SOFT ROBOTICS
Volume 1, Number 1, 2014

DOI: 10.1089/soro.2013.0010

SoRo
Soft Robotics

ORIGINAL ARTICLE

Dynamic Simulation of Soft Multimaterial 3D-Printed Objects

Jonathan Hiller[1] and Hod Lipson[1,2]

Abstract

This article describes a 2D and 3D simulation engine that *quantitatively* models the statics, dynamics, and nonlinear deformation of heterogeneous soft bodies in a computationally efficient manner. There is a large body of work simulating compliant mechanisms. These normally assume small deformations with homogeneous material properties actuated with external forces. There is also a large body of research on physically based deformable objects for applications in computer graphics with the purpose of generating realistic appearances at the expense of accuracy. Here we present a simulation framework in which an object may be composed of any number of interspersed materials with varying properties (stiffness, density, Poisson's ratio, thermal expansion coefficient, and friction coefficients) to enable true heterogeneous multimaterial simulation. Collisions are handled to prevent self-penetration due to large deformation, which also allows multiple bodies to interact. A volumetric actuation method is implemented to impart motion to the structures, which opens the door to the design of novel structures, and mechanisms. The simulator was implemented efficiently such that objects with thousands of degrees of freedom can be simulated at suitable frame rates for user interaction with a single thread of a typical desktop computer. The code is written in platform agnostic C++ and is fully open source. This research opens the door to the dynamic simulation of freeform 3D multimaterial mechanisms and objects in a manner suitable for design automation.

Introduction

Scientific physics simulators are traditionally used to model small deformations of homogeneous linear-elastic materials.[1] Recently, multimaterial additive manufacturing methods have been developed that fabricate heterogeneous objects out of two or more materials.[2] The properties of these cofabricated materials can range from rigid plastics and metals to very soft rubber with linear deformation greater than 200%. The inclusion of these soft, rubbery materials necessitates the consideration of large, nonlinear geometric deformations to accurately predict physical behavior. Because hard and soft materials can be internally combined and patterned in 3D with very few constraints, a new paradigm of physics simulation becomes necessary to efficiently predict the combined material properties and dynamic behavior.

There are many established methods and implementations for simulating deformable soft bodies.[3] Considering the large deformations and relatively low stiffness of the materials involved, the physically based dynamics are often significant and must be modeled (Fig. 1). Much of the development in simulating soft bodies has been driven by the computer graphics community. Many of the well-established physics engines provide support for dynamic deformable bodies, whether 1D rope, 2D cloth, or 3D ''jello.''[4,5]

The goal of these simulations is generally to create realistic visual effects in real time at the expense of accuracy.[6–9] For instance, lattice shape-matching[10] creates visually appealing rubber effects very efficiently and is unconditionally stable. However, the underlying methods are geometrically based, which limits their direct application to quantitative engineering analysis problems. Other simulators are derived from more physically based principles,[11] but their performance at predicting real-world behaviors is unverified. Deformable body simulators have also been developed specifically for real-time surgery simulation.[12,13] These simulators address challenges such as modifying the geometry dynamically to simulate incisions, but because of the variance of biological materials, it is also difficult to verify quantitative accuracy.

There is also a large body of work regarding the simulation of compliant mechanisms and design thereof.[14,15] Existing efforts focus on small displacements[16] or discrete thin beam members that can flex significantly.[17,18] In the simulation framework presented here, an entire freeform 3D shape can be non-linearly deformed, leading to many novel possibilities

[1]Sibley School of Mechanical and Aerospace Engineering; [2]Computing and Information Science; Cornell University, Ithaca, New York.

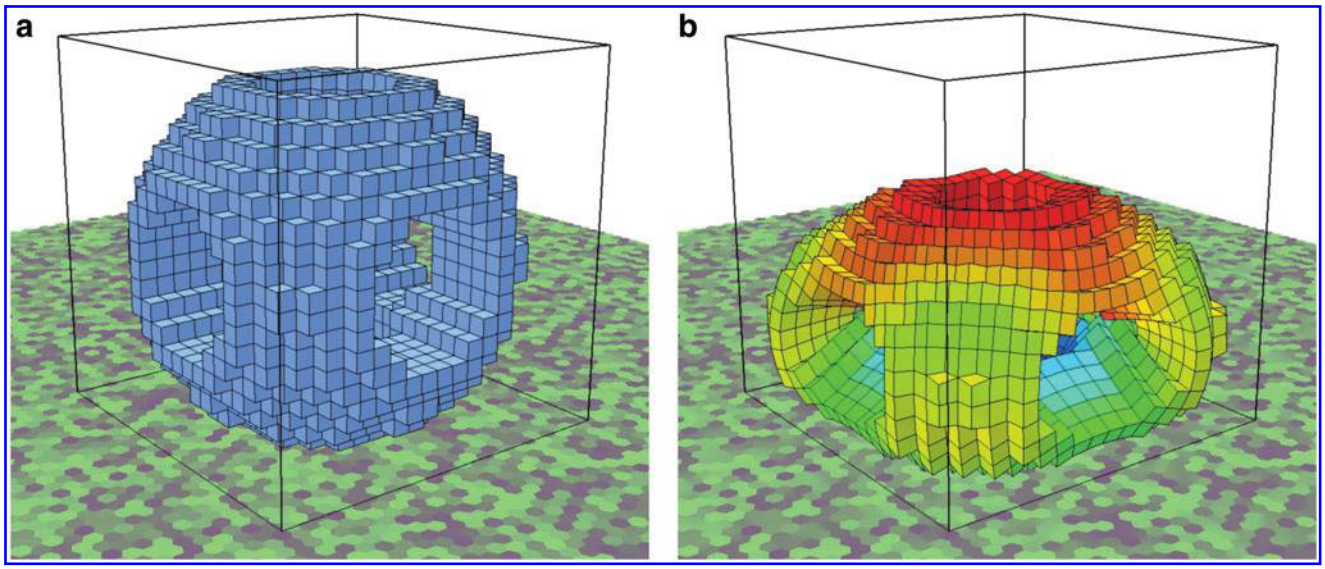

FIG. 1. Two frames of a hollow ball bouncing under gravity illustrate the undeformed ball just before impact **(a)** and the highly deformed ball at the moment of highest deformation **(b)**.

for soft mechanisms that cannot be simulated efficiently using current techniques.

Finite element vs. mass-spring methods

Finite element analysis (FEA) is a well-established method of simulating the mechanical behavior of objects. Advantages include the ability to solve a system with irregularly spaced discretized mesh elements. A stiffness matrix is composed containing information about the connectivity of the entire mesh and the local material properties at each node. However, this system can only be efficiently solved if the underlying equations are linear. Thus, deformations that change the geometry significantly require periodic remeshing.[13] Other nonlinearities such as friction and advanced material models require additional levels of iteration to solve.

Mass-spring methods are widely used for deformable bodies, especially in dynamic simulations for computer graphics.[3] Advantages include relative simplicity and handling large deformations and other nonlinearities with ease. An object is decomposed into discrete point masses connected by springs. Thus, the entire system forms a system of ordinary differential equations (ODEs) that can be integrated directly to solve for the behavior of the system. This makes these particle-based physical simulations very computationally efficient at the expense of accuracy.

Freeform mesh vs. voxels on a lattice

There are a number of tradeoffs associated with choosing either a freeform mesh or a lattice of voxels to dynamically simulate a heterogeneous object. Both FEA methods and many existing gaming physics simulators use a freeform mesh to discretize a 3D object for simulation. By allowing the vertices to lie at any position within the object, there is greater control over the local detail of the simulation. Specifically, this allows objects to be meshed based on the desired accuracy in a given region or dynamically remeshed based on the current regions of interest in a deformed shape.[19] Care must be taken when forming the mesh such that the aspect ratio of each element does not vary significantly in order to preserve accuracy. However, the advantages of freeform meshes quickly diminish as materials of different stiffnesses and properties are interspersed within the object. This constrains the mesh generation process and can potentially create very large and inefficient meshes, such as the case of a dithering between two materials.

Limiting the discretized elements in a simulation to voxels has a number of favorable advantages. This approach enables efficient computation of the force of each constituent element, since they begin on a principal axis with identical lengths. Additionally, the stiffness of each linking beam can be precomputed based on the stiffnesses of each constituent voxel so that each individual voxel can have a unique stiffness without altering the efficiency of the simulator (Fig. 2). This allows heterogeneous materials to be simulated with the same computational complexity as homogeneous materials. Additionally, using a voxel lattice eliminates the possibility of ill-formed meshes.[20] However, voxel lattices are at a disadvantage compared to freeform meshes when large regions of homogeneous material are present or very fine local details must be simulated.

