Multi-Resolution Modeling of Shapes in Contact

YIJING LI, University of Southern California, USA
JERNEJ BARBIČ, University of Southern California, USA

We describe an efficient method to model shapes undergoing contact and self-contact. Previous shape modeling methods mostly focused on deformations (without contact), and, if used directly for contact, suffer from excessively long calculation times when new contacts are detected. In our work, we demonstrate fast, output-sensitive shape modeling that does not substantially degrade when new contacts are detected and that degrades gracefully with contact complexity, even for complex geometries. We achieve this by constructing a rotationally invariant linear-precision multi-resolution hierarchy of shape deformation bases. Inspired by the active set method, we propose a new contact model suitable for shape modeling that greatly outperforms prior work in contact quality and smoothness. Our method requires no extensive precomputation and works with triangle meshes embedded in solid tetrahedral meshes. We apply our method to the widely used as-rigid-as-possible energy, enabling modeling of shapes in contact, with arbitrarily large rotations, smoothness and locality.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; **Collision detection**.

Additional Key Words and Phrases: shape modeling, hierarchical, multi-resolution, contact, subspace

1 INTRODUCTION

Geometric shape modeling is a fundamentally important task in computer graphics and related fields. Previous methods have largely focused on how to model shapes using various deformation energies, typically by prescribing a set of handles which the user can employ to adjust the shape. In many applications, however, shapes undergo contact or self-contact. There has been little work to address contact for geometric shape modeling. We describe an efficient, output-sensitive method to model shapes undergoing contact and self-contact. Our examples include corrective shapes in pose-space deformation for character animation where limbs can penetrate bodies, modeling shapes of trees, and editing the shape of clothing in contact with the human body. In computer animation, our work could potentially also be used to improve blendshapes for face animation, by resolving mouth and eye collisions.

Contact-awareness can in principle be added to any handle-based deformation method: simply add a new handle to the system whenever a new contact is detected, constraining the handle to some nearest collision-free position. The first issue with such a "naive" approach is that it is suitable only for contact against external objects. For self-contact, it is not clear where the displaced handles should be positioned. Naive placements leave gaps, cause penetrations or produce non-smooth deformations at the collision sites. The second issue is that adding handles is typically a very expensive operation, as it often requires re-calculating the shape functions, or solving large systems of equations. For example, the widely-used bounded biharmonic weights require re-solving an optimization problem on the mesh in order to add a new handle. This is problematic in the presence of time-varying contacts where new contacts may appear or disappear at every frame.

We alleviate these issues by constructing a multi-resolution hierarchy of shape deformation bases that is both *rotationally invariant* and *linearly precise*. We enable smooth self-contact handling using our constraint formulation (Section 5.2). Our hierarchy starts with the constant and linear precision non-hierarchical subspace proposed by Wang et al. [Wang et al. 2015], serving as the first level of the hierarchy. We then design a novel multi-resolution basis hierarchy that maintains constant and linear precision, and that can be used to add more degrees of freedom locally to address contact. At

Authors' addresses: Yijing Li, University of Southern California, Los Angeles, CA, USA, yijingl@usc.edu; Jernej Barbič, University of Southern California, Los Angeles, CA, USA, jnb@usc.edu.

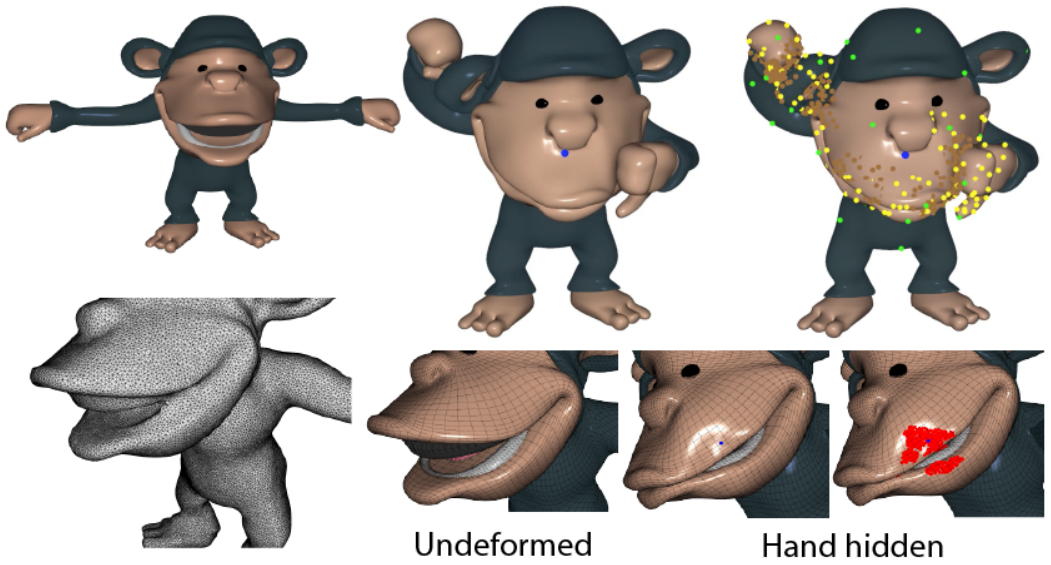

Fig. 1. **Example of editing the chimpanzee with multiple contact sites.** We close the chimpanzee's mouth, forming a mouth self-contact. We also pull one of its hands to collide with its face, and the other to collide with an ear. Upper-left: input triangle mesh, lower-left: tet mesh. Upper-middle: deformed mesh with collision resolved, upper-right: activated bases: green (L0), yellow (L1) and brown (L2). User handles are shown in blue. Lower-middle and lower-right: zoom-in on the collision between the mouth and the hand. Contact points are highlighted in red. The maximum computation time at any frame in our method is 109× smaller to the maximum time in full-space modeling.

runtime, smooth bases of various localities can be efficiently added to the existing basis. Based on the current contact complexity, the runtime traversal can be interrupted at any level, making it possible to fit the calculation into prescribed computational budgets.

We also propose a new contact model suitable for geometric shape modeling. Our model is inspired by the active set method; but is designed for shape modeling, which is characterized by a lack of need for modeling dynamics, and a focus on smoothness, stability and controllability. We utilize the special system matrix format to accelerate computation, and also use multi-core computing whereby the "updater" thread continually updates the internal data structures to maintain fast update rates during modeling of transient contact. We apply our hierarchy and contact model to geometric modeling of static shapes with the widely used as-rigid-as-possible (ARAP) energy. We present several challenging examples of complex geometry in self-contact (Figures 1,3) and contact with external objects (Figure 15). We demonstrate that our method outperforms full non-hierarchical shape modeling by at least an order of magnitude. We also experimentally compare our contact model to the state of the art for contact-aware shape modeling [Harmon et al. 2011], and demonstrate significant improvements in shape quality (Figure 2).

2 RELATED WORK

In this section, we introduce closely related work on geometric shape modeling, hierarchical modeling, contact handling and physically based methods, and discuss the relationship to our method.

2.1 Geometry editing

Interactive editing of geometry has long been an important topic in graphics research. For good surveys, please refer to Botsch and Sorkine [Botsch and Sorkine 2008] and the SIGGRAPH course notes by Alexa et al. [Alexa et al. 2006]. Many editing methods are based on differential surface properties, such as Laplacian surface editing [Sorkine et al. 2004] and as-rigid-as-possible (ARAP) manipulation [Igarashi et al. 2005; Sorkine and Alexa 2007]. To reduce the computational burden and improve controllability, one can use Free-Form Deformation (FFD), cages and skeletons [Ju et al. 2005; Sederberg and Parry 1986]. Recently, Jacobson et al. [Jacobson et al. 2011] presented bounded biharmonic weights (BBW) for linear blend skinning, which support various control structures, including points, bones and cages. Wang et al. [Wang et al. 2015] built a linear subspace to produce smooth, geometry-aware shapes using handles and rigid regions. Due to its good geometric properties and fast computation, we base our hierarchy of deformations on Wang's energy. These previous methods do not incorporate contact handling. An interesting work of Vaillant et al. [Vaillant et al. 2013] used implicit distance fields to form a collision free mesh. However, their method is based on skinning and does not handle general shape deformations.

2.2 Multi-resolution hierarchies

Multi-resolution hierarchies are commonly used for modeling, energy minimization and simulation. Boier-Martin et al. [Boier-Martin et al. 2005] surveyed subdivision methods and multiscale modeling. Botsch et al. [Botsch et al. 2006] and Winkler et al. [Winkler et al. 2010] performed hierarchical shape matching. Coarse meshes are commonly used to accelerate nonlinear optimization [Fröhlich and Botsch 2011; Manson and Schaefer 2011; Umetani et al. 2011]. Several researchers studied multi-resolution FEM [Capell et al. 2002; Debunne et al. 2001; Grinspun et al. 2002], and the multigrid method [Otaduy et al. 2007; Tamstorf et al. 2015; Zhu et al. 2010]. James and Pai [James and Pai 2003] modeled multi-resolution deformations using Green's functions and the capacitance matrix algorithm. Their method is limited to linear elasticity which produces artifacts under large rotations. We give a multi-resolution basis hierarchy suitable for geometric shape modeling with the ARAP energy in the presence of localized contact.

Malgat et al. [Malgat et al. 2015] gave a technique to combine two or more deformation models in separate hierarchical layers. Their core innovation is a generally very useful framework whereby deformations due to mechanical models at deeper layers are properly locally transformed with time-varying affine transformations imposed by the ancestor layers in the hierarchy. They describe how to create mass matrices and equations of motions consistent with these assumptions; and give a dynamics filter to avoid redundant DOFs from different layers. In our work, we formulate the total world-coordinate position of any mesh vertex as a linear superposition of fixed shape vectors that *do not rotate or affinely transform at runtime* (Equation 15). We therefore do not need the local affine transformability accommodated by Malgat's work. No filters are required and our basis hierarchy automatically avoids redundant DOFs (Section 4.2.3). Malgat et al. [Malgat et al. 2015] demonstrated applications to collision handling. However, in their examples, they manually specified parts of the mesh whereby detailed deformations are needed due to collisions. In our work, we activate and de-activate the local detail automatically, and demonstrate how this can be done efficiently when the collision sites are not known in advance.

2.3 Collision detection and response

A survey of real-time collision detection was given by Ericson [Ericson 2004]. In haptics and surgery simulation, real-time contact feedback is widely studied [Lin and Otaduy 2008]. Stable 6-DOF haptic manipulation of rigid models can be achieved with implicit integration [Otaduy and

Lin 2006]. Rigid body and reduced deformable object collisions can also be simulated at haptic rates [Barbič and James 2008; Kaufman et al. 2008; Yeung et al. 2016]. Barbic and James [Barbič and James 2008] employed a hierarchical point tree to provide an upper limit on collision detection time. Our system exploits a similar level-by-level collision detection, but demonstrates how to efficiently combine it with hierarchical deformation bases for efficient shape editing. Allard et al. [Allard et al. 2010] introduced volume constraints that can control contact response detail. Their method operates on the full model, whereas we use a reduced model for faster computation. If their method was directly applied to our system, their volume-constraint-based penalty forces would change the entire reduced system matrix at every iteration. In our work, we can achieve good computational speedups compared to the full method because we designed our method so that the reduced system matrix does not change unless there is a change of basis. Even then, only the part corresponding to the changed basis changes. Talvas et al. [Talvas et al. 2015] used nonuniform pressure on volume constraints for fast hand simulation with contact. Their method can be used on top of our hierarchical constrained system to simplify constraints.

