Interference-Aware Geometric Modeling

David Harmon[1] Daniele Panozzo[1,2] Olga Sorkine[1,3] Denis Zorin[1]
[1]New York University [2]University of Genova [3]ETH Zurich

Figure 1: *Interference-aware modeling greatly simplifies many complicated modeling tasks. We interactively fit the ogre with a shirt made for a human. We use our ability to fix existing intersections in a mesh and then "shrink-wrap" the shirt on the ogre, ensuring a perfect fit.*

Abstract

While often a requirement for geometric models, there has been little research in resolving the interaction of deforming surfaces during real-time modeling sessions. To address this important topic, we introduce an interference algorithm specifically designed for the domain of geometric modeling. This algorithm is general, easily working within existing modeling paradigms to maintain their important properties. Our algorithm is fast, and is able to maintain interactive rates on complex deforming meshes of over 75K faces, while robustly removing intersections. Lastly, our method is controllable, allowing fine-tuning to meet the specific needs of the user. This includes support for minimum separation between surfaces and control over the relative rigidity of interacting objects.

Links: DL PDF

1 Introduction

Many applications of geometric modeling require constructed shapes to be physically realizable. Shapes created using computer-aided design systems need to be manufactured; if the model is used in a physical simulation—either for special effects, animation, or engineering analysis—its geometric properties should be consistent with those of a real object. One significant impediment to this consistency is intersections within or between modeled objects. These (self-)intersections appear as glaring artifacts, and eliminate the ability to use the final model further down many software pipelines.

Despite its importance, there has been little research on the modeling of surface interference for geometric design. While a number of recent algorithms for collision *detection* are sufficiently fast for interactive applications, collision *response* in the context of geometric modeling has not received much attention. Once interference has been found through detection, the response algorithm modifies positions and trajectories to remove it.

Most existing contact response algorithms are developed for physically based simulations and try to follow the logic of physical laws. As a result, these methods are either too slow (due to strict physical requirements) or cannot be extended for application to general modeling scenarios. Meanwhile, free-form shape design is primarily concerned with surface quality, interactive control, and aesthetics, and typically is not governed by physical equations. The few works on interactive surface deformation that do handle collisions do so as a side effect of their particular modeling paradigm, and are thus limited to those specific tools to model intersection-free surfaces.

In this paper, we present a response algorithm for preventing interference between and within meshed surfaces, formulated in a purely geometric setting. Objects do not have any physical attributes, and their deformation is not necessarily driven by forces. We formulate non-interference constraints on *space-time interference volumes* (STIVs), defined as volumes in space-time traced out by parts of the surface after interpenetration occurs. The advantages of this formulation are two-fold: first, a trajectory-based method can robustly handle problems with thin features, boundaries, and rapid large deformations, without restrictions on the type of geometry. Second, by formulating the non-interference constraint to be zero for each STIV, rather than at the geometric primitive level, the dimension of the numerical problem to be solved is vastly reduced, improving response speed and robustness. Our proposed method has the following features:

- **Independent of deformation model.** Our algorithm is not tied to a specific modeling paradigm; it can resolve intersec-

tions while keeping the surface in the same subspace (*e.g.*, a subdivision surface remains a subdivision surface with the same controls, or a mesh modified using Laplacian editing still minimizes the same energy). The only input the algorithm requires for a specific deformation technique is the gradient of the function mapping control handles to surface point positions.

- **Handles general geometric data.** Our algorithm is able to robustly handle large deformations, complex intersections, sharp features, as well as self-collisions within a surface. It is capable of processing intersections between surfaces with boundary and large numbers of disconnected components.
- **Controllable.** Different modes of response to collisions suitable for modeling applications fit into our framework. For example, we allow the user to specify whether objects move rigidly and maintain their shape or deform due to interference.
- **Fast.** Interference detection typically dominates runtime costs. We treat detection as a black-box, so our interference algorithm can freely leverage state-of-the-art research in collision detection methods, which currently allow for interactive editing of large models.

We demonstrate the applicability of our method on a variety of scenarios using a range of modeling techniques, including subdivision surfaces, free-form deformation, and Laplacian surface editing. We demonstrate that in many cases, STIVs can also resolve self-intersection in existing meshes, making it easier to use our technique with existing geometry. We are able to achieve interactive rates for a number of realistic geometric modeling scenarios (Section 6 discusses performance in greater detail).

2 Related work

Collision detection. Collision detection is usually treated separately from collision response. This paper focuses on the response algorithm, so we refer the reader to a recent survey [Teschner et al. 2004] and book [Ericson 2004] which take a comprehensive look at collision detection techniques. We note that the literature has long recognized the need for so-called *continuous time* formulations in detecting interference, which we complement by presenting an appropriately paired response algorithm that operates in the same space-time domain. Provot [1997] presented a general method for deformable surfaces, where the roots of cubic polynomials (in time) are found to detect collisions. However, these works focused on the 4-dimensional *detection* problem, without offering a matched *response* model.