Application in design automation

Because of the exponentially increasing design space enabled by multimaterial additive fabrication methods, design automation will play an increasing role in the design and optimization of structures that fully take advantage of these capabilities. Most design automation algorithms depend on many, many physical evaluations.[21] Therefore, a balance must be struck between calculating accurate results while minimizing central processing unit (CPU) cycles. In the simulation framework presented here, static and dynamic properties are quantitatively very close to the analytical solutions for simple textbook scenarios. Features such as collision detection are in place to avoid the great inaccuracy of self-intersection, but are not meant to draw scientific conclusions about the interaction between two soft bodies. By carefully budgeting CPU cycles, the simulator can accurately

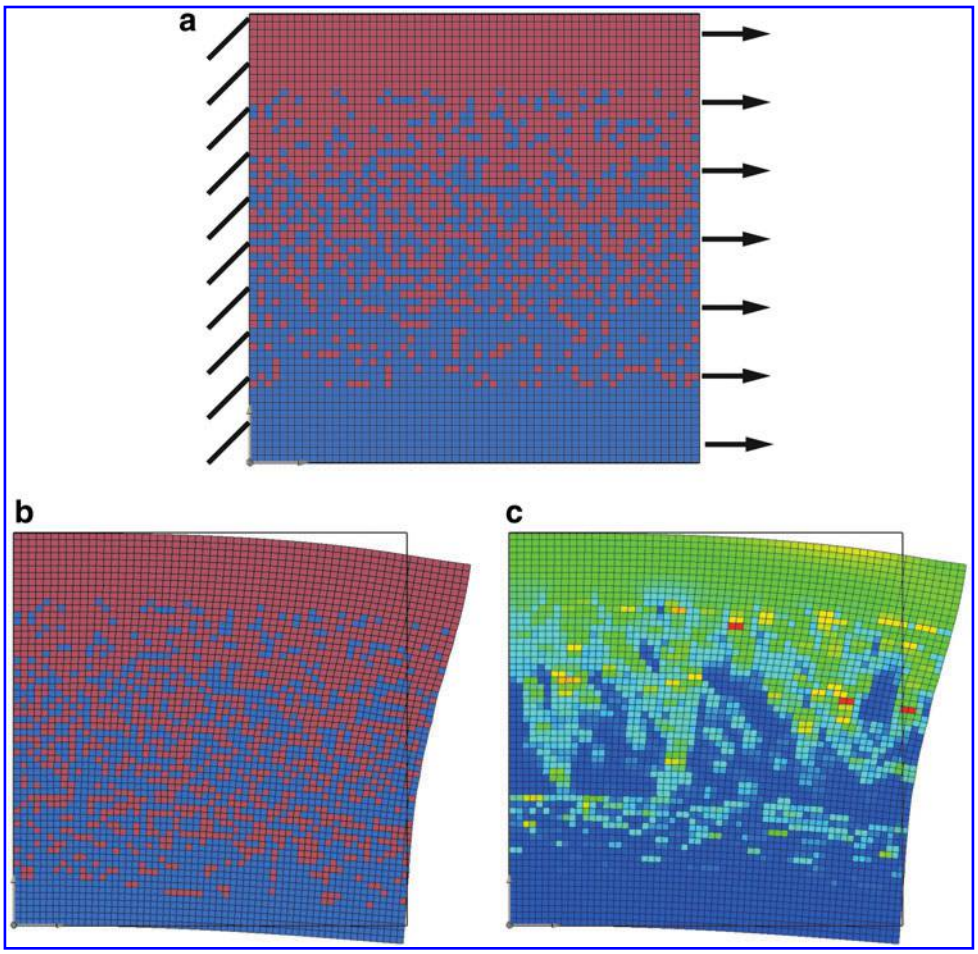

FIG. 2. Advantages of a lattice-based voxel simulation include the native ability to simulate objects with multiple interspersed materials of varying properties **(a)**. Here, the blue material is 100 times stiffer than the red, leading to higher deformation along the top **(b)**. Internal deformation (strain) is also shown **(c)**.

model physical properties while not wasting undue time on negligible effects.

Physics Engine

Heterogeneous deformable body core simulator

A voxel-based, mass-spring lattice was chosen to best simulate the dynamics of highly deformable heterogeneous materials. Several measures were incorporated to mitigate the lack of quantitative accuracy normally associated with the discrete particle-based simulations. Each lattice point was modeled with six degrees of freedom. In addition to the traditional three translational degrees of freedom, all three rotational degrees of freedom are stored and updated as part of the state.

It follows that not only is a mass stored for each voxel, but also its rotational equivalent moment of inertia. Instead of using simple extension springs to connect adjacent points, more complex beam elements were used that resist lateral shearing and rotation in all axes in addition to extension. By setting the properties of the beam equal to the equivalent size and stiffness of the bulk material connecting two voxels, a good approximation of the aggregate bulk material behavior is obtained.

To prepare a given geometry for simulation, the target geometry was first voxelized into cubic voxels. Each subsequent relaxation step consists of two steps: (1) calculating all internal forces, and then (2) updating all positions. In order to preserve the proper dynamics of the system, positions were updated synchronously. Therefore, the order of calculation is irrelevant. To capture information about both translation and rotation of the voxels relative to each other, a constant cross-section beam element was used to connect adjacent voxels in the lattice. Beam elements resist both translation and rotation by exerting biaxial bending, transverse shear, and axial stretching forces in response to appropriate displacements (Fig. 3). Here we use a standard Bernoulli-Euler beam theory.[22] It is important to note that the Bernoulli-Euler beam theory assumes a linearized beam model. This implies that even though the physics engine presented here is capable of modeling large aggregate nonlinear deformations, the accuracy drops off as the angle between any two adjacent voxels becomes too large for a reasonable small-angle approximation.

Compositing adjacent dissimilar materials. The Bernoulli-Euler beam theory also requires the material to be elastic and isotropic. When two adjacent voxels are composed of the same material, the elastic modulus and stiffness of this material are used in the equivalent beam connection. However, when the materials have differing properties, an appropriate composite property must be calculated. To this end, we approximate the composite stiffness of a bond between two dissimilar materials by:

$$E_c = \frac{2E_1E_2}{E_1 + E_2} \tag{1}$$

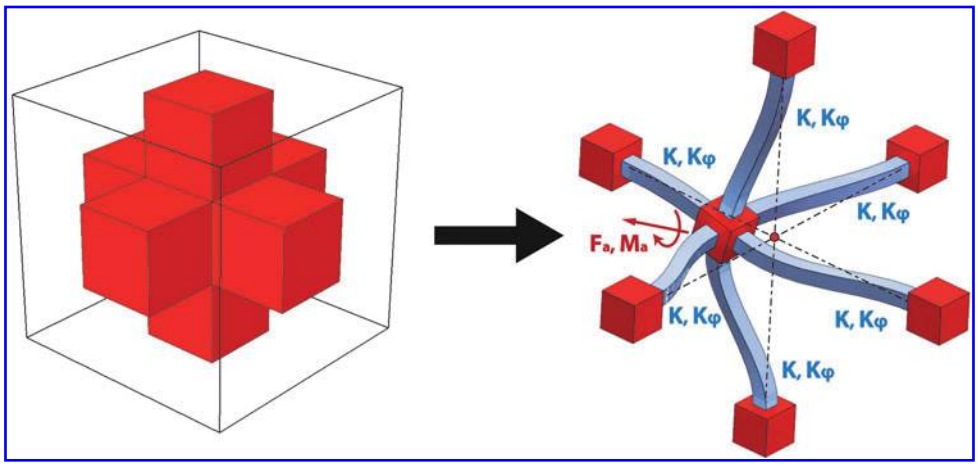

FIG. 3. Each voxel is modeled as a lattice point with mass and rotational inertia (red). Voxels are connected by beam elements (blue) with appropriate translational and rotational stiffness leading to realistic deformation under applied forces and moments.

where E_c is the composite elastic modulus and E_1 and E_2 are the two constituent stiffnesses. Since the elastic modulus directly corresponds to the spring constant of each bond, this is analogous to combining two springs of half the length in series with dissimilar stiffnesses. The composite shear modulus are calculated in a similar manner:

$$G_c = \frac{2G_1G_2}{G_1 + G_2} \tag{2}$$

where G_c is the composite shear modulus and G_1 and G_2 are the two constituent shear moduli. Because Poisson's ratio relates the elastic modulus to shear modulus, it follows that the composite Poisson's ratio μ_c is calculated by:

$$\mu_c = \frac{E_c}{2G_c} - 1 \tag{3}$$

Simulation elements. Since solid objects are represented by a network of beams connecting nodes, the physical parameters for these beams must be calculated. Because the geometry is constrained to voxels, the length l of the beam was taken to be the distance between the voxels, and the cross-sectional area of the beam A was l^2. The standard formula was used to calculate the bending moment of inertia (I), given that in this case both the base b and height h equal the lattice dimension l.