For reduced simulation, several publications added more degrees of freedom into contact regions [Harmon and Zorin 2013; Teng et al. 2015]. Different from these methods, we create a hierarchy of bases to allow better control on tradeoff between contact resolution and speed. Our iterative algorithm is designed for shape editing and modeling, while theirs are used in reduced physical simulation. We use constraint-based methods to handle collision, while they use penalty forces that are known to be difficult in determining penalty stiffness. Recently, Erleben [Erleben 2018] assessed various contact normal methods and conclude that no good local normals are available. Our method shares this limitation but it is possible to alleviate the problem by smoothing normals.

2.4 Local/global simulation

The recent introduction of projective dynamics [Bouaziz et al. 2014], its follow-up methods [Brandt et al. 2018; Liu et al. 2016] and ADMM [Narain et al. 2016] makes it possible to perform plausible real-time physically based simulation on complex meshes. These methods are also directly applicable to interactive shape modeling, and follow the same local/global optimization mechanism as the ARAP method used in our work. Therefore, our basis hierarchy could be combined with projective dynamics / ADMM methods to replace the ARAP energy. Note that Brandt et al. [Brandt et al. 2018] used model reduction with local/global optimization. Their work, however, cannot handle self-collisions, whereas our method works with both external and self-collisions; requiring a more complex constraint handling mechanism (as presented in our work).

Generally, our framework differs from realtime deformable object simulation methods with collision handling in that our application is geometric deformation, which does not require a *per-frame* accurate result, but focuses on interactivity and convergence. In contrast, simulation methods focus on speed and physical plausibility. Similar energies such as [Chao et al. 2010] could also be used. For shape modeling, we need speed, control, and gradual refinement of the shape. Our system satisfies these requirements. Our method enables direct local control of shapes, gradually refines the shape, and produces smooth and shape-aware deformations. As per collision constraints, our formulation is similar to other vertex-constraint-based simulation methods. The difference is that we can exploit the system structure to optimize computation (Section 5.2 and 6).

2.5 Shape modeling with contact

There are not many papers on modeling geometric shapes in contact. Gain and Dodgson [Gain and Dodgson 2001] developed a self-intersection-free FFD scheme. Harmon et al. [Harmon et al. 2011] resolved contacts by minimizing space-time interference volumes, using the controls of the given modeling tool. Their contact resolution method is essentially a post-processing step to modeling.

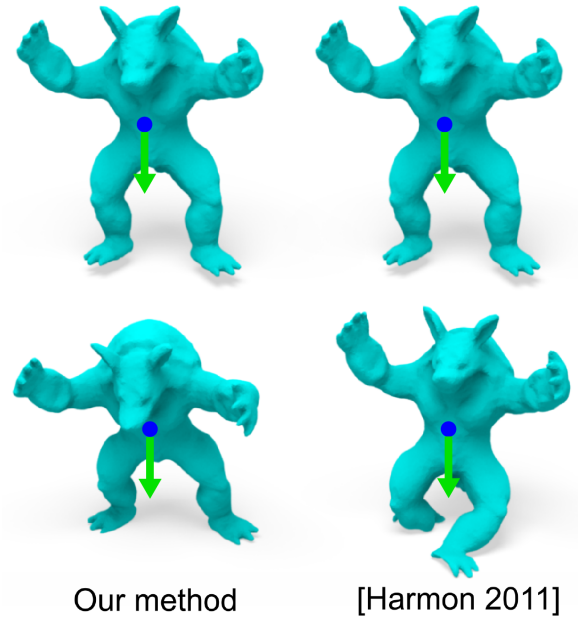

Fig. 2. **Comparison to Harmon's method:** Our algorithm creates more natural and smoother shapes. We add a handle to the armadillo and pull it to collide with the ground plane. Our approach (left column) then solves for smoothness and collisions together, whereas Harmon's method (right column) fixes collisions only as a post-process. With our method, the feet of the armadillo bend naturally in the initial iterations; slowly, the entire body leans forward to minimize ARAP energy. Harmon's method projects the subspace control points to the closest collision-free states, moving only control points at the feet up, creating unnatural shapes.

Our system integrates contact handling directly into the modeling loop and provides the flexibility of adding localized bases to address local contact handling. We compare to Harmon's method in Figure 2 and Section 7, demonstrating that our method is more stable and produces smoother shapes. Jacobson et al. [Jacobson et al. 2013] used generalized winding numbers to perform robust inside-outside segmentation of input geometry. Sacht et al. [Sacht et al. 2013] separated the colliding components for sphere-topology surfaces. Ye and Zhao [Ye and Zhao 2012] used the intersection contour minimization method to remove collisions in given static shapes. These last three methods discussed collisions in the context of shape processing, but did not demonstrate a method to interactively edit shapes with contact handling.

3 SHAPE MODELING

We first describe our shape modeling in the full space (without reduction), subject to arbitrary user and contact constraints. We adopt the widely-used as-rigid-as-possible (ARAP) energy [Sorkine and Alexa 2007] to measure deformation smoothness. Although ARAP is the main energy used in our work, we note that we also implemented and tested other energies, such as the volumetric strain energy combined with Projective Dynamics [Bouaziz et al. 2014]. Let $\mathbf{p} \in \mathbb{R}^{n\times 3}$ contain the deformed mesh vertex positions. The user controls the shape by prescribing the positions of one or more user handles placed at arbitrary mesh vertices, $\mathbf{S}\mathbf{p} = \mathbf{s}$, where $\mathbf{S} \in \mathbb{R}^{h\times n}$ is the selection matrix that selects h handle indices in $\mathbf{p}$, and $\mathbf{s} \in \mathbb{R}^{h\times 3}$ are the handle positions controlled by the user. At each iteration, we perform the ARAP local/global optimization [Sorkine and Alexa 2007].

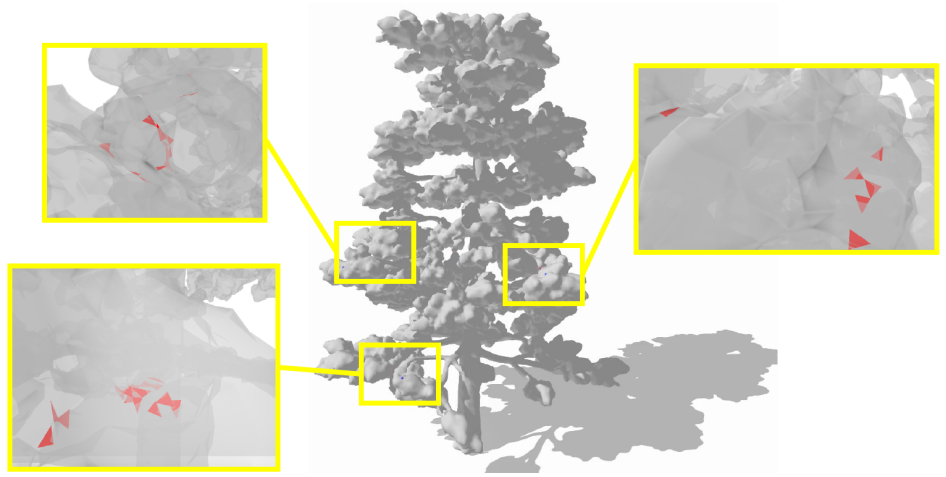

Fig. 3. **Editing the Huangshan pine tree with collisions.** Three handles (in highlighted contact regions) are added on three branches to pull them against other branches, creating self-collisions. Branches are displayed as translucent in the highlighted regions. Triangles in contact are shown in red. Our hierarchical subspace algorithm provides shorter and stabler computation times compared to the full method.

The local step finds the best rotation around each vertex, via polar decomposition. The global step solves a linear system

$$\begin{bmatrix} \mathbf{L} & \mathbf{S}^T & \mathbf{C}^T \\ \mathbf{S} & & \\ \mathbf{C} & & \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \lambda_S \\ \lambda_C \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{s} \\ \mathbf{c} \end{bmatrix}, \text{ where} \tag{1}$$

$$\mathbf{L} = \begin{bmatrix} \mathsf{L} & & \\ & \mathsf{L} & \\ & & \mathsf{L} \end{bmatrix}, \ \mathbf{S} = \begin{bmatrix} \mathsf{S} & & \\ & \mathsf{S} & \\ & & \mathsf{S} \end{bmatrix}. \tag{2}$$

Matrix $\mathsf{L} \in \mathbb{R}^{n \times n}$ is the cotangent Laplacian matrix [Sorkine and Alexa 2007], and can be pre-factored. The right-hand side matrix $\mathbf{b} \in \mathbb{R}^{3n}$ depends on local rotations R_i (Appendix A). Vector $\mathbf{p} \in \mathbb{R}^{3n}$ stacks the vertex positions at each dimension sequentially, $\mathbf{p} = [\mathsf{p}_{[:0]}^T \mathsf{p}_{[:1]}^T \mathsf{p}_{[:2]}^T]^T$, where $\mathsf{p}_{[:i]}$ represents the i-th column of p. Similar conventions are followed for $\mathbf{b}$ and $\mathbf{s}$. Throughout the paper, we use bold-face such as $\mathbf{p} \in \mathbb{R}^{3n}$ to denote a vector containing mesh vertex displacements (first x-DOFs for all vertices, then y-DOFs, then z-DOFs), as well as matrices that operate on such vectors. We use the sans-serif typeface such as $\mathsf{p} \in \mathbb{R}^{n \times 3}$ to denote arranging the displacements into a $n \times 3$ matrix. To resolve contact, we add planar contact constraints $\mathbf{Cp} = \mathbf{c}$, where we will enforce the additional requirement that the constraints are not pulling but only pushing. We will discuss this further in Section 5. The Lagrange multipliers are denoted by λ_S and λ_C.

The local/global optimization is effectively a block-coordinate descent, which in practice decreases the energy at each iteration. One benefit of the ARAP energy is that the x, y, z dimensions are decoupled, enabling us to solve three pre-factored $n \times n$ systems in parallel. However, since contact normals can point in arbitrary directions, and we use sliding constraints for better contact resolution, the contact constraints couple all three dimensions. The user can freely move the user handles while the algorithm deforms the rest of the mesh to minimize the ARAP energy and resolve contact (Figure 3). We note that an alternative approach would be to treat the positional constraints as

Dirichlet boundary conditions, at the cost of re-factoring the system each time a new constraint is added. We use Lagrange multipliers because they support non-positional (tangential sliding) constraints needed for contact, and because they enable a faster system update method (Section 6).

3.1 Modeling in a Shape Subspace

The full optimization method given in Section 3 does not run at interactive rates for detailed geometry. To make the modeling tractable, we solve the system in a hierarchical shape subspace. Other advantages of subspace methods are faster convergence compared to a full method, and a decrease in the spiky artifacts around manipulated handles. Suppose the mesh vertex positions can be expressed as $\mathsf{p} = \mathsf{U}\mathsf{v}$, where $\mathsf{U} \in \mathbb{R}^{n \times r}$ is the shape basis, and $\mathsf{v} \in \mathbb{R}^{r \times 3}$ is the reduced coordinate. Equation 1 reduces to

$$\begin{bmatrix} \tilde{\mathbf{L}} & \tilde{\mathbf{S}}^T & \tilde{\mathbf{C}}^T \\ \tilde{\mathbf{S}} & & \\ \tilde{\mathbf{C}} & & \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \lambda_S \\ \lambda_C \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{b}} \\ \mathbf{s} \\ \mathbf{c} \end{bmatrix}, \text{ where} \tag{3}$$

$$\tilde{\mathbf{L}} = \begin{bmatrix} \tilde{\mathsf{L}} & & \\ & \tilde{\mathsf{L}} & \\ & & \tilde{\mathsf{L}} \end{bmatrix}, \; \tilde{\mathbf{S}} = \begin{bmatrix} \tilde{\mathsf{S}} & & \\ & \tilde{\mathsf{S}} & \\ & & \tilde{\mathsf{S}} \end{bmatrix}, \tag{4}$$

for $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{U}$. Here, $\mathbf{v} = [\mathsf{v}_{[:0]}^T \mathsf{v}_{[:1]}^T \mathsf{v}_{[:2]}^T]^T$, and similarly for $\tilde{\mathbf{b}}$. We use the symbol $\tilde{x}$ to denote the reduced counterpart of x. The $r \times r$ subspace system matrix

$$\tilde{\mathsf{L}} = \mathsf{U}^T \mathsf{L} \mathsf{U} \tag{5}$$

is much smaller to solve than L, assuming $r \ll n$. The subspace selection matrix is $\tilde{\mathsf{S}} = \mathsf{S}\mathsf{U}$, and $\tilde{\mathsf{b}} = \mathsf{U}^T \mathsf{b}$. Reduced models are known to be difficult for detailed collision handling. We address this by dynamically inserting more degrees of freedom using our hierarchical bases (Section 4).