Physical simulation. Most of the work on *response* to collisions and intersections is done in the context of physical simulation, and we build on some of the techniques from this domain. However, our problem is different: on the one hand, we do not have stringent requirements on physical accuracy, especially in the context of dynamic effects (we only need natural and intuitive behavior suitable for geometric modeling applications). On the other hand, physically based response (*e.g.*, penalty forces) does not fit well into a pure geometric framework, and the requirements for robustness, generality and efficiency are more restrictive, compared to a typical physical simulation. We can broadly divide physically based response methods into three categories: penalty forces, impulses, and constraints.

Penalty forces are additional forces acting to separate the surfaces, or maintain contact. These forces are well-understood in the contact mechanics community [Wriggers and Laursen 2007] and were introduced to computer graphics by the early work of Terzopoulos *et. al.* [1987]. While penalty forces are easy to implement, difficulties often arise in adjusting the stiffness of these forces—too weak, and objects simply pass through one another, too strong, and the system becomes poorly conditioned. Harmon *et. al.* [2009] address this problem through asynchronous timestepping, albeit at a significant computational cost. For our application, we cannot afford the high runtimes necessary to ensure robustness.

An alternative is to view collision response as an instantaneous reaction (an *impulse*) representing an abrupt change in momentum. While using sequentially applied impulses for response is possible [Moore and Wilhelms 1988; Mirtich and Canny 1995], impulses quickly becomes computationally challenging even with a small number of collisions. Furthermore, impulses are known to not always converge, and we need a method that reliably resolves intersections. Many circumvent these problems either by using time integration schemes that handle such discontinuities [Stewart and Trinkle 1996], or accepting the error in treating all collisions during a single timestep as though they were simultaneous, such as in Bridson *et. al.* [2002]. The latter approach is called a *velocity filter*, as it passes over velocities, correcting motion in offending directions. It has proven popular, not just for its original purpose of self-collisions in cloth simulation, but also in a variety of collision scenarios, *e.g.*, hair simulation [Selle et al. 2008]. Despite the fact that their three-pass algorithm robustly resolves intersections, we could not use it while keeping the shape in the modeling subspace, which is necessary to preserve the features that drew the user to choose a specific modeling algorithm in the first place (see Fig. 7).

Lastly, contacts can be viewed as hard, inviolable constraints within the system. This alleviates many problems of sequential impulses, in particular the "bouncing" back and forth between active collisions. Initial research formulated these constraints at the acceleration level, and focused on rigid bodies [Lötstedt 1984; Baraff 1989]. Baraff [1994] noted that these constraints do not always have a solution, and thus proposed constraining motion at the velocity level. These constraint-maintaining impulses have enjoyed popularity, with progress in custom-designed numerical methods [Stewart and Trinkle 1996] as well as a growing interest in friction [Kaufman et al. 2005], a particularly challenging problem. Unfortunately, they only apply to rigid and quasi-rigid objects, and do not scale well to general highly deformable surfaces. Our approach is most similar in spirit to the constraint-based approach of Allard *et. al.* [2010]; we compare to this work in greater detail in §3.

Haptics are concerned with fast interference detection and response, due to the demands of interactive object manipulation. Barbič and James [2008], inspired by McNeely *et. al.* [1999], use a hierarchy of point-shells to approximate objects and obtain extremely stable collision response. However, their method requires significant pre-processing, and the reduced-order deformation model limits the range of edits that can be applied.

Geometric modeling. Research in interference response for geometric modeling has been quite limited. The work of von Funck *et. al.* [2006] offers a modeling tool that deforms surfaces via integration of a smooth vector field. As a by-product of this smoothness property, the mesh is free of local self-intersections. A similar result is obtained from the method of Swirling Sweepers [Angelidis et al. 2006]. In the context of preventing local self-intersections, this behavior is a result of the deformation method under consideration, limiting its applicability to other models.

Spatial deformations affect all geometry it overlaps, preventing interference as long as it defines a bijective map. Gain and Dodgson [2001] specifically discuss intersections within free-form de-

formation (FFD), and the mathematical requirement to construct a FFD scheme that does not introduce self-intersections. In particular, they prove that injectivity of the FFD mapping is sufficient to guarantee no self-intersections, and offer a modified FFD that performs such injective deformations. This work does not easily extend to other modeling techniques.

Snyder [1995] presents a method for placing objects in a scene while avoiding intersections. It only supports rigid bodies and works by applying pseudo-forces and various backtracking methods to find a realistic configuration. In contrast, we desire for a more general modeling environment not restricted to only those configurations that are realistically physical.