$$I = \frac{bh^3}{12} = \frac{l^4}{12} \tag{4}$$

The torsion constant (J) was approximated by the polar moment of inertia of a rectangular cross-section beam and calculated as

$$J = \frac{bh(bb + hh)}{12} = \frac{l^4}{6} \tag{5}$$

Using the standard Hermitian cubic shape functions for beam elements, the stiffness matrix was determined for a beam element with 12 degrees of freedom: Three translational and three rotational degrees of freedom for each endpoint of the beam. This can be assembled into a stiffness matrix. The result for a beam element oriented in the positive X direction is as follows:

$$\begin{bmatrix} F_{x_1} \\ F_{y_1} \\ F_{z_1} \\ M_{\theta_{x_1}} \\ M_{\theta_{y_1}} \\ M_{\theta_{z_1}} \\ F_{x_2} \\ F_{y_2} \\ F_{z_2} \\ M_{\theta_{x_2}} \\ M_{\theta_{y_2}} \\ M_{\theta_{z_2}} \end{bmatrix} = [K] \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ \theta_{x_1} \\ \theta_{y_1} \\ \theta_{y_1} \\ X_2 \\ Y_2 \\ Z_2 \\ \theta_{x_2} \\ \theta_{y_2} \\ \theta_{z_2} \end{bmatrix} \tag{6}$$

where the stiffness matrix $[K]$ is

$$[K] = \begin{vmatrix} a_1 & 0 & 0 & 0 & 0 & 0 & -a_1 & 0 & 0 & 0 & 0 & 0 \\ & b_1 & 0 & 0 & 0 & b_2 & 0 & -b_1 & 0 & 0 & 0 & b_2 \\ & & b_1 & 0 & -b_2 & 0 & 0 & 0 & -b_1 & 0 & -b_2 & 0 \\ & & & a_2 & 0 & 0 & 0 & 0 & 0 & -a_2 & 0 & 0 \\ & & & & 2b_3 & 0 & 0 & 0 & b_2 & 0 & b_3 & 0 \\ & & & & & 2b_3 & 0 & -b_2 & 0 & 0 & 0 & b_3 \\ & & & & & & a_1 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & b_1 & 0 & 0 & 0 & -b_2 \\ & & & & & & & & b_1 & 0 & b_2 & 0 \\ & & & & & & & & & a_2 & 0 & 0 \\ & (sym) & & & & & & & & & 2b_3 & 0 \\ & & & & & & & & & & & 2b_3 \end{vmatrix} \tag{7}$$

where

$$a_1 = \frac{E_cA}{l} \tag{8}$$

$$a_2 = \frac{G_cJ}{l} \tag{9}$$

$$b_1 = \frac{12E_cI}{l^3} \tag{10}$$

$$b_2 = \frac{6E_cI}{l^2} \tag{11}$$

$$b_3 = \frac{2E_cI}{l} \tag{12}$$

To reduce the computational complexity, the transformations to rotate each individual element from its initial orientation into the positive X direction were precomputed. Because of the cubic voxel lattice constraints, all elements are initially located on principal axes. This makes the transformation into the X-direction stiffness matrix computationally trivial. As the structure deforms over the course of a simulation, an additional transformation was calculated and applied to each element to translate the position and angle of the first voxel to zero. Therefore X_1, Y_1, Z_1, θ_{x1}, θ_{y1}, and θ_{z1} all become zero and drop out of the calculation. The large matrix calculation above is reduced to:

$$F_{x_1} = -a_1X_2 \tag{13}$$

$$F_{y_1} = -b_1Y_2 + b_2\theta_{z_2} \tag{14}$$

$$F_{z_1} = -b_1Z_2 + b_2\theta_{y_2} \tag{15}$$

$$M_{\theta_{x_1}} = -a_2\theta_{x_2} \tag{16}$$

$$M_{\theta_{y_1}} = b_2Z_2 + b_3\theta_{y_2} \tag{17}$$

$$M_{\theta_{z_1}} = -b_2Y_2 + b_3\theta_{z_2} \tag{18}$$

$$F_{x_2} = -F_{x_1} \tag{19}$$

$$F_{y_2} = -F_{y_1} \tag{20}$$

$$F_{z_2} = -F_{z_1} \tag{21}$$

$$M_{\theta_{x_2}} = a_2\theta_{x_2} \tag{22}$$

$$M_{\theta_{y_2}} = b_2Z_2 + 2b_3\theta_{y_2} \tag{23}$$

$$M_{\theta_{z_2}} = -b_2Y_2 + 2b_3\theta_{z_2} \tag{24}$$

The resulting forces and moments are then transformed back to the current orientation of the bond using the inverse of the transform calculated to arrange them in the positive X axis. The forces for all the bonds are calculated separately, then total forces (F_t) and moments (M_t) on each voxel are summed according to how many bonds n are connected to it.

$$F_t = \sum_{b=0}^{b=n} \vec{F}_b \tag{25}$$

$$M_t = \sum_{b=0}^{b=n} \vec{M}_b \tag{26}$$

Integration. Because momentum plays a key role in all dynamic simulations, two integrations are necessary to update the position realistically. For this physics engine, double Euler integration was used. Although there are more accurate integration methods, such as the Runge-Kutta (RK4) method, Euler was chosen because the massive number of discrete voxels and nonlinear effects such as stick-slip friction are not well suited for the predictive steps of the RK4 integration scheme. The state of each voxel was represented by 3D position ($\vec{D}$) and rotation ($\vec{\theta}$) vectors and 3D linear ($\vec{P}$) and angular momentum ($\vec{\phi}$) vectors. In order to advance the simulation from time t_n to time $t_{n+1} = t + dt$,

$$\vec{P}_{t_{n+1}} = \vec{P}_{t_n} + F_t dt \tag{27}$$

$$\vec{D}_{t_{n+1}} = \vec{D}_{t_n} + \frac{\vec{P}_{t_{n+1}}}{m} dt \tag{28}$$

$$\vec{\phi}_{t_{n+1}} = \vec{\phi}_{t_n} + M_t dt \tag{29}$$

$$\vec{\theta}_{t_{n+1}} = \vec{\theta}_{t_n} + \frac{\vec{\phi}_{t_{n+1}}}{I} dt \tag{30}$$

where m is the mass of the voxel, and I is the rotational inertia.

Choosing the timestep. A critical aspect of implementing a robust physics simulation driven by Euler integration is to choose a suitably small timestep to prevent numerical instability. However, in order to be computationally efficient, the timestep should not be unduly small. Fortunately, it is trivial both conceptually and computationally to determine the longest stable timestep at each iteration of the simulation. In an oscillating system, the simulation will be stable if

$$dt < \frac{1}{2\pi\omega_{0_m}} \tag{31}$$

Because each bond between voxels is essentially a mass-spring-damper system, ω_{0m} is simply the maximum natural frequency of any bond in the system. The stiffness of each bond was divided by the minimum mass of either voxel connected to it to calculate the maximum natural frequency of each bond according to

$$\omega_{0_{max}} = \sqrt{\frac{k_b}{m_m}} \tag{32}$$

where k_b is the stiffness of the bond and m_m is the minimum of either mass connected by this bond.

Damping. Once an optimal timestep has been chosen, it is necessary to implement damping into the system to avoid the accumulation of numerical error as well as to enable realistically damped material properties. Because one application goal of this simulation involves unconstrained motion of soft bodies, damping must be included at the local interaction between voxels, not just applying a force proportional to each voxel's global velocity, which would damp rigid body motion.

The local damping between adjacent voxels ensures that modal resonances at the scale of a single voxel do not accumulate. For each bond between two voxels, a force was applied to each voxel opposing the relative velocity between them. However, because rotational degrees of freedom allow this bond to be spinning, both angular and translational

velocities must be correctly accomodated to make sure rigid body motion is not being damped. For each bond, first the average position, velocity, and angular velocity were calculated. Then the velocity of the second voxel relative to the first ($\vec{V}_{2\to1}$) is calculated according to

$$\vec{V}_{2\to1} = (\vec{V}_2 - \vec{V}_a) + (\vec{D}_2 - \vec{D}_a) \times \vec{\omega}_a \tag{33}$$

where $\vec{V}_2$ is the velocity of the second voxel, $\vec{V}_a$ is the average velocity of two voxels, $\vec{D}_2$ is the position of the second voxel, $\vec{D}_a$ is the average position, and $\vec{\omega}_a$ is the average angular velocity. This is in effect subtracting out the rigid motion components of the relative velocity such that they are not damped. Then for each voxel the damping force is calculated according to the standard linear damping formula:

$$F_d = 2\zeta\sqrt{mk}V_r \tag{34}$$

where F_d is the damping force to be applied to the voxels with mass m attached to a bond with stiffness k and a relative velocity V_r. The damping ratio ζ is normally selected to be 1, corresponding to critical damping. Likewise, angular velocities are also damped according to

$$M_d = 2\zeta\sqrt{Ik_\phi}\omega_r \tag{35}$$

where M_d is the damping force to be applied to the voxels with a rotational moment of inertia I attached to a bond with rotational stiffness k and a relative angular velocity ω_r. The rotational damping ratio ζ is also normally selected to be unity. However, even though each bond is critically damped locally, the structure as a whole is still quite underdamped. So, each voxel was also variably damped relative to the ground in a similar manner.

Collisions

Gravity, floor, and friction model. In order to properly simulate freely moving soft bodies, gravity is necessary. As the force is summed on each voxel, the mass of the voxel times the acceleration of gravity was subtracted from the vertical component of force. In conjunction with gravity, a floor was implemented for objects to rest on. Because the maximum simulation timestep that can be taken is limited by the maximum stiffness between any two connected masses, the effective normal stiffness of the floor on any voxels in contact with it cannot be infinitely high. In order to keep the simulation as efficient as possible, the stiffness of each voxel contacting the floor was the stiffness of the floor in that location. Although this allows significant floor penetration in some cases, the qualitative behavior is appropriate. Potential collisions with the floor are trivial to detect by simply comparing the vertical position of each voxel to the ground plane after accounting for the current size of the voxel.