Accelerating right-hand side computation: The time complexity of computing $\tilde{\mathsf{b}}$ is $O(nr)$. We can approximate $\tilde{\mathsf{b}}$ efficiently by exploiting the fact that the subspace vertex positions are highly correlated. We have evaluated rotation clustering [Jacobson et al. 2012] and cubature-like [An et al. 2008] methods, both properly adapted to our setting (details are in Appendix A). Both methods provide fast right-hand side evaluation and we found them to have similar runtime performance and accuracy.

4 MULTI-RESOLUTION SHAPE DEFORMATION BASIS

In this section, we describe how our hierarchical bases are created to allow a compromise between collision detail and speed (Figure 4).

4.1 Single-Level Basis

A good shape deformation basis should be smooth and shape-aware. The deformation quality and the user experience is greatly improved if the basis achieves both constant and linear precision [Wang et al. 2015], i.e., if

$$\mathsf{U}1_r = 1_n, \tag{6}$$

$$\bar{\mathsf{p}} = \mathsf{U}\bar{\mathsf{v}}, \tag{7}$$

where $1_k \in \mathbb{R}^k$ is a vector of ones, $\bar{\mathsf{p}} \in \mathbb{R}^{n \times 3}$ contains the rest positions of mesh vertices, and $\bar{v}$ contains the rest positions of manipulated vertices. Any affine transformation can be achieved once

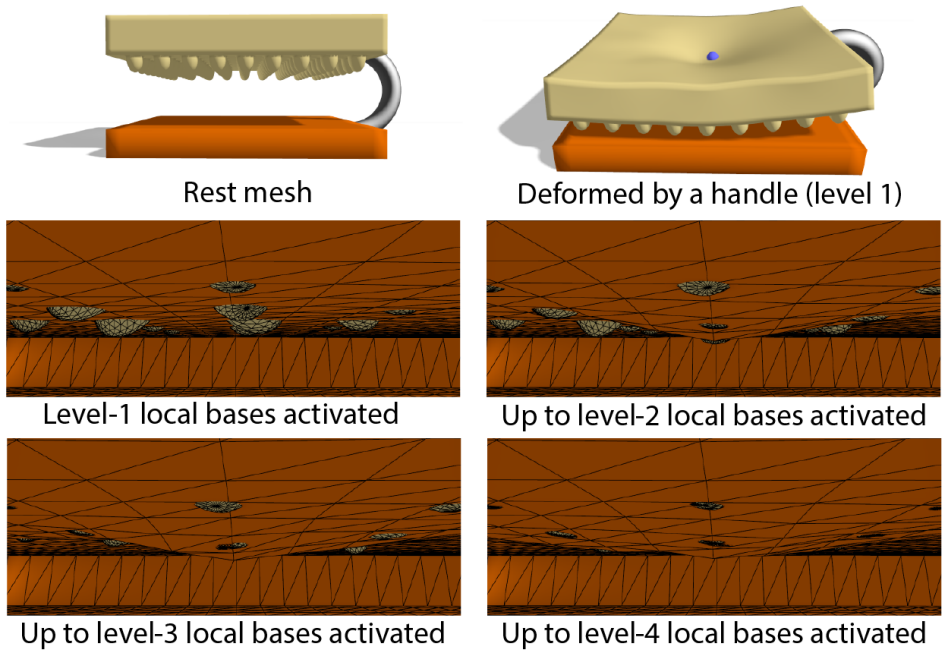

Fig. 4. **Our hierarchical bases provide a tradeoff between collision details and speed.** The top part (mustard color) of the mesh (top-left sub-figure) is pulled to cause a self-contact against the bottom (scarlet color), as shown in the top-right sub-figure. With level-1 local bases activated (top-right and middle-left in closer view), collisions are resolved with visible penetrations and no deformations of the bottom part, at 1× computation time. When level-2 local bases are added (middle-right), our method is able to express the deformation of the bottom part, at the cost of 1.19× of the computation time. When level-3 local bases are added (bottom-left), the penetrations are reduced at the cost of 1.34× time. When level-4 local bases are added (bottom-right), the penetrations are minimal. With many modes activated, our method slows down to 3.10× time.

Equations 6 and 7 are satisfied,

$$\mathrm{p}Q + 1_n t = (\mathrm{U}\mathrm{v})Q + (\mathrm{U}1_r)t = \mathrm{U}(\mathrm{v}Q + 1_r t). \tag{8}$$

Given an affine transformation ($Q \in \mathbb{R}^{3\times3}, t \in \mathbb{R}^{1\times3}$), the transformation of the positions p can therefore be recovered by transforming the control points v. The basis generation method presented by Wang et al. [Wang et al. 2015] satisfies Equations 6 and 7, and we therefore choose it as the first level of our hierarchy.

For completeness, the computation of Wang's basis is described below. Given a solid 3-manifold $\bar{\Omega} \in \mathbb{R}^3$ made of 3D tetrahedra, Wang divides the tet vertices into two disjoint sets: "manipulators" and "free" vertices. For each manipulator vertex, Wang's method computes one basis vector that describes how the free vertices move if one moves the manipulator vertex while keeping the other manipulator vertices fixed. We use the word "manipulator" here to avoid a confusion with the word "handles" which in our paper denotes the user handles to control the shape. Given $r \geq 1$ manipulator vertices with prescribed positions stacked in a matrix $\mathrm{y} \in \mathbb{R}^{r\times3}$, a smoothness energy is minimized,

$$\mathrm{p} = \underset{\mathrm{x}\in\mathbb{R}^{n\times3}}{\operatorname{argmin}} \frac{1}{2}\operatorname{trace}(\mathrm{x}^T\mathrm{A}\mathrm{x}) \quad \text{subject to} \quad \mathrm{Y}\mathrm{x} = \mathrm{y}, \tag{9}$$

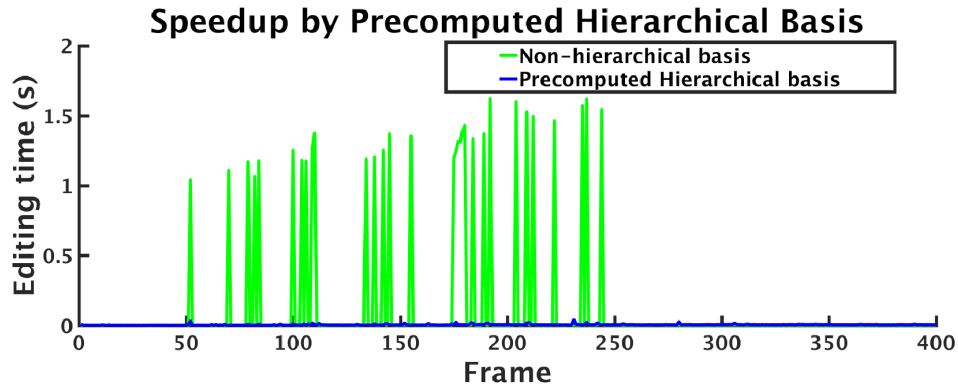

Fig. 5. **Recomputing the Wang's bases during collisions at runtime is extremely slow**, compared to our precomputed hierarchical bases. Octopus model with 20,165 vertices and 92,003 tetrahedra.

where $\mathsf{p} \in \mathbb{R}^{n\times 3}$ are the minimizing deformed mesh positions. The matrix $\mathsf{A} = \mathsf{K}^T\mathsf{M}^{-1}\mathsf{K} \in \mathbb{R}^{n\times n}$ gives a positive semi-definite quadratic form measuring the fairness of x, where $\mathsf{K} \in \mathbb{R}^{n\times n}$ is a linearly precise cotangent Laplacian matrix, and M is the mass matrix. The selection matrix $\mathsf{Y} \in \mathbb{R}^{r\times n}$ selects the r manipulator positions. The solution of Equation 9 is

$$\mathsf{p} = \mathsf{W}\mathsf{y} = \bar{\mathsf{p}} + \mathsf{W}(\mathsf{y} - \bar{\mathsf{y}}), \quad \text{for} \tag{10}$$

$$\mathsf{W} = \mathsf{Y}^T - \hat{\mathsf{Y}}^T(\hat{\mathsf{Y}}\mathsf{A}\hat{\mathsf{Y}}^T)^{-1}\hat{\mathsf{Y}}\mathsf{A}\mathsf{Y}^T, \tag{11}$$

where $\hat{\mathsf{Y}} \in \mathbb{R}^{(n-r)\times n}$ is the selection matrix that selects the free degrees of freedom, and $\bar{\mathsf{y}}$ are the manipulator positions at rest. Matrix $\mathsf{W} \in \mathbb{R}^{n\times r}$ gives the Wang's basis. Each basis vector is smooth and global, and assumes a value of 1 at exactly one manipulator, and 0 at all other manipulators. The largest deformation magnitudes occur close to the manipulator that assumes the value of 1. The basis vectors can be computed using Equation 11, by solving r linear systems with the same sparse matrix $\hat{\mathsf{Y}}\mathsf{A}\hat{\mathsf{Y}}^T$. We factor the system matrix and solve for the columns of W in parallel.

Wang's method allows arbitrary sets of vertices to serve as manipulators. We therefore first tried the following direct and "naive" approach, which we eventually abandoned in favor of the methods presented in the rest of the paper. The idea was to produce localized bases by adjusting the number and distribution of Wang's manipulators at runtime, based on the occurring collisions, and recomputing W. Note that this "localization" is not exact, as all basis vectors are globally supported across the entire mesh, albeit decaying to zero far away from the manipulators. Therefore, thresholding is required to enforce any local support. Furthermore, the recomputation of W is a major drawback because the basis functions need to be recomputed each time a manipulator is added or removed. This is a global solve of size n, regardless of the intended detail and basis content. It is slow and not practical to perform at runtime (Figure 5).

