frequency than if the mass was distributed further away. L2 matches G more closely than L1 because lumping at the subtree center better approximates the actual mass distribution.

There is a pyramid of methods that model the subtree inertia progressively better, at the cost of additional implementation complexity. All these methods take the form of Equation 15, but differ in how M_{ij} and f^0_{ij} are computed. All examples in Table 1 use method L1, which we found to be the simplest approach producing stable, deformation-rich results. In L1, each domain feels the total weight of the attached subtree, but not the rotational or deformable inertia. Variant L2 improves the accuracy for physically long domains. In some examples, however, we observed that L2 suffers from instabilities; we attribute this to the quickly time-varying matrices M_{ij} in method L2. At a significant additional implementation complexity, one could compute correct, non-lumped subtree inertia for kinematic chains with arbitrary tree depth (cf. [Shabana 2005]). Such M_{ij} and f^0_{ij} would fully parallel Featherstone's algorithm, but may require modifications to the integrator to maintain stability.

With uniform material parameters, small mesh parts (e.g., protrusions) vibrate at higher frequencies than the rest of the mesh, which manifests as little or no deformations in those regions even with the ground truth (unless poked explicitly). The frequency content of each domain can be linearly scaled at runtime without redoing the precomputation, simply by scaling the domain's precomputed reduced internal forces. In the flower example, we adjusted the frequencies of the leaves so that the leaves gave interesting large deformations when pulling on the stem. Similarly, we made the horns, tail, spike and mouth of the dragon softer than the rest of the mesh, to cause larger deformations in those regions.

Rendering: We render triangle meshes embedded into volumetric meshes. Although the volumetric mesh for the entire object is a manifold mesh in the rest configuration, the domains slightly sep-

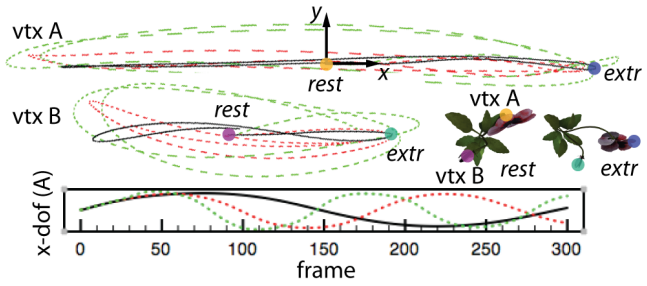

Figure 7: Similar trajectories, but differing frequencies: *Top and middle: Bird's-eye view on the trajectories of two flower vertices: top of primary (A) and secondary bloom (B). Same scale used for A and B. Rest and extreme poses are indicated. Ground truth (G)=solid black, interface lumping (L1)=dashed green, center of mass lumping (L2)=dashed red. Bottom: x-dof of A versus time.*

arate at the interfaces under deformation. At first, we anticipated this to be problematic – however, we found that it can be very easily handled with techniques similar to those employed in discontinuous Galerkin FEM [Kaufmann et al. 2009]. We can establish C^0 continuity simply by averaging the two copies of each interface vertex (see Figure 8), whereas C^1 continuity could be achieved via MLS embeddings [Kaufmann et al. 2009]. Once a consistent volumetric mesh deformation is computed, it is transferred to the embedded triangle mesh using barycentric interpolation. Because our bases are local in space, computing the vertex deformations via equation $u_i = U_i q_i$ only involves matrices U_i with a small number of columns, leading to small memory footprints and fast computation.

Figure 8: C_0 **embedding is effective with nearly-rigid interfaces:** *(a),(c): individual domain meshes, for two representative deformations. Domain gaps (black) are generally small; most are sub-pixel size. (b),(d): Mesh deformations with* C_0 *continuity.*

Decomposing the object into domains: We designed a simple user interface that enabled us to manually decompose the input meshes into domains. The interface permits the user to select vertices or elements, and add or subtract them to a domain. Once a domain is created, the user can replicate it (instancing), or, given an input triangle mesh (e.g., the oak tree mesh purchased online), our software can automatically compute the necessary rest pose domain transformations for the branches and leaves. The domain interfaces, reduction bases and reduced nonlinear internal force coefficients are then computed automatically in a manner of minutes (see Table 1), permitting the user to iterate the decomposition and adjusting the material properties. It would also be possible to decompose the mesh automatically, e.g., guided by the modal properties of the object [Huang et al. 2009], or by any of the many methods proposed for this task in engineering [Farhat 1988].

6 Conclusion

We presented a real-time algorithm for simulations of reduced nonlinear flexible multibody systems undergoing large deformations. The algorithm was made possible by deriving the first and second time derivatives of the rotation matrix used in polar decomposition. The algorithm supports localized deformations, requires no constraints, and runs in time linear in the number of domains.