Although a standard linear friction model would provide a relatively realistic simulation, much more interesting and realistic behavior can be observed using a Coulomb friction model. This implies that a voxel at rest with the floor will resist any motion until

$$|F_l| > \mu_s F_n \tag{36}$$

where F_l is the horizontal force parallel to the ground, μ_s is the coefficient of static friction between the voxel and the ground plane, and F_n is the normal force pressing this voxel into the plane of the ground. A Boolean flag is set, indicating to the simulation that this voxel should not move laterally, but can still move in the direction normal to the ground such that it can be unweighted and then moved laterally. Once the static friction threshold has been exceeded at any given timestep, the voxel is allowed to begin motion in the appropriate lateral direction by clearing the Boolean static friction flag. The voxel is allowed to move in all three dimensions, but a friction force is applied, opposing the lateral direction of motion according to:

$$|F_l| = \mu_d F_n \tag{37}$$

where μ_d is the dynamic coefficient of friction. In order to properly detect when a voxel has stopped lateral motion, a minimum motion threshold must be set. Otherwise the voxel will never reenter the static friction state until the velocity is less than the precision of a floating point variable. To detect a stopping voxel, especially one that would change direction and incorrectly bypass the effects of static friction, a voxel is artificially halted if

$$V_l \le \frac{F_n \mu_d dt}{m} \tag{38}$$

where m is the mass of the voxel in question. Because the force of friction ($F_n\mu_d$) is always directly opposed to the voxel's lateral velocity (V_l), the voxel is stopped if the projected change in velocity would change the direction of the surface velocity, which would involve the momentary stopping of the voxel. Collisions are also damped normal to the direction of contact with a user variable damping ratio ranging from zero (no damping) to one (critical damping).

Self-collision detection and handling. Collision detection between voxels must be implemented carefully to avoid this being a bottleneck in CPU cycles. Especially in simulations with many independently moving particles, the $O(n^2)$ process of checking every particle against every other to detect collisions is prohibitively expensive in CPU cycles. In a voxel simulation such as this there are many ways we can make collision detection more efficient. Since large deformations and multiple bodies are possible, we cannot simply exclude collision detecting between voxels that are connected. However, we can suppose that voxels on the interior of an object may be disregarded for any collisions, assuming that collisions are handled in such a way that overlaps cannot penetrate the outer shell. Upon import into the simulation, a list of surface voxels is precomputed, since this information will never change.

The next step is to build a list of voxel pairs that are within a collision horizon (reasonable range) of each other. Voxels on this shortlist should be compared at every timestep for potential overlap. The collision horizon was chosen to be a distance equivalent to two voxels. However, it is undesirable to watch for potential collisions between voxels that are adjacent and connected in the lattice since the internal forces between them already resist penetration. To account for this, a list of voxels within a 3D Manhattan distance of 3 in the

lattice is precompiled upon import into the simulation for each voxel. Once complete, any potential collision interactions can be compared against this list in linear time to exclude the computational overhead of calculating spurious collision interactions.

Finally, the list of potential voxel collision pairs must be updated often enough that out-of-range voxels beyond the collision horizon do not have a chance to penetrate before being recognized as potential collisions. To accomplish this, a global maximum motion variable is initialized in the simulation. Each timestep, the magnitude of the maximum velocity of any voxel in the simulation is added to this variable. While the maximum motion is less than half the collision horizon, it is guaranteed that no voxels can overlap into a collision. This is extremely conservative, but also computationally trivial to compute.

Volumetric actuation

For convenience, we will refer to volumetric actuation in the context of materials with a nonzero coefficient of thermal expansion (CTE) in conjunction with a changing ''temperature'' control variable. However, volumetric actuation may be physically achieved in a variety of ways, so there is no reason to assume that the results presented here are applicable only to temperature changes. There is also no reason that different materials within the simulation couldn't expand or contract out of sync to multiple independent control variables. This would be analogous to having multiple ''temperatures'' that only affect certain materials. But in the following discussion, we will refer to only a single temperature control variable.

With the soft-body relaxation engine in place, such volumetric actuation is implemented by simply changing the nominal rest length between adjacent voxels when computing the elastic force between them. If the elastic force (F_E) between two voxels is normally calculated according to

$$\vec{F}_E = K(\vec{P}_2 - \vec{P}_1 - \vec{D}_{N_{P_1 \to P_2}}) \tag{39}$$

to add in the effects of volumetric actuation,

$$\vec{D}_{N_{P_1 \to P_2}}|_{T=T_c} = \left(1 + \frac{\alpha_1 + \alpha_2}{2}(T_c - T_r)\right)\vec{D}_{NP_1 \to P_2}|_{T=T_r} \tag{40}$$

where $\vec{D}_{N_{P_1 \to P_2}}|_{T=T_c}$ is the modified rest distance based on the current temperature, α_1 and α_2 are the coefficients of thermal expansion of the bond's constituent materials, T_c is the current temperature, and T_r is the reference temperature, at which there is no temperature-based expansion or contraction.

Validation

In order to ensure that the physics engine was performing properly, we compared both static and dynamic behaviors of cantilever beams to finite element and analytical solutions. To verify the static behavior of the simulation, beam deflections of both thin and thick cantilever beams were compared to a linear direct stiffness method and (in the case of the thin beam) to the analytical solution. The results are outlined in Table 1. For the thin beams, $20 \times 1 \times 1$ voxels were used with a physical size of 1 mm each for a total beam size of 20 mm long by 1 mm thick. A material stiffness of 1 MPa was specified. The force at the end of the beam was selected to be 0.03 mN so that the displacement would be small (less than a voxel-height). This ensures that small-angle approximations of the analytical solution are valid. The thick beams were modeled as $10 \times 5 \times 5$ blocks of 1 mm voxels with the same stiffness. In this case, 0.1 N of force was applied at the free end to achieve a non-negligible displacement.

TABLE 1. MAXIMUM DISPLACEMENTS OF THIN AND THICK CANTILEVER BEAMS

Geometry	*Mass/ spring*	*Direct stiffness*	*Analytical*
Thin cantilever beam	0.822 mm	0.823 mm	0.823 mm
Thick cantilever beam	0.538 mm	0.546 mm	N/A

The nonlinear mass-spring method presented here results in slightly smaller displacements than the direct stiffness method and the analytical solutions (Table 1). This difference is negligible in the thin beam case. The difference is more pronounced in the thick beam case. This is likely because the deformation (Fig. 4) is large enough that the change in geometry in the relaxation method factors into the results. Therefore, we suspect that in this case the linear methods slightly overpredict the deflection.

To verify the dynamic properties of the simulation, a thin cantilever beam of the same dimensions and properties as used in the previous section was excited with an impulse force at the free end. Damping was turned off, except a trace

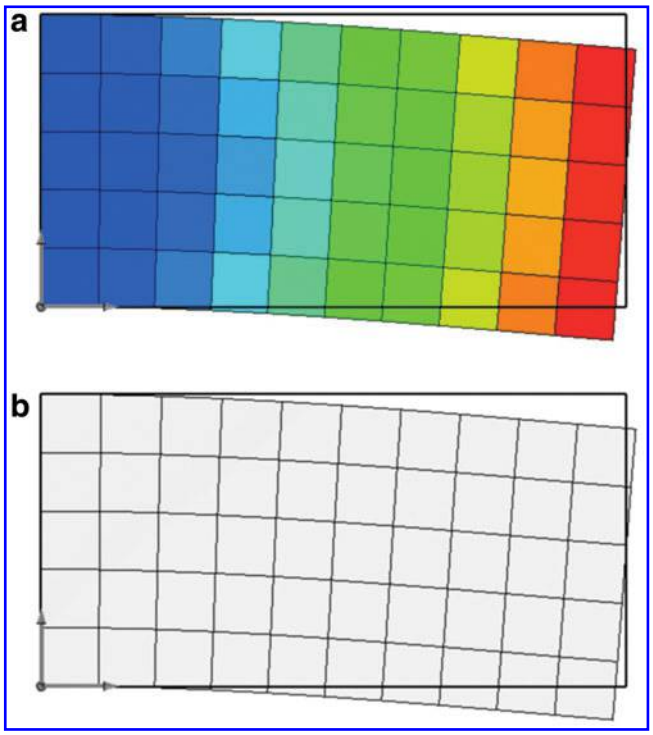

FIG. 4. The deflection of a thick cantilever beam as calculated by the direct-stiffness method (finite element analysis) **(a)** and the mass-spring method **(b)**.

amount of local bond rotational damping ($\zeta = 0.01$) to maintain numerical stability. A total of 20,000 data points were collected of the z position of the voxel at the free end of the beam, corresponding to 0.13 seconds of physical time, or about 60 oscillations of the lowest-frequency fundamental mode. The frequency characteristics of this data were then plotted with the analytically calculated natural frequencies of a thin cantilever beam. (Fig. 5). The analytical natural frequencies were calculated according to

$$\omega_n = K_n \sqrt{\frac{EI}{\bar{m}L^4}} \tag{41}$$

where ω_n is the natural frequency of mode n in radians per second, K_n is the standard scaling factor for this mode, and $\bar{m}$ is the mass per unit length of the beam.[23] These values are overlaid on Figure 5, and the simulated and analytical natural frequencies are tabulated in Table 2. The simulation predicts natural frequencies that are slightly lower than those predicted by beam theory. This is likely because the 20×1 aspect ratio of the simulated beam is not quite an ideal thin beam, and because there is a small amount of damping in the simulation that would tend toward under-predicting natural frequencies.