4.2 Hierarchical Basis

Our hierarchical basis addresses these issues by pre-computing local bases and combining them with the global basis at runtime, as visualized in Figure 1 and 6. At runtime, we always activate the global basis. When collisions or other events require more detailed deformations in a region, we activate different levels of local bases of the region, thereby adding more degrees of freedom into the solver. We build our basis hierarchy using Wang's bases, using a hierarchy of manipulators. The structure we adopt is a forest, where a group of globally distributed manipulators serves as the roots of the forest. In this hierarchy, higher manipulators correspond to more global modes, covering deformations on larger areas, while lower manipulators correspond to more local modes,

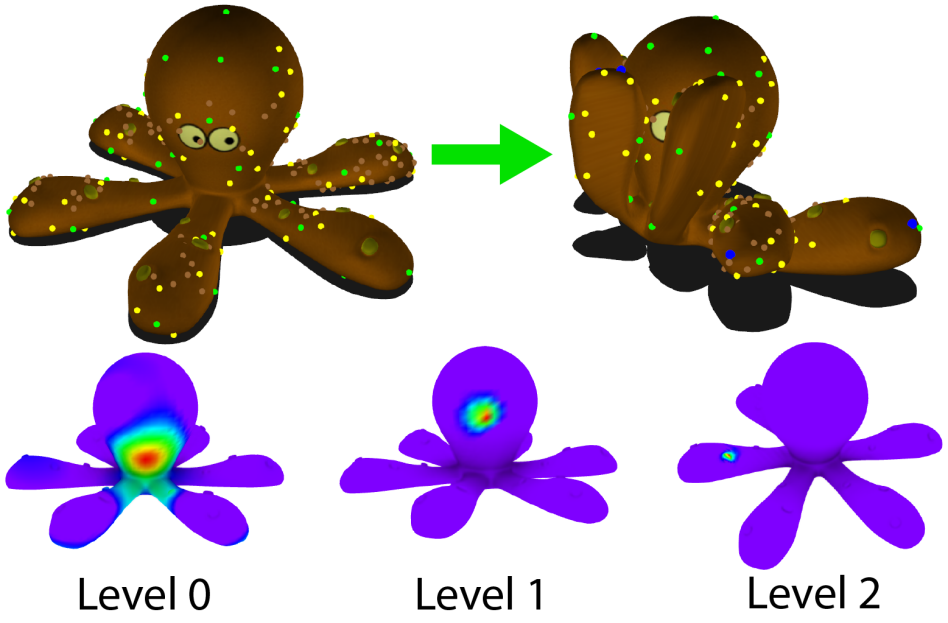

Fig. 6. **Examples of basis vectors at different levels of the hierarchy** (second row). Highest values are colored red and lowest are purple. Basis vectors in higher levels (e.g., global level L0) have larger support while basis vectors in lower levels (e.g., L2) have smaller support. Activated basis vectors due to self-collision are displayed as colored points in the first row. They are green (L0), yellow (L1) and brown (L2). The user handles are colored blue.

covering deformations on smaller areas. Our modes are grouped into global and local bases, and so are the manipulators.

4.2.1 Global Basis. The global manipulator group is created using low-discrepancy sampling. The algorithm starts at a random vertex, and then always picks the vertex that is the furthest from the already selected manipulators and adds it into the selected manipulators, measured in semi-geodesic distances, until the number of selected manipulators reaches a pre-defined value of B_0. To avoid excessive precomputation times on costly geodesic computations, we measure distances by running Dijkstra's algorithm on a graph where each mesh vertex is a node, and two nodes are connected by an edge if the two corresponding vertices share a common tetrahedra; hence "semi-geodesic" distances as opposed to geodesic distances. The length of an edge is the Euclidean distance between the two vertices. In this way, we compute B_0 tree roots for the manipulator forest. In Figures 8 and 9, red points are part of the root manipulators. We refer to this group of manipulators as the global group, or level-0 group. Its corresponding basis is computed using Wang's method on the entire mesh (Equation 11). The basis is called the global basis, or level-0 basis. We divide the mesh into B_0 disjoint sets. These sets are the Voronoi regions of the level-0 manipulators with respect to the semi-geodesic metric described above. Note that the Voronoi regions are also used in meshless deformation models [Faure et al. 2011].

4.2.2 Wang's bases with exact locality. Before we can define the local basis, we need to first give a method to define and compute localized Wang basis vectors, with exact localization and without any thresholding. In addition to free and manipulated vertices, we introduce *fixed* vertices: these are vertices where we want Wang's basis to be identically zero, such as, for example, outside of some

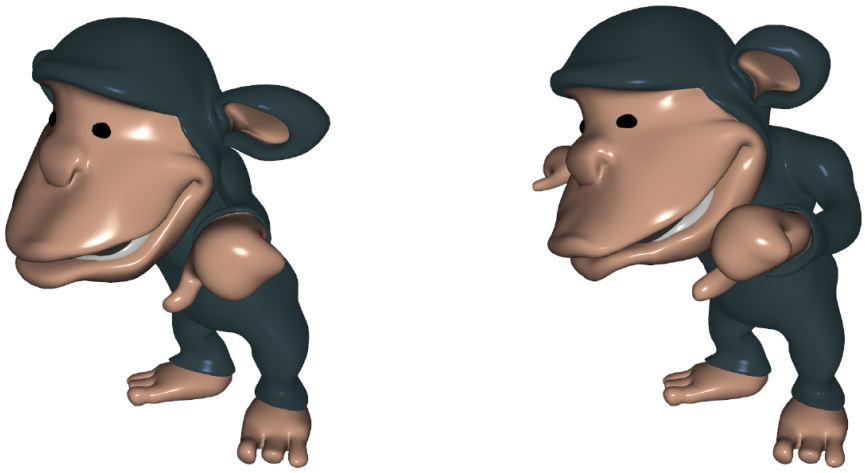

Fig. 7. **Comparison between a different number of global manipulators:** Fewer global manipulators give un-natural motion when deformed only by the global basis. In the left image, the chimpanzee's left arm is not bent properly, compared to the right image where a larger number of global manipulators smoothly deform the arm at the elbow.

localized region in a hierarchy. The construction below works for arbitrary sets of manipulators, free and fixed vertices. Our idea is to make the fixed vertices into "manipulators" whose positions are permanently set to their rest positions. Let Z be the selection matrix that selects the fixed vertices. Then, the constraint in Equation 9 becomes

$$\begin{bmatrix} Y \\ Z \end{bmatrix} x = \begin{bmatrix} y \\ Z\bar{p} \end{bmatrix}. \tag{12}$$

Equation 11 now becomes

$$p = \bar{p} + [W \;\; X] \begin{bmatrix} y - \bar{y} \\ 0 \end{bmatrix} = \bar{p} + W(y - \bar{y}), \quad \text{where} \tag{13}$$

$$[W \;\; X] = \left(I - \hat{Y}^T(\hat{Y}A\hat{Y}^T)^{-1}\hat{Y}A\right)[Y^T \;\; Z^T], \tag{14}$$

and therefore the formula for W is the same as the one given in Equation 11. Matrix W is identically zero on rows corresponding to the fixed vertices, by construction. The submatrix of the manipulator rows is the identity matrix.

4.2.3 Local Basis. We grow the forest by expanding its current leaf nodes, one at a time. This process converts nodes that were previously leaves into interior nodes, and generates new leaf nodes. Suppose node v_i is currently a leaf, located at level i. Recall that given any set of tet mesh vertices $\mathcal{V}$ and a subset of $\mathcal{V}$ called *sites*, the *Voronoi diagram* is a decomposition of $\mathcal{V}$ into disjoint sets whereby all elements of a set share the same site as the closest site (in our pseudo-geodesic metric). Now, let us define a Voronoi diagram whose set $\mathcal{V}$ itself is a Voronoi region determined earlier in the recursion, namely the Voronoi region of the parent node of i. Let the sites be all children of the parent of i (including v_i). The Voronoi region of site v_i is denoted by $\mathcal{V}_i$. We create the children of v_i (level-$(i+1)$ manipulators; the new leaves) by performing low-discrepancy sampling

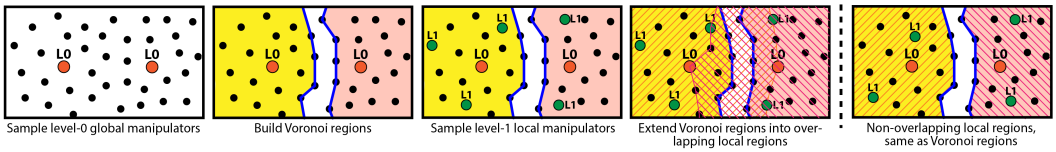

Fig. 8. **Building manipulators and Voronoi regions:** This shows the process of forming local regions to build local bases. From left to right: sample level-0 (L0) manipulators (two in this case; orange); create Voronoi regions (yellow and pink) for each manipulator; sample level-1 manipulators as children of level-0 manipulators (green); extend each Voronoi region into overlapping local regions (cross-hatched). The rightmost image shows the approach of using Voronoi regions directly as local regions. This approach produces artifacts (Figure 9) and we abandoned it.

in $\mathcal{V}_i$. Our low-discrepancy sampling picks the vertex that is the furthest from the already selected manipulators, and vertices outside of $\mathcal{V}_i$. We also skip vertices already selected as manipulators to avoid manipulator redundancy. The result of the sampling are B_{i+1} new manipulators on level-$(i+1)$ that are the children of v_i in the forest hierarchy.

We keep expanding the leaf nodes until all tet mesh vertices are assigned, or a maximum level is reached. Deeper-level nodes have smaller support, and therefore more localized bases. In Figures 8 and 9, green points are manipulators at level-1, and are children of root-level manipulators. A subset of the forest is shown in Figure 9. In practice, our B_0 is larger than B_i, for $i \geq 1$. On level 0, we typically use $30 \leq B_0 \leq 50$ for small meshes, and $100 \leq B_0 \leq 200$ for complex meshes. For other levels, we typically use $4 \leq B_i \leq 10$. A larger level-0 basis produces meaningful global deformation, whereas smaller deeper-level bases give local deformation (Figure 7), without adding too many degrees of freedom to the solver at runtime.

To create a level $i+1$ local basis for a local group which are the children of v_i, our initial approach was to treat the vertices outside of the Voronoi region $\mathcal{V}_i$ of v_i as fixed vertices and build Wang's basis on the children manipulators, as explained in Section 4.2.2. We also include v_i and all its ancestors into the fixed vertex set, so that all manipulator bases on level $i+1$ vanish at the nodes on level i and higher. Because each Wang's basis vector vanishes at the fixed vertices (Equation 11), the local basis has zero values on vertices outside of $\mathcal{V}_i$. We note that *no thresholding is required*; basis vectors are automatically zero, by construction, outside of $\mathcal{V}_i$. However, this basis is problematic because the basis vectors of children of v_i do not overlap with their "cousins", i.e., other bases from the groups at the same level. This means that level-$(i+1)$ bases cannot represent non-zero deformations that cross the boundary of $\mathcal{V}_i$.

We address this issue by extending the Voronoi region into a larger local region that overlaps with other nearby local regions at the same level (Figure 8), and then using the construction from Section 4.2.2 with respect to this enlarged region. For each mesh vertex v in $\mathcal{V}_i$, we compute the largest ball centered at v that does not include any manipulators from levels 0 to $i+1$ located outside of $\mathcal{V}_i$ (Figure 9, top-left). We call the union of these balls the "local region" (denoted in yellow in Figure 9, top-left). We then treat the vertices *outside* of the local region as fixed vertices to compute the local basis for the children of v_i. The effect of this is that our basis functions on the children of v_i extend to the nearby Voronoi regions (Figure 9, top-left). In this way, we achieve the desired overlap while still keeping $\tilde{\mathsf{L}}$ sparse. We note that the basis functions are still automatically zero, by construction, outside of the local region; no thresholding is required.