Aldrich *et. al.* [2011] deforms volumes through the use of collisions. The basic response model pushes vertices outside of the interfering object. The method we present could be substituted for this step, offering far more robust collision handling in the context of their volume-preserving model.

3 Space-time interference volumes

Let S denote a collection of meshed surfaces in space, described by a vertex position vector $\mathbf{q} \in \mathbb{R}^{3N}$; $\mathbf{q}_i$ is the i-th 3-dimensional vertex of the mesh. For the purposes of our algorithm, we do not distinguish between different objects—they are all regarded as a single surface (possibly with disconnected components). An arbitrary point $\mathbf{r} \in S$ can be written as a weighted sum of vertices, $\mathbf{r} = \sum_i w_i \mathbf{q}_i$, where $w_i \neq 0$ for the primitive vertices which enclose $\mathbf{r}$.

During editing, the user modifies a low-dimensional *control* mesh with configuration $\mathbf{p} \in \mathbb{R}^{3M}$, where $M \leq N$. From the change in the low dimensional configuration $\mathbf{p}$, we update our high-dimensional configuration $\mathbf{q} = \mathbf{f}(\mathbf{p})$, where $\mathbf{f}$ may be a linear or non-linear function given explicitly or as a solution of an optimization problem. Many methods for surface representation and deformation can be easily cast in this form, including subdivision surfaces [Catmull and Clark 1978], free-form deformation [Sederberg and Parry 1986], various linear surface editing schemes [Botsch and Sorkine 2007], PriMo surface modeling [Botsch et al. 2006], and as-rigid-as-possible surface modeling [Sorkine and Alexa 2007]. The function $\mathbf{f}$ is a concatenation of possibly distinct deformation functions defined on different subsets of geometry.

We organize the deformation of $\mathbf{q}$ into a sequence of edits, $\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, ..., \mathbf{q}^{(n)}$, where $\mathbf{q}^{(0)}$ is the initial mesh configuration. Interpolating the edits linearly, we define a continuous deformation of the shape parametrized by t:

$$\mathbf{q}^{(n)}(t) = \mathbf{q}^{(n-1)} + t\Delta\mathbf{q}^{(n)}, \text{ for } 0 \leq t \leq 1,$$

where $\Delta\mathbf{q}^{(n)} = \mathbf{q}^{(n)} - \mathbf{q}^{(n-1)}$, This also defines a piecewise-linear trajectory for every point on the surface $\mathbf{r}^{(n)}(t) \in S^{(n)}(t)$ with corresponding trajectory vector field $\Delta\mathbf{r}^{(n)} = \mathbf{r}^{(n)} - \mathbf{r}^{(n-1)}$. We use this trajectory to detect and respond to mesh interference.

3.1 Defining interference

Consider the case where a pair of deforming points, $\mathbf{r}$ and $\mathbf{r}'$ coincide, $\mathbf{r}(t_I) = \mathbf{r}'(t_I)$. $0 < t_I \leq 1$ is the moment of intersection along the trajectory. We call the triplet $(\mathbf{r}, \mathbf{r}', t_I)$ an *interference event*. For a given point $\mathbf{r}$ in an interference event, we define $\mathbf{r}' = I(\mathbf{r})$ as the complementary point and $t_I(\mathbf{r})$ as the "time" parameter; the moment along the trajectory where the two points coincide. Note that by requiring $t_I > 0$ we require the surface at the start of an edit, $S(0)$, to be intersection-free.

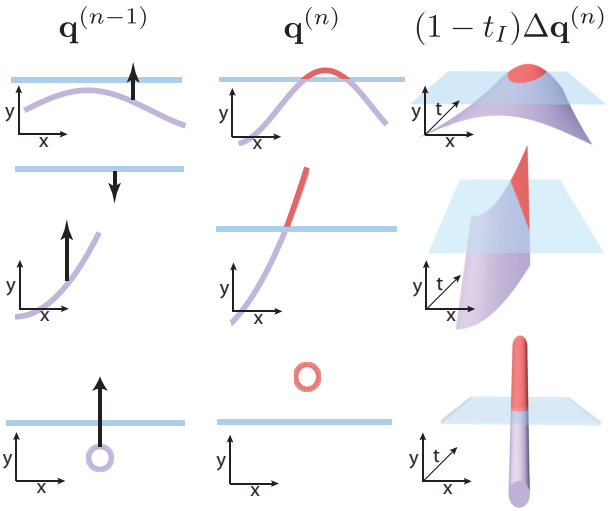


Figure 2: *The variety of interference captured by STIVs. Left: start configuration. Middle: configuration after edit. Right: interpolated configurations, resulting in a space-time surface. Top row shows a typical intersection between two surfaces. STIVs are always well-defined, even for boundaries (middle), and never miss interference, even when completely passing through a thin surface in a single edit (bottom).*

The set of all interfering points forms a subset of the surface which we call the *interference surface*, $S_I \subset S$. For a response algorithm to be considered robust, it must reliably reduce S_I to the empty set by appropriately modifying trajectories.