Results

Simulation performance

Several parameters were explored to characterize the performance of the soft body simulator. All results presented here assume the simulation is run on a single worker thread of a CoreI7 CPU at 2.67 GHz. As implemented, the simulation proved very computationally efficient. For a reasonable size object of 4000 voxels, 122 complete simulation iterations were completed per second, or approximately 500,000 voxel calculations per second. As the number of voxels increases in the object, the total voxels calculated per second decreases, but not dramatically. The simulation speed per voxel for cubic blocks of various numbers of elements are shown in Figure 6.

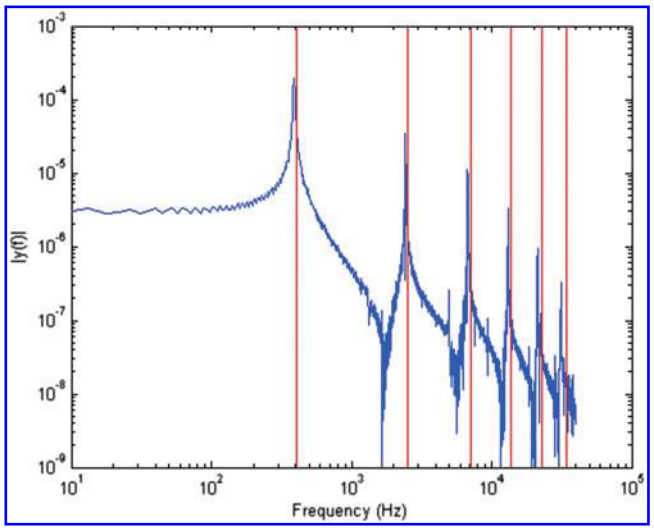

FIG. 5. The frequency response of a simulated cantilever beam with low damping clearly shows modal resonances that agree well with analytically calculated values (red overlay lines).

TABLE 2. THE MODAL FREQUENCIES OF A SIMULATED THIN BEAM AND ANALYTICALLY CALCULATED RESULTS, WITH DYNAMIC MASS-SPRING SIMULATION UNDER-PREDICTING SLIGHTLY DUE TO DAMPING AND HAVING A FINITE THICKNESS

Mode	*Analytical*	*Mass-spring*
1	389 Hz	404 Hz
2	2428 Hz	2531 Hz
3	6749 Hz	7087 Hz
4	13070 Hz	13890 Hz
5	21260 Hz	22960 Hz
6	31180 Hz	34290 Hz

Effects of local bond damping

By applying a combination of damping both to the individual bonds and to the voxels relative to ground, the solution converges quickly to steady state with very little numerical jitter. The additional local damping does not significantly affect the convergence speed but allows the solution to converge to a residual static error approximately seven orders of magnitude lower (Fig. 7). By suppressing jitter in this manner, both static and dynamic solutions are less susceptible to numerical instability.

Speedup of self-collision schemes

Different combinations of self-collision detection methods were directly compared using the test geometry shown in Figure 8. All materials in this setup were defined with a stiffness of 1 MPa. The red and blue materials each were assigned to have a thermal expansion coefficient with magnitude of 0.02, although one was positive and one negative. The temperature of the environment was then sinusoidally varied with an amplitude of 30 degrees, which corresponds to a 60% expansion and contraction of the red and blue materials 180 degrees out of phase. This sets up a periodic collision between the extremities that are repeatedly entering and exiting the assigned collision horizon.

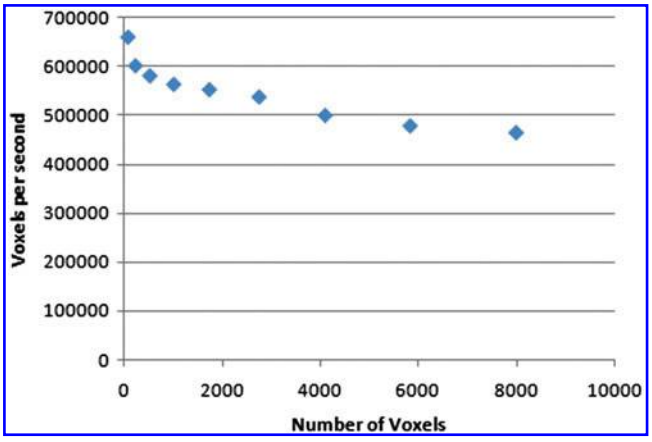

FIG. 6. The computational speed per voxel drops off slightly as the number of voxels in the simulation increases.

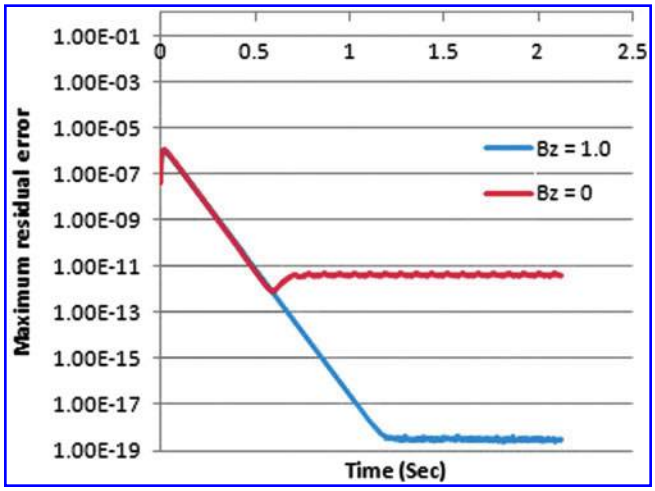

FIG. 7. Damping individual bonds critically ($B_z = 1.0$) lowers the noise floor by approximately seven orders of magnitude compared to the undamped case ($B_z = 0$).

The case of comparing the distance from all voxels to all other voxels in the structure at every timestep (All + Every) was included as a baseline. Comparing only surface voxels to each other at every timestep (Surf + Every) resulted in a very minor speedup. This is expected given the geometry chosen, because the majority of voxels in the structure are surface voxels. However, large gains in speed are realized when incorporating the collision horizon. Even when comparing all voxels to all voxels when recalculation is needed (All + Horizon), the simulation as a whole speeds up almost 6×. Again, minor acceleration is realized when considering only surface voxels with the collision horizon (Surf + Horizon) (Table 3).

It should be stressed that these results are merely representative. In cases with larger numbers of voxels, the bottleneck of the entire simulation is in collision detection. In this case, comparing all voxels to all other voxels is $O(N^2)$ (where N is the number of voxels), which dominates the $O(N)$ scaling of calculating forces and updating positions. In this case, precompiling a list of surface voxels and only using them in collision detection can reduce collision detection down to approximately $O(N^{1.5})$, although this depends on the geometry. Of course, if the geometry is thin such that all voxels are on the surface, no speedup will be observed.

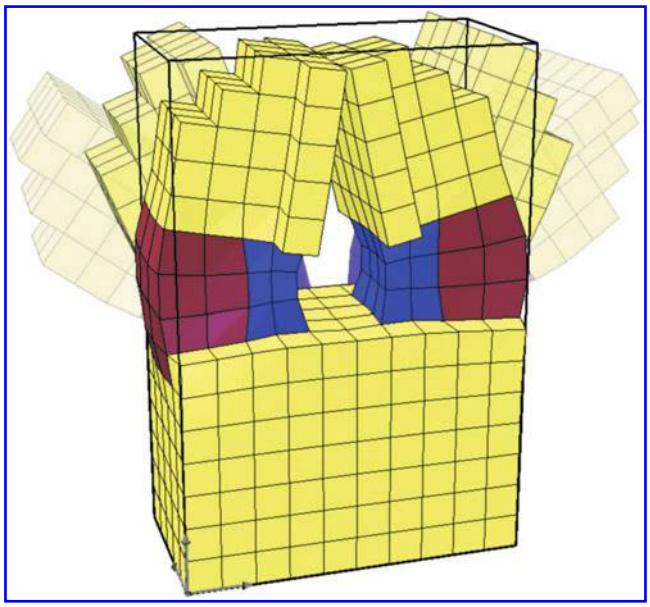

FIG. 8. An arbitrary clapper setup to test net iteration rates with different collision types. The red and blue materials change volume sinusoidally 180 degrees out of phase to provide the actuation.

Additionally, incorporating the collision horizon has widely variable effects on the speed of the simulation. In the extreme case of voxels moving very fast, no speedup is observed since the collision horizon may be exceeded at every timestep. However, this is only possible in extremely fast, rigid-body motion collisions, for which this simulation is not intended. On the opposite end of the spectrum, if the object is stationary or moving slowly the collision horizon will take many timesteps to be exceeded. Therefore, almost no collision detection calculations will be needed, resulting in dramatic acceleration of the simulation.