4.2.4 Assembling Bases. Finally, we describe how our bases functions affect the final position of each tet mesh vertex. We will now briefly assume that only one local basis W_ℓ is activated in addition to the global basis W_g; the hierarchical case can be easily generalized. Our vertex positions

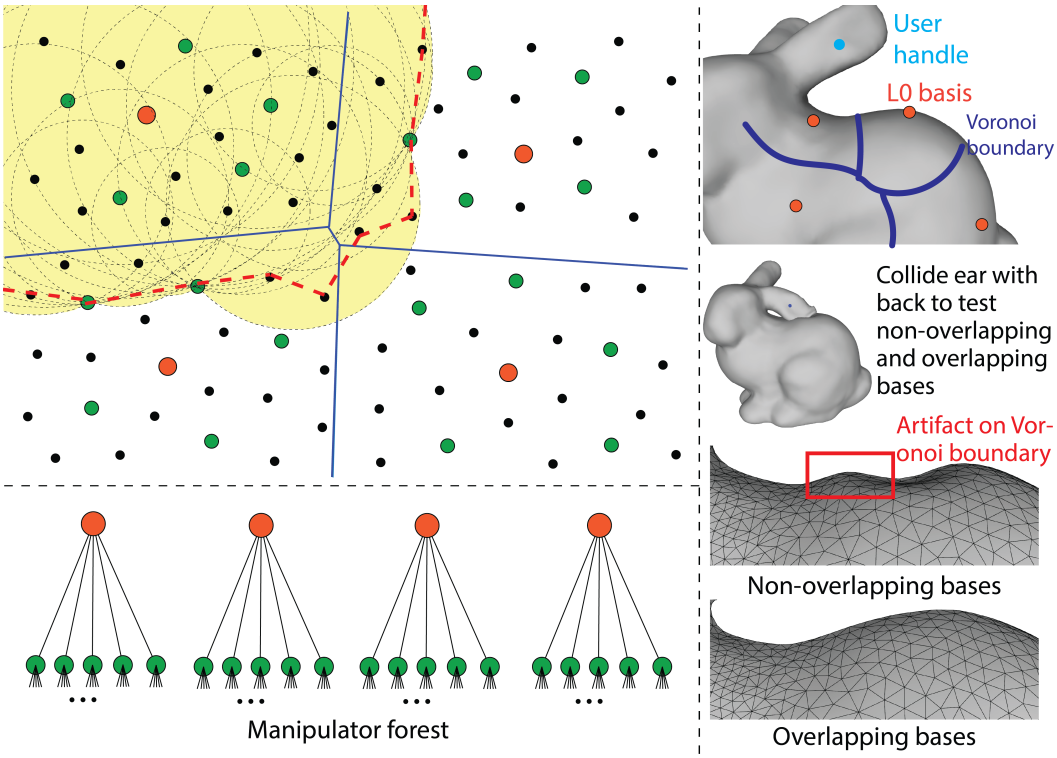

Fig. 9. **Overlapping local region construction details, manipulator forest and artifacts of non-overlapping bases:** Top-left: manipulators on part of the bunny mesh (top-right). There are four level-0 (L0) manipulators (orange) and many L1 ones (green). The local region (yellow) of the top-left L0 manipulator is constructed by unioning balls centered on each vertex in the Voronoi region of the L0 manipulator. Voronoi regions of different L0 manipulators are separated by the blue lines. Dashed orange lines represent the actual local region boundary on mesh vertices. Vertices inside the local region are thus covered by the computed local basis of the L0 manipulator's children manipulator group. Note that this figure only shows a part of the bunny mesh. Bottom-left: the manipulator forest corresponding to the manipulators in the top-left. Right: artifacts occur if the local regions do not overlap. In this case, local regions are just the Voronoi regions. We pulled the bunny's ear to collide with its back. We only activated L1 bases to resolve collisions in order to demonstrate the artifacts of non-overlapping L1 bases. Observe that the non-overlapping L1 bases cannot represent non-zero deformations across Voronoi boundaries, as highlighted in the red rectangle.

are computed as

$$\mathrm{p} = \mathrm{W_g v_g} + \mathrm{W_\ell v_\ell} = \begin{bmatrix} \mathrm{W_g} & \mathrm{W_\ell} \end{bmatrix} \begin{bmatrix} \mathrm{v_g} \\ \mathrm{v_\ell} \end{bmatrix}. \tag{15}$$

We call $[\mathrm{W_g} \quad \mathrm{W_\ell}]$ the "working basis". In Wang's work, v represents both the manipulator positions and the reduced coordinates of the system. However, in our basis hierarchy, $\mathrm{v_\ell}$ should *not* be interpreted as the manipulator positions in the local region, because the global and local bases overlap in each local region. Instead, a crucial insight that makes the entire construction possible is to treat v_ℓ as the *displacements* from positions $\mathrm{W_g v_g}$. This is demonstrated by the following equation. The positions of manipulators $\mathrm{p_\ell}$ in each local region are

$$\mathrm{p_\ell} = \mathrm{S_\ell}(\mathrm{W_g v_g} + \mathrm{W_\ell v_\ell}) = \mathrm{S_\ell W_g v_g} + \mathrm{v_\ell}, \tag{16}$$

where S_ℓ is a selection matrix that selects vertices ℓ from all positions. Note that $S_\ell W_\ell$ is an identity matrix because, by Wang's basis construction, each basis function in W_ℓ has a weight of 1 on its own manipulator, and 0 on other manipulators in each local region.

Linear precision is still preserved in Equation 15. Any 3×3 linear transformation Q can be produced in the subspace by transforming all the control points,

$$pQ = (W_g v_g + W_\ell v_\ell)Q = W_g(v_g Q) + W_\ell(v_\ell Q). \tag{17}$$

Likewise, constant precision holds because any 1×3 translation vector t can be produced by translating *only* the global control points,

$$p + 1_n t = (W_g v_g + 1_n t) + W_\ell v_\ell = W_g(v_g + 1_r t) + W_\ell v_\ell. \tag{18}$$

Note that the above two equations hold because W_g and W_ℓ both satisfy Equations 6 and 7.

We used Wang's basis in our paper due to its good shape modeling properties [Wang et al. 2015]. However, we note that our multi-resolution basis construction could be employed with other shape deformation basis construction methods. For example, one could use Bounded Biharmonic Weights [Jacobson et al. 2011] or Green's functions computed using BEM [James and Pai 2003]. In addition, if the basis weights are linearly precise, so will be our multi-resolution bases.

Speed and memory considerations: Before modeling, we quickly precompute the basis hierarchy, using multi-threading. The local bases are sparse and can be stored efficiently and loaded into memory at startup. At runtime, when more degrees of freedom are needed, global and local bases are simply concatenated into the working basis. In code, this can be done very efficiently simply by assembling pointers to the modes. This provides good flexibility for contact handling. We can, for example, trade collision resolution for speed by setting the maximum permitted activated level at runtime. When changing the working basis, one needs to re-project the system matrix using Equation 5. Since the entire basis hierarchy is pre-computed, we can pre-compute the projection of L into any basis vector in the hierarchy, obtaining an $n \times n$ matrix $\tilde{L}_{all}$. This makes it possible to rapidly project L into *any* working basis, simply by selecting proper rows and columns of $\tilde{L}_{all}$. Note that $\tilde{L}_{all}$ is block-sparse, since the basis vectors in different trees overlap minimally. Alternatively, if memory is scarce and there are not many added basis vectors, we can update $\tilde{L}$ by exploiting

$$\tilde{L} = \begin{bmatrix} U_0{}^T L U_0 & U_0{}^T L U_1 \\ U_1{}^T L U_0 & U_1{}^T L U_1 \end{bmatrix}, \tag{19}$$

where $\tilde{L}$ is re-built using the previous basis U_0 and the newly added basis vectors U_1. We only need to compute $U_1{}^T L U_0$ and $U_1{}^T L U_1$. Since $\tilde{L}$ is symmetric, we only compute its upper-triangular part.

Basis update oracle: At runtime, when new contacts appear, we add local bases into the working basis. Specifically, for a colliding vertex v on level-i, we add the basis vectors of v and its sibling nodes (the local Wang's basis of the parent of v). We then repeat this process for the parent of v, by adding the local Wang's basis of the parent of parent of v. We continue this cascading process until the top of the hierarchy is reached. The user can adjust the maximum permitted level of local bases, trading solver efficiency for collision resolution. For any node ν, we can remove the basis vectors of its supporting region if all the descendant nodes have been contact-free for a given number of frames.

4.3 Uniqueness of solution

Our local basis construction described in Section 4.2.3 ensures that no manipulator is assigned to two or more basis vectors. We prove that any basis selected at runtime is always linearly independent, and therefore the computed solution is unique. To prove linear independence, it is

sufficient to prove that the entire hierarchical basis is linearly independent, as any subset of a linearly independent set is linearly independent.

Lemma: For each node v in the hierarchy, the only basis vectors that do not vanish at v are its basis vector, and the basis vectors of nodes whose hierarchical level is closer to the root than v.
Proof: Observe that the basis functions of siblings of v and descendants of v vanish at v by construction. For the other nodes at equal or larger hierarchical level than v, revisit our construction of overlapping regions in Section 4.2.3 (yellow sphere union in Figure 9): spheres are grown until reaching nodes in sibling Voronoi diagram regions, and therefore basis functions vanish at those nodes. □

Theorem: The entire hierarchical basis is linearly independent.
Proof: Suppose that a linear combination of the basis vectors equals zero at all mesh vertices. For each node on level 0, the only basis vector that is non-zero is that corresponding to the node itself; and hence the linear combination coefficient of all nodes on level 0 must be zero. For a node on level 1, by the Lemma, the only basis vector that is non-zero is that corresponding to the node itself, and the nodes on level 0, but those have zero coefficients. Hence, the coefficient for each node on level 1 must also be zero. By repeating this process, we determine that the coefficients of all basis vectors in the entire hierarchy must be zero, i.e., basis vectors are linearly independent. □

5 COLLISION DETECTION AND HANDLING

Our system supports collisions with external static objects, and self-collisions. Our collision model uses signed distance fields to model external static objects.

5.1 Collision Detection

The collision points can be any set of points sampling the surface of the object. In our results, we use the set of mesh vertices; but one could employ a denser set by sampling, say, more points on the surface triangles. For external collisions, we query the signed distance value of each vertex to check whether it is in contact. For self-collisions, we use spatial hashing [Teschner et al. 2003], where we partition the entire space into uniform grid cells. Each cell stores the deformed mesh vertices it contains, as well as the deformed tetrahedra that overlap with it. We rebuild the hash table at every frame. For each cell, we check the containing points against the containing tetrahedra. If we detect a point x inside a tetrahedron that is not the tetrahedron containing x in the rest configuration, then this is a self-collision that we resolve as follows. We compute the barycentric coordinates of x in the tet, and then "pull-back" x to its position $\hat{x}$ in the rest configuration, using the tetrahedron's inverse affine transformation. We then query the rest shape signed distance field to find a surface point $\hat{x}^s$ that is closest to $\hat{x}$ [Teran et al. 2005]. We then transform $\hat{x}^s$ forward using the affine transformation of, and barycentric coordinates in, its containing tetrahedron, obtaining x^s.

Our collision scheme is easy to implement and is embarrassingly parallel. Unless the hash table degenerates (it did not in our examples), the query time complexity is $O(n)$ when collision points are mesh vertices. Signed distance field queries, and the hash table traversal to read the candidate (point, tetrahedron) pairs can be performed in parallel. Typically, collision detection takes most of the time in our system. To further reduce collision detection costs, we can limit the collision detection to only a subset of the tetrahedra at each frame, in a round-robin manner to ensure all tetrahedra are checked in a fixed number of frames. In this heuristic, the colliding tetrahedra, and nearly colliding tetrahedra (defined by a fixed maximum topological distance to colliding tets) are checked at each frame, regardless of which subsets they belong to.