Limitations and future work: Our work is limited to domain topologies without loops. Several bodies connected and rooted to the ground form a loop, and must be simulated as a single domain in our system. Loops could be closed by adding springs; or more formally, using extensions paralleling those for rigid articulated chains [Featherstone 1987]. We assume that the domain interfaces undergo only a small amount of non-rigid deformation. This assumption is valid when the interfaces are small, and worked well in our examples. Flexible interfaces could be simulated by adding additional "boundary" modes to each domain [Storaasli and Bergan 1987]. Related to that, it may seem plausible to add additional terms to the equations of motion that would explicitly couple the elastic deformations of two domains meeting at an interface. Note that for strictly rigid interfaces such terms are identically zero, and that any external forces already are properly propagated to parent domains via our interface forces. Our algorithm was implemented on a single-core and without GPU computation. Multi-core extensions could map each domain to a separate core, especially when the domain graphs are free of long kinematic chains. In our work, frames follow the motion imposed by the parent exactly, without any freedom for deviation. Some of the six degrees of freedom could be relaxed by introducing joints, which would lead to a method supporting both articulation and large deformations.

Acknowledgements: The authors would like to thank Eitan Grinspun, Doug L. James, James F. O'Brien and anonymous reviewers for their helpful comments and suggestions. This research was sponsored in part by the National Science Foundation (CAREER-53-4509-6600) and the James H. Zumberge Research and Innovation Fund at the University of Southern California.

A Polar Decomposition Rotation Gradient

Let $A = A(t)$ be a 3×3 matrix that depends on a scalar parameter $t \in \mathbb{R}$. For any t, one can perform polar decomposition, $A(t) = R(t)S(t)$, where R is orthogonal and S is symmetric positive semi-definite. Fix $t_0 \in \mathbb{R}$. We first shift the problem by defining

$$B(t) = R^T(t_0)A(t) = \Big(R^T(t_0)R(t)\Big)S(t). \tag{18}$$

Because $R^T(t_0)R(t)$ is identity for $t = t_0$, its derivative at $t = t_0$ must be a skew-symmetric matrix $\tilde{\omega}$ for some $\omega \in \mathbb{R}^3$. By differentiating Equation 18 by t, and setting $t = t_0$, one obtains

$$\dot{B}(t_0) = R^T(t_0)\dot{A}(t_0) = \tilde{\omega}S(t_0) + \dot{S}(t_0). \tag{19}$$

We now replace t_0 with t, and apply the skew operator (Section 3) to both sides of Equation 19. This causes the symmetric term $\dot{S}$ to drop and yields 3 linear equations for the 3 components of ω (Equation 7). In order to derive $\dot{\omega}$ and $\dot{R}$, differentiate both sides of Equation 7 with respect to t (note that G and ω depend on t). After rearranging, one obtains a 3×3 linear system for $\dot{\omega}$ (Equation 9).

B System Force Integrals

The constants of Equation 14 are integrals over each domain $\mathcal{D}_i$:

$$W^1_{i\ell} = -\int_{\mathcal{D}_i} \rho U^\ell_i dV \in \mathbb{R}^3,\ W^2_{i\ell} = \int_{\mathcal{D}_i} \rho \tilde{X} U^\ell_i dV \in \mathbb{R}^3, \tag{20}$$

$$W^3_{i\ell} = \int_{\mathcal{D}_i} \rho U^{\ell T}_i X dV \in \mathbb{R},\ W^4_{i\ell} = \int_{\mathcal{D}_i} \rho U^T_i U^\ell_i dV \in \mathbb{R}^{r_i}, \tag{21}$$

$$W^5_{i\ell} = \int_{\mathcal{D}_i} \rho U^\ell_i X^T dV \in \mathbb{R}^{3\times 3},\ W^6_{i\ell} = \int_{\mathcal{D}_i} \rho U^T_i \widetilde{U^\ell_i} dV \in \mathbb{R}^{r_i \times 3}, \tag{22}$$

$$W^{7p}_{i\ell} = \int_{\mathcal{D}_i} \rho U^\ell_i U^{pT}_i dV \in \mathbb{R}^{3\times 3}, \tag{23}$$

where $U^\ell_i \in \mathbb{R}^3$ is the ℓ-th column of mode $U_i(X) \in \mathbb{R}^{3\times r_i}$.

References

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. on Graphics 27*, 5, 165:1–165:10.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. on Graphics 24*, 3, 982–990.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. on Graphics 28*, 3.

BERTAILS, F. 2009. Linear time super-helices. *Comput. Graphics Forum 28*, 2, 417–426.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. on Graphics 21*, 3, 586–593.

CHANGIZI, K., AND SHABANA, A. A. 1988. A recursive formulation for the dynamic analysis of open loop deformable multibody systems. *Journal of Applied Mechanics 55*, 3, 687–693.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A Simple Geometric Model for Elastic Deformations. *ACM Transactions on Graphics 29*, 3, 38:1–38:6.