Demonstrations of Simple Volumetrically Actuated Mechanisms

Several demonstration scenes were created to illustrate the simulator in action. In the first scene, (Fig. 9a–c), an actuated beam kicks a ball into a bowling pin. The beam has a stiffness 10 times greater than the ball, which in turn has a stiffness 10 times greater than the bowling pin. The frequency of the red and blue volumetric actuation was selected such that the beam would swing in resonance. The second scene (Fig. 9d–f) shows a 2D layer of voxels falling and interacting with a fixed sphere as cloth would. In the third scene, a quadruped with periodic leg actuation walks forward using the nonlinearities of the surface friction with the floor as well as the side-to-side resonance of the head swinging back and forth. Animations of each example may be viewed online.[24] These illustrations demonstrate not just the dynamics and large deformation capabilities of the simulation, but also the use of volumetric actuation and the efficient collision system.

Conclusion

We have demonstrated a computationally efficient soft body simulator with applications in nonlinear material modeling and dynamic soft object simulation. By virtue of being voxel-based, this simulation can accurately model heterogeneous materials with differing stiffnesses and densities in a physically accurate environment, even when the materials are well-interspersed among each other. This enables modeling of gradient and composite materials. By incorporating a collision horizon that is updated only when needed, self-intersection is eliminated with low computational overhead. By implementing volumetric actuation, structures can be actuated without imposing arbitrary external forces to create self-contained mechanisms.

Several practical projects have already successfully utilized the simulation engine presented here. It has been used to evolve various locomoting soft robots by the authors[25] and their colleagues.[26] This has enabled a new level of computational design and analysis of soft robots. Fueled by the freely available graphical user interface (GUI) (VoxCad) and the intuitiveness of making 3D objects with voxels

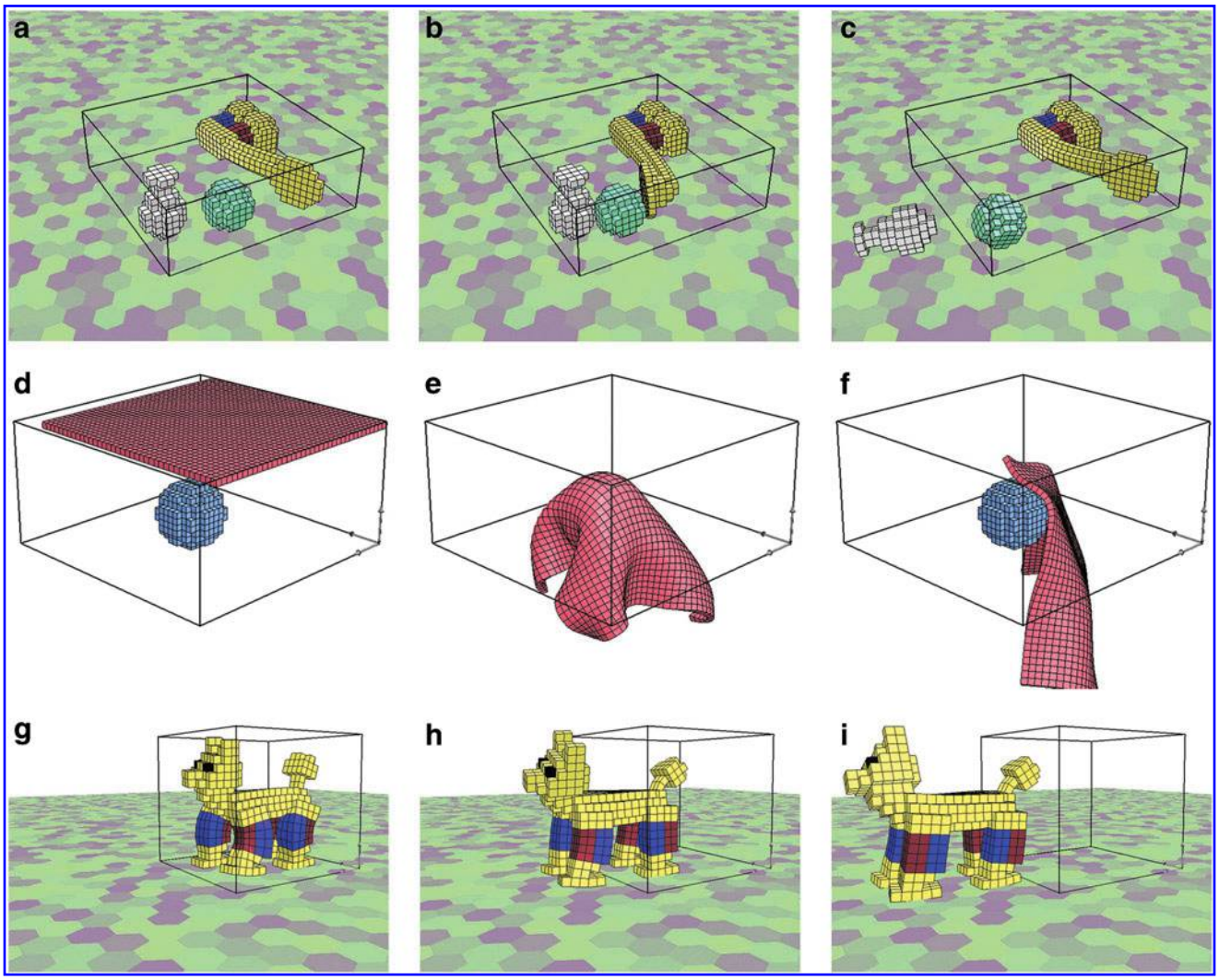

FIG. 9. Frames from demonstration scenes show a volumetrically actuated flexible beam swung in resonance to kick a soft ball into an even softer pin **(a–c)**. A 2D layer of voxels falling under gravity and interacting with a fixed sphere **(d–f)**, and a locomoting quadruped **(g–i)**.

(blocks), it has found educational applications as well. There is a growing online library of soft robotics design challenges and science fair projects for children ages 8 and up that involve designing and racing soft robots in VoxCad.[27] The multimaterial modeling capabilities have also been used for analyzing the static and time-varying properties of 3D-printed multimaterial objects. The programmable materials developed by Stratasys in their 4D printing research collaborations with the Massachusetts Institute of Technology utilized the large-deformation and volumetric actuation features presented here to design active joints and hinges.[28]

TABLE 3. ITERATION RATES FOR VARIOUS COLLISIONS DETECTION AND HANDLING SCHEMES (HIGHER IS BETTER)

Geometry	*Rate (iter/sec)*
All + every	140.4
Surf + every	140.6
All + horizon	834.4
Surf + horizon	835.4

All code and documentation is freely available online including a standalone GUI for editing and simulating objects in a real-time interactive environment. A brief overview of how to use the code is included in the appendix. This simulation opens the door to the design automation of a wide variety of nonlinear physical structures and mechanisms that were not possible with previous soft body physics simulation packages.

Acknowledgments

This work was supported in part by a National Science Foundation Graduate Research Fellowship, DARPA Open Manufacturing program and Stratasys Ltd. The work presented here does not represent the views of the sponsoring organizations.

References

1. Bathe K. Finite Element Procedures. Prentice Hall: Upper Saddle River, NJ; 2007.
2. Stratasys Ltd. Fullcure materials datasheets. www.stratasys.com/materials/polyjet (accessed Dec. 30, 2013).