5.2 Contact Constraint Formulation

We use a constraint-based contact model. Our contacts are modeled as sliding constraints in the plane of contact, as opposed to, say fixing a vertex to the closest surface point. We found that the sliding flexibility is essential for good shape modeling. Given an external collision on point i with position $x_i = S_i p \in \mathbb{R}^3$ and collision normal n_i, we create a constraint

$$n_i^T (x_i - x_i^s) = 0, \tag{20}$$

where x_i^s is the closest point to x_i on the surface of the external object. The selection matrix $S_i \in \mathbb{R}^{1\times n}$ selects the colliding vertex. Each contact contributes a single row to the collision constraint matrix $\mathbf{C}$ and right-hand side $\mathbf{c}$ in Equation 1, as follows

$$\mathbf{C}_{\text{single row}} = \begin{bmatrix} n_{0i}\mathrm{S_i} & n_{1i}\mathrm{S_i} & n_{2i}\mathrm{S_i} \end{bmatrix}, \quad \mathbf{c}_{\text{single row}} = n_i^T x_i^s, \tag{21}$$

where $n_i = [n_{0i}, n_{1i}, n_{2i}]^T$. Given a self-collision on point x_i, we create a constraint

$$n_i^T (x_i - \sum_{j=0}^{2} w_j x_j^t) = 0, \tag{22}$$

where x_j^t are vertex positions of the triangle that embeds x_i^s, and w_j are the barycentric coordinates of x_i^s. For self-collisions, the constraint rows of the system, and the right-hand side, are

$$\mathbf{C}_{\text{single row}} = \begin{bmatrix} n_{0i}\mathrm{S}^t & n_{1i}\mathrm{S}^t & n_{2i}\mathrm{S}^t \end{bmatrix}, \quad \mathbf{c}_{\text{single row}} = 0, \tag{23}$$

where $\mathrm{S}^t = \mathrm{S_i} - \sum_{j=0}^{2} w_j \mathrm{S}_j^t$, and S_j^t selects the triangle's j-th vertex.

After collision detection, we assemble all constraints and form $\mathbf{C}$ and $\mathbf{c}$ to be added to Equation 3. To avoid over-constraining the subspace, we only allow constrained vertices to be the currently active tree hierarchy nodes. This creates at most r constraints (r is current basis size), while the entire system has $3r$ degrees of freedom. To resolve detailed collisions that the current control points cannot represent, we add new local control basis vectors into the system at runtime (Section 4). If the user moves the user handle into contact, we move it out of contact to prevent further collisions.

5.3 Updating the Contact Constraints

We now describe how we model unilateral (i.e., non-sticking) contact. Our approach is inspired by the active set method [Nocedal and Wright 2006], adapted here to shape collision editing, where there is no dynamics, but shape quality is important. Our approach ensures shape smoothness. At the end of each solve, we determine a set of contacts C that *must* persist to the next iteration. We do so by checking the sign of the Lagrange multipliers λ_C of all constraints. A positive Lagrange multiplier means that there is a "sticking force" on the collision surface, in which case the contact is not added to C. All other contacts are added to C. At the next iteration, we perform collision detection again, forming new contacts. Contacts that are in C and that also appear as new contacts, are simply treated as new contacts. Sometimes contact sites drift on the surface after each iteration; this tends to happen in convex regions of the mesh. To encourage smooth sliding over convex surfaces, we modify the contact procedure slightly, by keeping contact constraints on vertices that are no longer in contact but that have a negative Lagrange multiplier. While this adds a small amount of contact stickiness (negligible in our examples), we found it very beneficial for sliding on convex surfaces. After all collision sites are updated, we perform the solve and generate a new set C based on the new Lagrange multipliers.

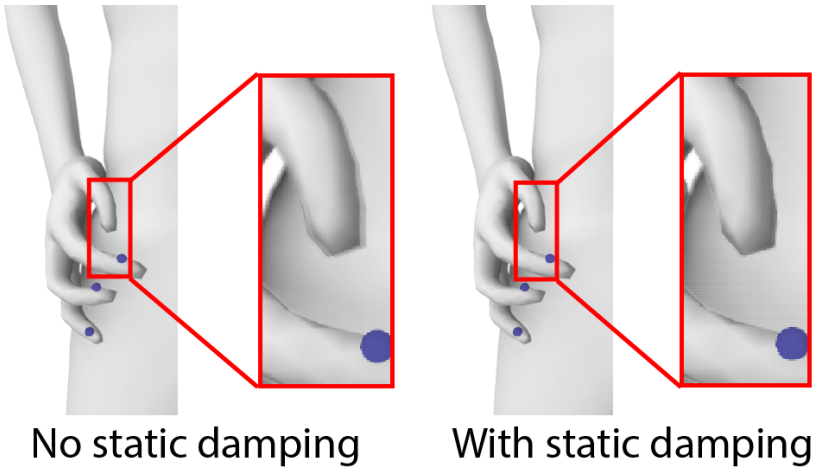

Fig. 10. **Static damping alleviates contact limit cycles:** Here, we edit the shape of the hand of a character. Left: Six frames performed without static damping are superimposed with transparency to show the high-frequency vibration on the thumb, highlighted in a red rectangle. The thumb looks blurred because it is the superposition of several frames of a vibrating motion. Right: Six frames, with static damping enabled ($\alpha = 0.5$). Same (recorded) handle motion as in the Left image. The thumb does not vibrate and the shape converges, therefore the superposition is sharp.

5.4 Static Damping

In complex contact configurations, the solution can be prone to contact limit cycles, where the shape does not converge, but jumps between two or more contact configurations, creating unpleasant visual high-frequency vibrations on the mesh (Figure 10, left). We address this issue using *static damping*. We note that we initially tried to address this issue using a line search; but we found that the line search sometimes makes very slow progress and is not compatible with interactivity. The idea of static damping is that once we obtain the target positions $\mathrm{p_t}$ under contact, we do not attempt to update the mesh to that position, but instead to $(1-\alpha)\mathrm{p_t} + \alpha \mathrm{p_c}$, where $\mathrm{p_c}$ is the current vertex positions. The parameter $0 \leq \alpha \leq 1$ is the *static damping* parameter; for $\alpha = 0$, we obtain the undamped solution and for $\alpha = 1$ we completely discard new shapes, stopping the shape update. The user can set $\alpha = 0.5$ or other values after the manipulation of the handles is over. We can also automatically start static damping each time the user stops moving the user handle, and gradually increases α from 0 to the desired value, letting the contact "settle". Figure 10 demonstrates how static damping can help resolving the cycles.

6 SOLVING THE LINEAR SYSTEM

In Equation 1, the collision constraint $(\mathbf{C}, \mathbf{c})$ changes frequently during the user interaction. If no collision exists, we can pre-factor the system in Equation 3. Unfortunately, pre-factoring does not work well under contact. Each update of $\mathbf{C}$ requires re-factoring. We have considered using an iterative solver such as the preconditioned conjugate gradient (PCG) to avoid factoring. Although PCG only works on positive-definite systems, a filtered version given by Tamstorf et al. [Tamstorf et al. 2015] could work on positive-definite systems with linear constraints. However, for our examples, we found iterative solving to be slower than using a direct method. When lots of local bases are added in U to handle collisions, U becomes sparse and so does L. Therefore we choose to use a direct sparse solver in Eigen [Guennebaud et al. 2010] to solve Equation 1.

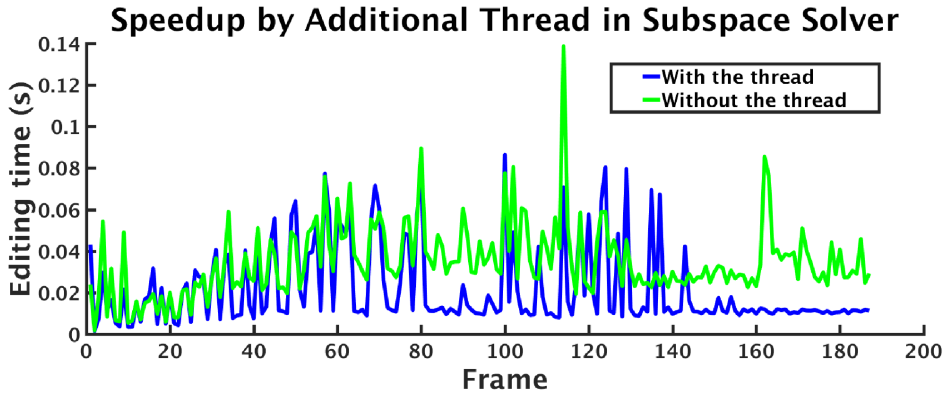

Fig. 11. Comparison between using vs not using the additional thread to compute F_i. Klara cloth example.

In order to maintain the speed of the direct method without the need to re-factor, we adopt incremental solving. Suppose we have factored a symmetric matrix A, then solving

$$\begin{bmatrix} A & B^T \\ B & \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix} \tag{24}$$

can be performed as

$$\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} A^{-1} - FDF^T & FD \\ DF^T & -D \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix}, \tag{25}$$

where $D = S^{-1}, S = BA^{-1}B^T, F = A^{-1}B^T$. Such incremental solving is standard (see, e.g., Barbic et al. [Barbič et al. 2012] and Yeung et al. [Yeung et al. 2016]). The dense solve on S is performed using LAPACK routines in Intel MKL. Given n rows in A and k rows in B, the incremental solve requires $k+1$ pre-factored $n \times n$ solves, and one $k \times k$ dense solve. For a large n, this is faster than to factor the sparse $(n+k) \times (n+k)$ system at every iteration. However, the performance of the incremental solving decreases as more rows are added to B.

We now describe our novel accelerations to speed up incremental solving. Fortunately, we can exploit the fixed structure of A and B. In our system, $A = \tilde{\mathbf{L}}, B = [\tilde{\mathbf{S}}^T \quad \tilde{\mathbf{C}}^T]^T$. Since $\tilde{\mathbf{L}}$ is a block-diagonal matrix, we compute $\tilde{\mathbf{L}}^{-1}\tilde{\mathbf{b}}$ by computing $\tilde{\mathbf{L}}^{-1}\tilde{\mathbf{b}}_{[:i]}$, $i = 0, 1, 2$, and stacking the results. The matrix $\tilde{\mathbf{S}}$ comes from user handle constraints and they are less likely to change frequently. We store $\tilde{\mathbf{F}} = \tilde{\mathbf{L}}^{-1}\tilde{\mathbf{S}}^T$, and reuse it at subsequent iterations. Computing $\tilde{\mathbf{F}}$ can be accelerated by observing that $\tilde{\mathbf{S}}$ is also a block-diagonal matrix, consisting of SU. We only need to solve $\mathbb{L}^{-1}(\mathsf{SU})^T$ once, and stack the results to build $\tilde{\mathbf{F}}$. The matrix $\tilde{\mathbf{C}}$ equals CU, and C has a structure indicated in Equations 21 and 23. For an external collision on vertex i, we compute and store $\mathsf{F}_i = \mathbb{L}^{-1}(\mathsf{S}_i\mathsf{U})^T$. Then, solving for the constraint is simplified: $\tilde{\mathbf{L}}^{-1}\tilde{\mathbf{C}}_i^T = [n_0\mathsf{F}_i^T \quad n_1\mathsf{F}_i^T \quad n_2\mathsf{F}_i^T]^T$. Similarly, for a self-collision between vertex i and triangle t, we compute and store F_i, F_0^t, F_1^t, and F_2^t. Overall, given U, we store the result of F_i so that if there is a contact occurring at vertex i in subsequent iterations, we can accelerate incremental solving. During contact, we use a dedicated thread to compute F_i, while the main thread performs the incremental solve. Performance is analyzed in Figure 11. If the user slides the mesh along the contact surface, new F_i are added and the old entries can be discarded if memory is scarce.

7 RESULTS

We tested our subspace hierarchical method and the full method on several examples. All experiments were conducted on a workstation with Intel Xeon 2.3GHz 2×6 cores and 32 GB memory.

Statistics for our examples are given in Table 1 and 2. For both methods, the slowest frames occur during collisions. Because our application is *interactive* shape modeling, we are primarily interested in the *maximum* slowdown between the full method and our method, as these slow-downs impact interactivity the most. We also report *average* slowdowns. It can be seen that our method is approximately an order of magnitude faster than the full method, both in largest and maximum slowdowns.