CRAIG, R., AND BAMPTON, M. 1968. Coupling of substructures for dynamic analysis. *AIAA Journal 6*, 7, 1313–1319.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling. In *Proc. of ACM SIGGRAPH 2001*, 31–36.

DODDS, R. H., AND LOPEZ, L. A. 1980. Substructuring in Linear and Nonlinear Analysis. *International Journal for Numerical Methods in Engineering 15*, 583–597.

FARHAT, C. 1988. A simple and efficient automatic FEM domain decomposer. *Computers and Structures 28*, 5, 579–602.

FEATHERSTONE, R. 1987. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston.

GEORGII, J., AND WESTERMANN, R. 2005. A multigrid framework for real-time simulation of deformable volumes. In *Proc. of the 2nd Workshop On Virtual Reality Interaction and Physical Simulation*, 50–57.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. CHARMS: A Simple Framework for Adaptive Simulation. In *Proc. of ACM SIGGRAPH 2002*.

HUANG, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. An efficient large deformation method using domain decomposition. *Computers & Graphics 30*, 6, 927 – 935.

HUANG, Q., WICKE, M., ADAMS, B., AND GUIBAS, L. 2009. Shape decomposition using modal analysis. *Computer Graphics Forum 28*, 2, 407–416.

JAMES, D. L., AND PAI, D. K. 2002. DyRT: Dynamic Response Textures for Real Time Deformation Simulation With Graphics Hardware. *ACM Trans. on Graphics 21*, 3, 582–585.

JAMES, D. L., AND PAI, D. K. 2002. Real Time Simulation of Multizone Elastokinematic Models. In *IEEE Int. Conf. on Robotics and Automation*, 927–932.

KAUFMAN, D. M., SUEDA, S., JAMES, D. L., AND PAI, D. K. 2008. Staggered Projections for Frictional Contact in Multibody Systems. *ACM Transactions on Graphics 27*, 5, 164:1–164:11.

KAUFMANN, P., MARTIN, S., BOTSCH, M., AND GROSS, M. 2009. Flexible Simulation of Deformable Models Using Discontinuous Galerkin FEM. *J. of Graphical Models 71*, 4, 153–167.

KHAREVYCH, L., MULLEN, P., OWHADI, H., AND DESBRUN, M. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. on Graphics 28*, 3, 51:1–51:8.

KIM, T., AND JAMES, D. L. 2009. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph. 28*, 5, 1–9.

KRYSL, P., LALL, S., AND MARSDEN, J. E. 2001. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *Int. J. for Numerical Methods in Engineering 51*, 479–504.

MATHAI, A. M. 1997. *Jacobians of Matrix Transformations and Functions of Matrix Argument*. World Scientific Publishing Co.

METAXAS, D., AND TERZOPOULOS, D. 1992. Dynamic deformation of solid primitives with constraints. In *Computer Graphics (Proc. of SIGGRAPH 92)*, 309–312.

MÜLLER, M., AND GROSS, M. 2004. Interactive Virtual Materials. In *Proc. of Graphics Interface 2004*, 239–246.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless Deformations Based on Shape Matching. In *Proc. of ACM SIGGRAPH 2005*, 471–478.

NEALEN, A., MÜLLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2006. Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4, 809–836.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. on Graphics 28*, 3, 52:1–52:9.

PATNAIK, S., GENDY, A., AND HOPKINS, D. 1994. Design optimization of large structural systems with substructuring in a parallel computational environment. *Computing Systems in Engineering 5*, 4-6, 425–440.

RYU, Y. S., AND ARORA, J. S. 1985. Review of Nonlinear FE Methods with Substructures. *Journal of Engineering Mechanics 111*, 11, 1361–1379.

SHABANA, A. A. 2005. *Dynamics of Multibody Systems*. Cambridge Univ. Press, New York, NY.

SHARF, I., AND D'ELEUTERIO, G. 1988. Computer simulation of elastic chains using a recursive formulation. In *IEEE Conf. on Robotics and Automation*, vol. 3, 1539–1545.

STORAASLI, O. O., AND BERGAN, P. 1987. Nonlinear Substructuring Method for Concurrent Processing Computers. *AIAA 25*, 6, 871–876.

TOSELLI, A., AND WIDLUND, O. 2004. *Domain Decomposition Methods*. Springer.

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. *ACM Trans. on Graphics 25*, 3, 826–834.

TWIGG, C., AND KAČIĆ-ALESIĆ, Z. 2010. Point cloud glue: constraining simulations using the procrustes transform. In *Symp. on Computer Animation (SCA)*, 45–54.

WICKE, M., STANTON, M., AND TREUILLE, A. 2009. Modular bases for fluid dynamics. *ACM Trans. on Graphics 28*, 3, 39:1–39:8.