3. Nealen A, Müller M, Keiser R, Boxerman E, Carlson M. Physically based deformable models in computer graphics. Comput Graph Forum 2006;25(4):809–836.
4. Bullet Physics. Bullet physics: Project summary, 2011. http://code.google.com/p/bullet/ (accessed Dec. 30, 2013).
5. PhysX. Physx features: Soft bodies, 2011. http://developer.nvidia.com/object/physx_features.html (accessed Dec. 30, 2013).
6. Eberhardt B, Weber A, Strasser W. A fast, flexible, particle-system model for cloth draping. IEEE Computer Graphics and Applications 1996;16:52–59.
7. Faloutsos P, Van de Panne M, Terzopoulos D. Dynamic free-form deformations for animation synthesis. Visualization and Computer Graphics, IEEE Transactions on Jul–Sep 1997;3(3):201–214.
8. Muller M, Dorsey J, McMillan L, Jagnow R, Cutler B. Stable real-time deformations. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '02. New York; ACM: 2002, pp. 49–54.
9. Teschner M, Heidelberger B, Muller M, Gross M. A versatile and robust model for geometrically complex deformable solids. In Computer Graphics International, 2004 Proceedings. June 2004;312–319.
10. Rivers AR, James DL. Fastlsm: fast lattice shape matching for robust real-time deformation. In ACM SIGGRAPH 2007 Papers, SIGGRAPH '07. New York; ACM: 2007.
11. James DL, Pai DK. Artdefo: accurate real time deformable objects. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99. New York; ACM Press/Addison-Wesley Publishing Co: 1999, pp. 65–72.
12. Cotins S, Delingette H, Ayache N. Real-time elastic deformations of soft tissues for surgery simulation. Visualization and Computer Graphics, IEEE Transactions on Jan–Mar 1999;5(1):62–73.
13. Wu X, Downes MS, Goktekin T, Tendick F. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. Computer Graphics Forum 2001;20(3):349–358.
14. Lipson H. A relaxation method for simulating the kinematics of compound nonlinear mechanisms. J Mech Des NY 2006;128(4):719–728.
15. Pedersen CBW, Buhl T, Sigmund O. Topology synthesis of large-displacement compliant mechanisms. Int J Number Methods Eng. 2001;50(12):2683–2705.
16. Nishiwaki S, Frecker MI, Min S, Kikuchi N. Topology optimization of compliant mechanisms using the homogenization method. Int J Numer Methods Eng. 1998;42(3):535–559.
17. Howell LL, Midha A, Norton TW. Evaluation of equivalent spring stiffness for use in a pseudo-rigid-body model of large-deflection compliant mechanisms. Journal of Mechanical Design 1996;118(1):126–131.
18. Saxena A, Ananthasuresh GK. Topology synthesis of compliant mechanisms for nonlinear force-deflection and curved path specifications. J Mech Des NY 2001;123(1):33–42.
19. Nesme M, Faure F, Payan Y. Hierarchical multi-resolution finite element model for soft body simulation. In Harders M, Szkely G (Eds). Biomedical Simulation, Vol. 4072 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg: 2006, pp. 40–47.
20. Shephard MS, Georges MK. Automatic three-dimensional mesh generation by the finite octree technique. Int J Numer Methods Eng. 1991;324(4):709–749.
21. Hassani B, Hinton E. A review of homogenization and topology opimization i–homogenization theory for media with periodic structure. Computers & Structures 1998;69(6):707–717.
22. Carrera E, Giunta G, Petrolo M. Beam Structures: Classical and Advanced Theories. New York; Wiley: 2011.
23. Volterra E, Zachmanoglou EC. Dynamics of Vibrations. New York; Charles E. Merrill Books, Inc: 1965.
24. VoxCAD homepage. www.voxcad.com (accessed Dec. 30, 2013).
25. Hiller J, Lipson H. Automatic design and manufature of soft robots. IEEE Transactions on Robotics 2012;28(2):457–466.
26. Cheney N, MacCurdy R, Clune J, Lipson H. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013), 2013.
27. Finio B. Robot race! Use a computer to design, simulate, and race robots with voxcad. www.sciencebuddies.org/science-fair-projects/project_ideas/robotics_p016.shtml (accessed Dec. 30, 2013).
28. Stratasys Ltd. 4d printing: Revolutionizing material form and control. www.stratasys.com/industries/education/4d-printing-project (accessed Dec. 30, 2013).

Address correspondence to:
Jonathan Hiller
Sibley School of Mechanical and Aerospace Engineering
Cornell University
238 Upson Hall
Ithaca, NY 14850

E-mail: jdh74@cornell.edu

(*Appendix follows* →)

Appendix

Simple implementation of the simulator

A minimal C++ program is provided here to demonstrate the ease of coding a bare-bones dynamic simulation. A 5 mm × 10 mm × 5mm beam with a material stiffness of 10 MPa is cantilevered, and a 1 kN force is applied to the free end.

```
CVX_Object Object; //Voxel object
CVX_Environment Environment; //Environment object
CVX_Sim Simulator; //Simulator object


Object.InitializeMatter(0.001f, 5, 10, 5); //Creates a 5×10×5 workspace of 1 mm voxels
int MatIndex=Object.AddMat(''Material1'', 10000000.0f, 0.3f); //Adds a 10 MPa material to the palette
for (int x=0; x<5; x++){
  for (int y=0; y<10; y++){
    for (int z=0; z<5; z++){
      Object.SetMat(x, y, z, MatIndex); //Sets each voxel to 10 MPa material
    }
  }
}
Environment.AddObject(&Object); //Imports the object into the environment
Environment.AddFixedRegion(Vec3D(0, 0, 0), Vec3D(1.0f, 0.01f, 1.0f)); //Fixes the −Y plane
Environment.AddForcedRegion(Vec3D(0, 0.99, 0), Vec3D(1.0f, 0.01f, 1.0f), Vec3D(0, 0, −1000.0f));
//Adds a downward force of 1 kN to the +Y plane
Simulator.Import(&Environment); //Imports the environment into the simulator
Simulator.SetSlowDampZ(0.013); //Sets the global damping ratio to an appropriate value
for(int i=0; i<400; i++)Simulator.TimeStep(); //Simulates 400 timesteps
```

Selected function definitions from dynamic voxel simulation classes

A small subset of functions used in the voxel simulator are documented here. With only these functions, a dynamic simulation may be created and run. Complete documentation of the source code is available at www.voxcad.com

Class CVX_Object. Describes the geometry and materials of a voxel object.

```
void CVX_Object::InitializeMatter (float iLattice_Dim, int xV, int yV, int zV)
```

Initializes voxel object with the specified voxel size and default lattice. A cubic lattice is assumed at the provided inter-voxel lattice dimension.

in	iLattice_Dim	The base lattice dimension between adjacent voxels in meters.
in	xV	The number of voxels in the X dimension of the workspace.
in	yV	The number of voxels in the Y dimension of the workspace.
in	zV	The number of voxels in the Z dimension of the workspace.

```
int CVX_Object::AddMat (std::string Name, double EMod, double PRatio, std::string *RetMessage=
NULL)
```

Appends a material to the palette. Returns the index within the palette that this material was created. This index may change if materials are deleted from the palette. All colors and physical properties besides elastic modulus and Poisson's ratio are set to defaults.

in	Name	The name of the material to add. If the name is already in use a variant will be generated.
in	EMod	The elastic modulus of the new material in Pascals.
in	PRatio	The Poisson's ratio of the new material.
out	RetMessage	Pointer to an initialized string. Messages generated in this function will be appended to the string.

```
bool CVX_Object::SetMat (int x, int y, int z, int MatIndex) [inline]
```

Sets a single voxel to the specified material. Returns true if successful. Returns false if provided indices are outside of the workspace, or the material index is not contained within the current palette.

in	x	Integer X index of voxel location to set.
in	y	Integer Y index of voxel location to set.
in	z	Integer Z index of voxel location to set.
in	MatIndex	Specifies the index within the material palette at which to set this voxel.

Class CVX_Environment. Describes the physical environment and boundary conditions for the voxel object.

```
void CVX_Environment::AddObject (CVX_Object *pObjIn) [inline]
```

Links a voxel object to this environment. Only one voxel object may be linked at a time.

in	pObjIn	Pointer to an initialized voxel object to link to this simulation.

```
void CVX_Environment::AddFixedBc (Vec3D<>& Location, Vec3D<>& Size, char DofToFix=DOF_ALL)
```

Adds a region of voxels to be fixed to ground in the specified degrees of freedom. All voxels touching this region will be affected in the simulation.

in	Location	The corner of the region closest to the origin. Specified as a percentage (in X, Y, and Z respectively) of the overall workspace. (Location.x, Location.y, and Location.z each have a range of [0.0, 1.0]).
in	Size	The size of the region. Specified as a percentage (in X, Y, and Z respectively) of the overall workspace. (Size.x, Size.y, and Size.z each have a range of [0.0, 1.0]).
in	DofToFix	The degree(s) of freedom to fix, specified by an 8-bit field where each set bit indicates a fixed degree of freedom. Translation degrees of freedom: DOF_X = 0x01, DOF_Y = 0x02, and DOF_Z = 0x04. Rotational degrees of freedom: DOF_TX = 0x08, DOF_TY = 0x10, and DOF_TZ = 0x20. By extension (and for convenience), DOF_ALL = 0x3F and DOF_NONE = 0x00 are also defined. Individual degrees of freedom may be combined in any way.

```
void CVX_Environment::AddForcedBc (Vec3D<>& Location, Vec3D<>& Size, Vec3D<>& Force, Vec3D<>&
Torque=Vec3D<>(0,0,0))
```

Applies a force and/or torque to a region of voxels. All voxels touching this region will be affected. The provided force or torque vector will be divided equally among all voxels.

in	Location	The corner of the region closest to the origin. Specified as a percentage (in X, Y, and Z respectively) of the overall workspace. (Location.x, Location.y, and Location.z each have a range of [0.0, 1.0]).
in	Size	The size of the region. Specified as a percentage (in X, Y, and Z respectively) of the overall workspace. (Size.x, Size.y, and Size.z each have a range of [0.0, 1.0]).
in	Force	The force to be distributed across this region in Newtons. The force is divided equally among all voxels in the specified region.
in	Torque	The torque to be distributed across this region in Newton-meters. The torque is divided equally among all voxels in the specified region. Currently torques are only applied if all rotational degrees of freedom are unfixed.