Example	vtx	tets	cons	basis size	at:pWv	at:rhs	at:col	at:solve	at:total	at:sp	mt:total	mt:sp
Armadillo	9,746	44,371	5.7	43 / 133	13 %	8.4 %	58 %	6.1 %	3.5 ms	3.9×	30 ms	5.3×
Klara (cloth)	11,718	40,399	34	100 / 885	3.1 %	4.8 %	8.2 %	34 %	23 ms	5.2×	139 ms	27×
Bunny	71,724	215,172	49	50 / 555	12 %	2.6 %	37 %	20 %	19.9 ms	26×	160 ms	96×
Chimpanzee	84,613	334,222	47	50 / 870	4.2 %	16 %	29 %	15 %	88.3 ms	16×	246 ms	109×
Huangshan pine	111,364	348,467	37	200 / 1049	9.2 %	17 %	22 %	10 %	100 ms	2.4×	437 ms	8.5×

Table 1. **Runtime statistics** for #mesh vertices (vtx) and #tetrahedra (tets), #average collision constraints during contact (cons), min/max basis size used during contact (basis size), average computation time (at) broken down in terms of constructing positions $p = Wv$ (at:pWv), constructing right-hand sides (at:rhs), collision detection (at:col), and solving the linear system (at:solve). The average total computation time per iteration (at:total) and the average speedup compared to the full method (at:sp) are also given. We also report the maximum computation time (mt) during the manipulation (mt:total) and its speedup (mt:sp). We use multi-threading with Intel TBB; we parallelize collision detection, rotation extraction, basis formation, and computing mesh vertex positions from reduced coordinates. We also accelerated the full method with the same amount of parallelism, for a fair comparison. We do not employ GPU computation.

Example	hier	clust	bases	proj	total
Armadillo	0.55 s	0.32 s	0.91 s	1.0 s	2.8 s
Klara (cloth)	0.83 s	2.5 s	0.87 s	2.4 s	6.6 s
Bunny	5.7 s	5.1 s	10 s	40 s	61 s
Chimpanzee	8.8 s	—	24 s	55 s	88 s
Huangshan pine	21 s	—	52 s	411 s	484 s

Table 2. **Preprocessing Statistics** for building the hierarchy ("hier"), clustering rotations ("clust"), creating all basis vectors ("bases"), projecting L ("proj"). Column "total" gives the total preprocessing time for each model in Table 1. For the Chimpanzee and Huangshan pine examples, we do not use rotation clustering because many clusters would be needed to accurately model deformations on cylindrical-like shapes such as branches and limbs. Pre-processing such clusters was relatively expensive and only gave marginal runtime speed improvements.

Figure 2 gives a comparison to the method of Harmon et al. [Harmon et al. 2011]. Different from our iterative system, Harmon's method operates as a post-process. We compare it to our method by performing Harmon's collision projection after a collision-free solve in each iteration. The experiment shows that our method outperforms Harmon's method in output quality. This is because we resolve the contact and elastic energy in one solve, producing smooth collision-free shapes. In contrast, Harmon's method ignores smoothness and only seeks the closest collision-free configuration.

In Figure 14, we demonstrate that our method can work with other deformation energies beyond ARAP. We use the volumetric strain energy (Section 5.1 of [Bouaziz et al. 2014]) combined with Projective Dynamics [Bouaziz et al. 2014]. As usual, we compute our hierarchical basis functions, and then project and solve the shape editing problem in this basis.

In the bunny example (Figure 12), we pull one ear of the bunny to collide against the back. The bottom of the bunny is fixed via constraints. Then, the other ear is pulled to collide with the first ear, demonstrating self-collisions. We also did edits on the armadillo with self-collisions (Figure 13).

Fig. 12. **Editing the bunny with self-contacts.** Top row: rest shape. Middle row: edited shape with both ears in contact (left) and the same handle movement but without contact handling (right). Bottom row: back view of the edited shape with (left) and without (right) contact handling.

In the Klara example, we edit cloth in contact with an external object (Klara's body; Figure 15). At the top, the cloth is constrained to Klara's chest. The cloth is composed of 11,718 vertices and 40,399 tetrahedra. Because the subspace is only defined on tetrahedral meshes, we extrude the triangle mesh to create a closed, manifold mesh, and convert it into a tetrahedral mesh using *Tetgen* [Si and TetGen 2006]. We focus on external collisions in this example. The cloth model generates many collisions, which are successfully resolved by our method. The user adjusts the shape of the cloth, similar, say, to silhouette sculpting in film post-production.

The Huangshan pine triangle mesh is embedded into a tetrahedral mesh of 111,364 vertices and 348,467 tetrahedra. In this challenging contact example, the user pulls several branches to collide with other branches (Figure 3). The total number of distinct collision sites appearing during the entire modeling session is ~ 40, and there are between 5 and 10 active contact sites at any time during most of the session. Static damping is added at the end of the session to stabilize the results. Our method is stable and makes it possible to perform the intended modeling at a 8x speedup

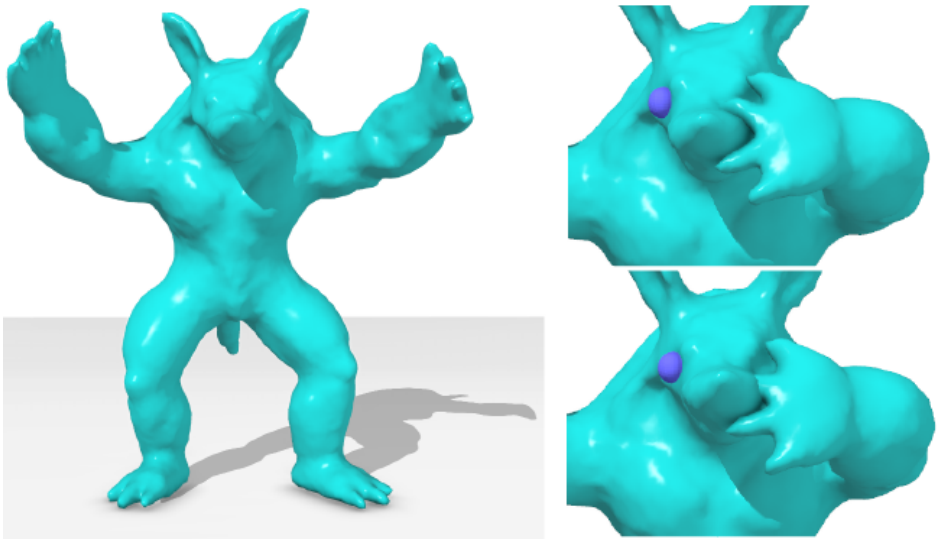

Fig. 13. **Editing the armadillo with self-contacts.** Left: the rest shape. Upper-right: the edited shape with one hand in contact with the head. Lower-right: same handle movement but without contact handling. Note that a finger penetrates into the head.

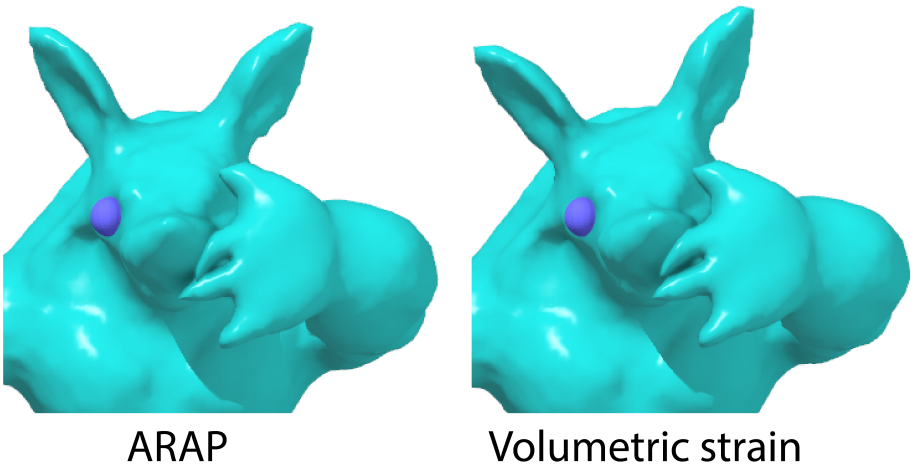

Fig. 14. **Comparison between the ARAP energy and Projective Dynamics.** Although the energies are different, the output visual differences are minor in this example.

in maximum time, compared to full method (Table 1), despite numerous contact sites and many activated basis functions.

8 CONCLUSION

We presented a novel contact model for geometric shapes undergoing collisions and self-collisions. We introduced a multi-resolution hierarchy for shape modeling, and demonstrated how to employ it with complex time-varying contact. We limit maximum computation time by proposing a novel

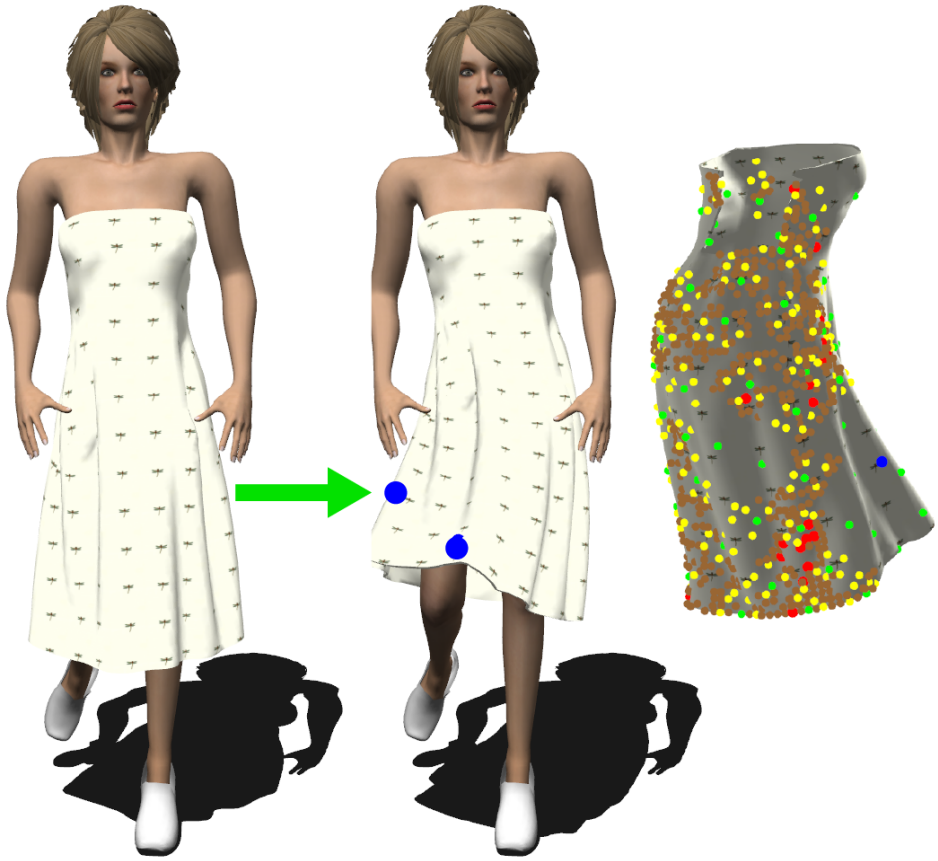

Fig. 15. **Modeling cloth in contact with the body.** The user adds two handles on the cloth and adjusts the cloth shape while our subspace hierarchical algorithm properly respects collisions with the body. Different levels of activated control points are visualized in green (L0), yellow (L1) and brown (L2) in the right figure. The user handles are shown blue and the collided manipulators are red. 11,518 cloth triangles, 32 contacts. Shape-editing runs at 43 FPS on average.

incremental contact caching system suitable for geometric shape modeling with contact. We also stabilized contact using static damping.

We only use vertex handles as user inputs to our system. Use of rigid regions, cages, skeletons and other control methods are left for future work. Our energy operates on a 3D solid tet mesh. In the future, we would like to extend our method to shells and rods. We use standard vertex weights as described in the prior work for the ARAP energy. It would be possible to apply different weights to model heterogeneous objects; the specifics are left for future work.

We use discrete self collision detection on vertex-tet pairs, and therefore we may miss collisions on thin features. Collision detection on edge-edge pairs [Tang et al. 2018b] would improve contact handling, at the cost of additional computation. Continuous collision detection could be added at an additional computational cost. Our system could also benefit from GPU collision detection algorithms [Tang et al. 2018a].

Our multi-resolution basis hierarchy is quickly generated at startup. While the basis vectors can be dynamically added or removed at runtime, we do not support modifying the shape of the basis vectors at runtime. Changing the working basis at runtime can create visual popping; this can

be visually alleviated by temporally blending the output shape. Much like all contact resolution schemes, our performance degrades when large parts of the model undergo contact. We observed that collisions are generally stabler on convex surfaces than on concave ones. Although our static damping improves stability, it adds one extra dimension for parameter tuning and affects the shape globally. An interesting future extension is local static damping for each unstable contact site.

Yeung et al. [Yeung et al. 2016] accelerated incremental solving by launching another thread to factor the existing system matrix, and replace the original matrix with the factored matrix from time to time, to avoid too many rows in B. In our case, factoring the new system matrix means losing the favorable structure of $\tilde{\mathbf{L}}$. While we generally did not encounter significant slowdowns due to a large B, we leave the handling of an arbitrary number of collisions to future work.

ACKNOWLEDGMENTS

This research was sponsored in part by NSF (IIS-1422869), Bosch Research and Adobe Research.

REFERENCES

M. Alexa, A. Angelidis, M.-P. Cani, S. Frisken, K. Singh, S. Schkolne, and D. Zorin. 2006. Interactive Shape Modeling. In *ACM SIGGRAPH 2006 Courses*. 93.

Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G Kry. 2010. Volume contact constraints at arbitrary resolution. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 82.

S. An, T. Kim, and D. L. James. 2008. Optimizing cubature for efficient integration of subspace deformations. In *ACM Trans. on Graphics (TOG)*, Vol. 27. 165.

Jernej Barbič and Doug L James. 2008. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics* 1, 1 (2008), 39–52.

Jernej Barbič, Funshing Sin, and Eitan Grinspun. 2012. Interactive Editing of Deformable Simulations. *ACM Trans. on Graphics* 31, 4 (2012).

Ioana Boier-Martin, Denis Zorin, and Fausto Bernardini. 2005. A survey of subdivision-based tools for surface modeling. *DIMACS Series in Discrete Math and Theoretical CS* 67 (2005), 1.

M. Botsch, M. Pauly, M. H Gross, and L. Kobbelt. 2006. PriMo: coupled prisms for intuitive surface modeling. In *Symp. on Geometry Processing*. 11–20.

M. Botsch and O. Sorkine. 2008. On linear variational surface deformation methods. *IEEE Trans. on Vis. and Computer Graphics* 14, 1 (2008), 213–230.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 154.

C. Brandt, E. Eisemann, and K. Hildebrandt. 2018. Hyper-reduced projective dynamics. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 80.

S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. 2002. A multiresolution framework for dynamic deformations. In *Symp. on Comp. Animation (SCA)*.

I. Chao, U. Pinkall, P. Sanan, and P. Schröder. 2010. A Simple Geometric Model for Elastic Deformations. *ACM Transactions on Graphics* 29, 3 (2010), 38:1–38:6.

Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H Barr. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *Proc. of SIGGRAPH 2001*. 31–36.

Christer Ericson. 2004. *Real-time collision detection*. CRC Press.

Kenny Erleben. 2018. Methodology for Assessing Mesh-Based Contact Point Methods. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 39.

F. Faure, B. Gilles, G. Bousquet, and D. K. Pai. 2011. Sparse meshless models of complex deformable solids. In *ACM Trans. on Graphics (TOG)*, Vol. 30. 73.

Stefan Fröhlich and Mario Botsch. 2011. Example-Driven Deformations Based on Discrete Shells. In *Computer Graphics Forum*, Vol. 30. 2246–2257.

James E. Gain and Neil A. Dodgson. 2001. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics* 7, 4 (2001), 289–298.

E. Grinspun, P. Krysl, and P. Schröder. 2002. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Trans. on Graphics (TOG)* 21, 3 (2002), 281–290.

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

D. Harmon, D. Panozzo, O. Sorkine, and D. Zorin. 2011. Interference-aware geometric modeling. In *ACM Trans. on Graphics (TOG)*, Vol. 30. 137.

David Harmon and Denis Zorin. 2013. Subspace integration with local deformations. *ACM Trans. on Graphics (TOG)* 32, 4 (2013), 107.

T. Igarashi, T. Moscovich, and J. F. Hughes. 2005. As-rigid-as-possible shape manipulation. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM, 1134–1141.

A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine. 2012. Fast automatic skinning transformations. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 77.

A. Jacobson, I. Baran, J. Popović, and O. Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. on Graphics (TOG)* 30, 4 (2011), 78.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 33.

Doug L James and Dinesh K Pai. 2003. Multiresolution green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Trans. on Graphics (TOG)* 22, 1 (2003), 47–82.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. In *ACM Trans. on Graphics*, Vol. 24. 561–566.

Danny M Kaufman, Shinjiro Sueda, Doug L James, and Dinesh K Pai. 2008. Staggered projections for frictional contact in multibody systems. In *ACM Trans. on Graphics (TOG)*, Vol. 27. 164.

Ming C Lin and Miguel Otaduy. 2008. *Haptic rendering: foundations, algorithms, and applications.* CRC Press.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2016. Towards Real-time Simulation of Hyperelastic Materials. *arXiv preprint arXiv:1604.07378* (2016).

Richard Malgat, Benjamin Gilles, David IW Levin, Matthieu Nesme, and François Faure. 2015. Multifarious hierarchies of mechanical models for artist assigned levels-of-detail. In *Symp. on Computer Animation (SCA)*.

Josiah Manson and Scott Schaefer. 2011. Hierarchical deformation of locally rigid meshes. In *Computer Graphics Forum*, Vol. 30. 2387–2396.

Rahul Narain, Matthew Overby, and George E Brown. 2016. ADMM (superset) projective dynamics: fast simulation of general constitutive models.. In *Symposium on Computer Animation*. 21–28.

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

M. Otaduy, D. Germann, S. Redon, and M. Gross. 2007. Adaptive deformations with fast tight bounds. In *Symp. on Computer Animation (SCA)*. 181–190.

Miguel A Otaduy and Ming C Lin. 2006. A modular haptic rendering algorithm for stable and transparent 6-dof manipulation. *IEEE Transactions on Robotics* 22, 4 (2006), 751–762.

Leonardo Sacht, Alec Jacobson, Daniele Panozzo, Christian Schüller, and Olga Sorkine-Hornung. 2013. Consistent Volumetric Discretizations Inside Self-Intersecting Surfaces. In *Computer Graphics Forum*, Vol. 32. 147–156.

Thomas W Sederberg and Scott R Parry. 1986. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 151–160.

Hang Si and A TetGen. 2006. A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany* (2006).

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symp. on Geometry Processing*, Vol. 4. 109–116.

O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H-P Seidel. 2004. Laplacian surface editing. In *Symp. on Geometry processing*. 175–184.

Anthony Talvas, Maud Marchal, Christian Duriez, and Miguel A Otaduy. 2015. Aggregate constraints for virtual manipulation with soft fingers. *IEEE Trans. on Visualization and Computer Graphics* 21, 4 (2015), 452–461.

R. Tamstorf, T. Jones, and S. F. McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. on Graphics (TOG)* 34, 6 (2015), 245.

Min Tang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018a. PSCC: Parallel Self-Collision Culling with Spatial Hashing on GPUs. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 18.

Min Tang, Zhongyuan Liu, Ruofeng Tong, Dinesh Manocha, et al. 2018b. I-cloth: incremental collision handling for GPU-based interactive cloth simulation. In *SIGGRAPH Asia 2018 Technical Papers*. 204.

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace condensation: full space adaptivity for subspace deformations. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 76.

J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. In *Symp. on Comp. Animation (SCA)*. 181–190.

M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross. 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects.. In *VMV*, Vol. 3. 47–54.

N. Umetani, D. M. Kaufman, T. Igarashi, and E. Grinspun. 2011. Sensitive couture for interactive garment modeling and editing. *ACM Trans. on Graphics* 30, 4 (2011), 90.

R. Vaillant, L. Barthe, G. Guennebaud, M.-P. Cani, D. Rohmer, B. Wyvill, O. Gourmel, and M. Paulin. 2013. Implicit skinning: real-time skin deformation with contact modeling. *ACM Trans. on Graphics (TOG)* 32, 4 (2013), 125.

Y. Wang, A. Jacobson, J. Barbič, and L. Kavan. 2015. Linear subspace design for real-time shape deformation. *ACM Trans. on Graphics (TOG)* 34, 4 (2015), 57.

Tim Winkler, Jens Drieseberg, Marc Alexa, and Kai Hormann. 2010. Multi-Scale Geometry Interpolation. In *Computer Graphics Forum*, Vol. 29. 309–318.

J. Ye and J. Zhao. 2012. The intersection contour minimization method for untangling oriented deformable surfaces. In *Symp. on Computer Animation (SCA)*.

Yu-Hong Yeung, Jessica Crouch, and Alex Pothen. 2016. Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices. *ACM Trans. on Graphics (TOG)* 35, 2 (2016), 18.

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. on Graphics (TOG)* 29, 2 (2010), 16.

A FAST RIGHT-HAND SIDE EVALUATION

Let $N(i)$ be the 1-ring neighborhood of vertex i and R_i the "best-fit" rotation on $N(i)$. The ARAP right-hand side can be expressed as

$$b = \sum_i B_i R_i^T, \text{ for } B_i = \sum_{j \in N(i)} \frac{w_{ij}}{2} (S_i - S_j)^T (\bar{p}_i - \bar{p}_j)^T R_i^T. \tag{26}$$

Index i loops over all vertices, and $S_i \in \mathbb{R}^{1\times n}$ selects the i-th vertex. For rotation clustering, we partition the vertices using k-means. The right-hand side is then approximated by

$$b^c = \sum_k B_k^c R_k^T = \sum_k (\sum_{i \in C_k} B_i) R_k^T. \tag{27}$$

Index k loops over each cluster C_k. Rotation R_k is the rotation for cluster k. In the shape subspace, we have

$$\tilde{b}^c = \sum_k \tilde{B}_k^c R_k^T = \sum_k (U^T B_k^c) R_k^T. \tag{28}$$

We can pre-compute $\tilde{B}_k^c \in \mathbb{R}^{r\times 3}$. At runtime, the complexity of evaluating $\tilde{b}^c$ is only $O(r\, m_c)$, where m_c is the number of clusters.

When using cubature, we first create some training shapes. Given a few vertices forming a group $\mathcal{D}$ of size m_d, we wish to find $\tilde{B}_k^d$ so that the right-hand side computed via

$$\tilde{b}^d = \sum_{k \in \mathcal{D}} \tilde{B}_k^d R_k^T \tag{29}$$

is on average the closest to the ground truth values $\tilde{b}$, evaluated at all the training shapes. Since $\tilde{b}^d$ is linear in each element in $\tilde{B}_k^d$, a simple quadratic optimization can be used to minimize the sum of the errors evaluated at all the training shapes. However, we should be aware that this brute force optimization may not preserve linear precision, i.e., the method cannot recover the rest shape $\bar{p}$ when all rotations are identity. The rotation clustering method preserves this property naturally due to the way it is formulated. We resolve this problem by adding a linear constraint to the cubature quadratic optimization, $\tilde{b}(I_3) = \sum_{k\in\mathcal{D}} \tilde{B}_k^d I_3$, where I_3 is the 3×3 identity matrix. Typically, m_c and m_d can be set to a size of r, reducing right-hand side evaluation complexity to $O(r^2)$.