Class CVX_Sim. Contains the current physical state of the voxel object and advances the simulation.

```
void CVX_Sim::Import (CVX_Environment *pEnvIn=NULL, CMesh *pSurfMeshIn=NULL, std::string *RetMessage=NULL)
```

Imports a physical environment into the simulator. The environment should have been previously initialized and linked with a single voxel object. This function sets or resets the entire simulation with the new environment.

in	pEnvIn	A pointer to initialized CVX_Environment to import into the simulator.
in	pEnvpSurfMeshIn	A pointer to initialized CMesh object containing a smooth surface to paste onto the underlying voxels. If NULL, a surface will be automatically generated.
out	RetMessage	A pointer to initialized string. Output information from the import function is appended to this string.

```
void CVX_Sim::SetSlowDampZ (double SlowDampIn) [inline]
```

Sets the damping ratio that slows downs voxels. When this is nonzero, each voxel is damped (based on its mass and stiffness) to ground. Range is [0.0, 1.0]. Values greater than 1.0 may cause numerical instability.

out	SlowDampIn	Damping ratio for damping each voxel relative to ground.

```
void CVX_Sim::SetCollisionDampZ (double ColDampZIn) [inline]
```

Sets the damping ratio for voxels in a colliding state. When this is nonzero, each voxel is damped (based on its mass and stiffness) according to the relative penetration velocity. Range is [0.0, 1.0]. Values greater than 1.0 may cause numerical instability.

out	ColDampZIn	Collision-damping ratio for colliding voxels.

```
void CVX_Sim::SetBondDampZ (double BondDampZIn) [inline]
```

Sets the damping ratio for connected voxels. When this is nonzero, each voxel is damped (based on its mass and stiffness) according to its relative velocity to the other voxel in each bond. Range is [0.0, 1.0]. Values greater than 1.0 may cause numerical instability.

out	BondDampZIn	Damping ratio between each connected voxel.

```
bool CVX_Sim::TimeStep (std::string *pRetMessage=NULL)
```

Advances the simulation one-time step. Given the current state of the simulation (voxel positions and velocities) and information about the current environment, this function advances the simulation by the maximum stable timestep. Returns true if the time step was successful, false otherwise.

out	pRetMessage	Pointer to an initialized string. Messages generated in this function will be appended to the string.

This article has been cited by:

1. Yuen-Shan Leung, Tsz-Ho Kwok, Xiangjia Li, Yang Yang, Charlie C. L. Wang, Yong Chen. 2019. Challenges and Status on Design and Computation for Emerging Additive Manufacturing Technologies. *Journal of Computing and Information Science in Engineering* **19**:2, 021013. [Crossref]
2. Marco Livesu, Daniela Cabiddu, Marco Attene. 2019. slice2mesh: A meshing tool for the simulation of additive manufacturing processes. *Computers & Graphics* **80**, 73-84. [Crossref]
3. Johnson Chris, Philippides Andrew, Husbands Philip. 2019. Simulating Soft-Bodied Swimmers with Particle-Based Physics. *Soft Robotics* **6**:2, 263-275. [Abstract] [Full Text] [PDF] [PDF Plus] [Supplementary Material]
4. Thomas Morzadec, Damien Marcha, Christian Duriez. Toward Shape Optimization of Soft Robots 521-526. [Crossref]
5. Aman Chandra, Jekanthan Thangavelautham. Modular Inflatable Composites for Space Telescopes 1-9. [Crossref]
6. Jakub Lengiewicz, Paweł Hołobut. 2019. Efficient collective shape shifting and locomotion of massively-modular robotic structures. *Autonomous Robots* **43**:1, 97-122. [Crossref]
7. Sam Kriegman, Nick Cheney, Josh Bongard. 2018. How morphological development can guide evolution. *Scientific Reports* **8**:1. . [Crossref]
8. Bruno César Feltes, Bruno Iochins Grisci, Joice de Faria Poloni, Márcio Dorn. 2018. Perspectives and applications of machine learning for evolutionary developmental biology. *Molecular Omics* **14**:5, 289-306. [Crossref]
9. Zhang Hongying, Kumar A. Senthil, Fuh Jerry Ying Hsi, Wang Michael Yu. 2018. Design and Development of a Topology-Optimized Three-Dimensional Printed Soft Gripper. *Soft Robotics* **5**:5, 650-661. [Abstract] [Full Text] [PDF] [PDF Plus] [Supplementary Material]
10. Corucci Francesco, Cheney Nick, Giorgio-Serchi Francesco, Bongard Josh, Laschi Cecilia. 2018. Evolving Soft Locomotion in Aquatic and Terrestrial Environments: Effects of Material Properties and Environmental Transitions. *Soft Robotics* **5**:4, 475-495. [Abstract] [Full Text] [PDF] [PDF Plus] [Supplementary Material]
11. François Schmitt, Olivier Piccin, Laurent Barbé, Bernard Bayle. 2018. Soft Robots Manufacturing: A Review. *Frontiers in Robotics and AI* **5**. . [Crossref]
12. Nick Cheney, Josh Bongard, Vytas SunSpiral, Hod Lipson. 2018. Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface* **15**:143, 20170937. [Crossref]
13. Germain Sossou, Frédéric Demoly, Ghislain Montavon, Samuel Gomes. 2018. Design for 4D printing: rapidly exploring the design space around smart materials. *Procedia CIRP* **70**, 120-125. [Crossref]
14. E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt, C. Duriez. 2017. Software toolkit for modeling, simulation, and control of soft robots. *Advanced Robotics* **31**:22, 1208-1224. [Crossref]
15. Francesco Corucci, Nick Cheney, Sam Kriegman, Josh Bongard, Cecilia Laschi. 2017. Evolutionary Developmental Soft Robotics As a Framework to Study Intelligence and Adaptive Behavior in Animals and Plants. *Frontiers in Robotics and AI* **4**. . [Crossref]
16. Pawd Holobut, Jakub Lengiewicz. Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning 2103-2109. [Crossref]
17. Jenett Benjamin, Calisch Sam, Cellucci Daniel, Cramer Nick, Gershenfeld Neil, Swei Sean, Cheung Kenneth C.. 2017. Digital Morphing Wing: Active Wing Shaping Concept Using Composite Lattice-Based Cellular Structures. *Soft Robotics* **4**:1, 33-48. [Abstract] [Full Text] [PDF] [PDF Plus] [Supplementary Material]
18. Benjamin Jenett, Christine Gregg, Daniel Cellucci, Kenneth Cheung. Design of multifunctional hierarchical space structures 1-10. [Crossref]
19. Holly M. Jackson. Topological Optimization of a Cuboct Truss Structure Using a Genetic Algorithm . [Crossref]
20. Zhicong Deng, Martin Stommel, Weiliang Xu. Soft Robotics Technology and a Soft Table for Industrial Applications 397-409. [Crossref]
21. Tomi J. Ylikorpi, Aarne J. Halme, Pekka J. Forsman. 2017. Dynamic modeling and obstacle-crossing capability of flexible pendulum-driven ball-shaped robots. *Robotics and Autonomous Systems* **87**, 269-280. [Crossref]
22. Weeger Oliver, Kang Yue Sheng Benjamin, Yeung Sai-Kit, Dunn Martin L.. 2016. Optimal Design and Manufacture of Active Rod Structures with Spatially Variable Materials. *3D Printing and Additive Manufacturing* **3**:4, 204-215. [Abstract] [Full Text] [PDF] [PDF Plus]

23. Ross Dylan, Nemitz Markus P., Stokes Adam A.. 2016. Controlling and Simulating Soft Robotic Systems: Insights from a Thermodynamic Perspective. *Soft Robotics* **3**:4, 170-176. [Abstract] [Full Text] [PDF] [PDF Plus] [Supplementary Material]
24. Tomoya Kimura, Ziqiao Jin, Ryuma Niiyama, Yasuo Kuniyoshi. Evolving soft robots to execute multiple tasks with Combined-CPPN-NEAT 409-414. [Crossref]
25. C. Duriez, E. Coevoet, F. Largilliere, T. Morales-Bieze, Z. Zhang, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt. Framework for online simulation of soft robots with optimization-based inverse model 111-118. [Crossref]
26. Josie Hughes, Utku Culha, Fabio Giardina, Fabian Guenther, Andre Rosendo, Fumiya Iida. 2016. Soft Manipulators and Grippers: A Review. *Frontiers in Robotics and AI* **3**. . [Crossref]
27. Stefano Mintchev, Dario Floreano. 2016. Adaptive Morphology: A Design Principle for Multimodal and Multifunctional Robots. *IEEE Robotics & Automation Magazine* **23**:3, 42-54. [Crossref]
28. Nick Cheney, Hod Lipson. 2016. Topological evolution for embodied cellular automata. *Theoretical Computer Science* **633**, 19-27. [Crossref]
29. Martin Stommel, Weiliang Xu. 2016. Optimal, Efficient Sequential Control of a Soft-Bodied, Peristaltic Sorting Table. *IEEE Transactions on Automation Science and Engineering* **13**:2, 858-867. [Crossref]
30. Nick Cheney, Josh Bongard, Hod Lipson. Evolving Soft Robots in Tight Spaces 935-942. [Crossref]
31. Jürgen Roßmann, Michael Schluse, Malte Rast, Eric Guiffo Kaigom, Torben Cichon, Michael Schluse. Simulation Technology for Soft Robotics Applications 100-119. [Crossref]
32. Utku Culha, Umar Wani, Surya G. Nurzaman, Frank Clemens, Fumiya Iida. Motion pattern discrimination for soft robots with morphologically flexible sensors 567-572. [Crossref]
33. Utku Culha, Surya Nurzaman, Frank Clemens, Fumiya Iida. 2014. SVAS3: Strain Vector Aided Sensorization of Soft Structures. *Sensors* **14**:7, 12748. [Crossref]