After an interference event, the trajectory $\mathbf{r}(t)$, along with a small area dS of the surface around $\mathbf{r}$, sweeps out a tube in space. We can use the volume of this tube to measure the severity of interference. Informally, a *space-time interference volume* (STIV) can be thought of as the sum of the volumes of these tubes for all points that passed through another surface at some instance in time.

Observe that the tubes for different areas can overlap, so STIVs do not correspond to volume swept out by the surface in space. Rather, it is a volume of a 3-dimensional subset in space-time $\mathbb{R}^3 \times \mathbb{R}$, consisting of points $(\mathbf{r}(t), t)$ for an interval of values of t (see Figure 2).

More formally, for the geometry deformation from configuration $\mathbf{q}^{(n-1)}$ to $\mathbf{q}^{(n)}$ we define a STIV as

$$V = \int_{\mathbf{r} \in S_I} \left[(1 - t_I(\mathbf{r})) \Delta\mathbf{r}^{(n)} \cdot \hat{\mathbf{n}}(I(\mathbf{r})) \right] dS, \qquad (1)$$

where $\hat{\mathbf{n}}$ is the normalized surface normal of the *other* point $\mathbf{r}' = I(\mathbf{r})$ at the time of intersection, oriented so that each integrand is negative.

We have only modeled the surface interference. Before describing the process by which we eliminate this interference, we emphasize the following important features of this definition:

- The continuous integral over an arbitrary surface has no concern for the underlying object model, shape, or movement of the surface. It is completely general and can describe interference between thin objects, surfaces with sharp features, and surfaces with boundary.
- Considering continuous trajectories ensures that no intersection goes undetected, compared to sampling the geometry at

fixed-interval configurations, such as in Faure *et. al.* [2008], which can miss intersections between relatively fast-moving objects or geometry with thin features.

- The dot product with $\hat{\mathbf{n}}$ takes into account the angle between the linear trajectory of a point, and the normal of the surface it hits at the time of intersection: for near-sliding motions of $\mathbf{r}$, the volume is smaller, while for nearly orthogonal motion it is larger. This ensures only the motion which contributes to interference is penalized and allows surfaces to slide smoothly across one another.

- Our measure of interference is captured by a single volume rather than a separate measure for each interfering point, drastically decreasing the problem dimension. We explore the effect of partitioning measures in §3.3.

This formulation has all the desired properties we set out in §1. Nevertheless, their assurance depends on the exact manner we approximate the integral and resolve interference numerically.

3.2 Resolving interference

As discussed in §2, many choices exist for interference response in the context of physically based simulation. However, as addressed there, none of these apply to geometric modeling without sacrificing speed, generality, controllability, or robustness.

With our STIV construction, we have two general methods for reducing the magnitude to zero. The first is penalty forces formulated through an energy term (generally of the form $\frac{1}{2}kV^2$) that penalizes STIVs, and thus interference, until resolved. Unfortunately, such a force / energy pair has no place in our purely geometric setup where forces have no meaning. Furthermore, penalty forces are well-known to be insufficient for robustly resolving collisions due to the difficulty in tuning the parameter k [Baraff 1989; Harmon et al. 2009].

Our alternative is constraint-based methods. Instead of relying on a force with arbitrary stiffness k to resolve the interference over time, we can constrain V to be exactly 0 and allow the numerical method to find the exact stiffness necessary. Constraints are well-studied in optimization theory [Boyd and Vandenberghe 2004], thus making it simple to develop a purely geometric formulation. STIV constraints can be resolved by a straight-forward application of methods from numerical optimization. For completeness, we present the relevant material.

Constrained optimization. The general form of the constrained optimization problem we are solving is

$$\begin{aligned} &\text{minimize} && E(\mathbf{p}^{(n)}) \\ &\text{subject to} && \mathbf{V}(\mathbf{p}^{(n)}) \geq 0, \end{aligned} \tag{2}$$

where E is an energy measuring proximity to the desired trajectory. Many methods from geometric modeling use an energy to define the deformation. However, as the number of active constraints during any given edit varies, the Lagrange multiplier system for the problem will need to be reconstructed, making it difficult to use any precomputation or pre-factoring for interactivity. Instead, we consider interference response as a post-process, where we directly apply constraints on the configuration of the mesh. The user performs an edit, followed by computation of an unconstrained *candidate* configuration $\tilde{\mathbf{q}}^{(n)} = \mathbf{q}^{(n)} = \mathbf{f}(\mathbf{p}^{(n)})$. We then perform interference detection on the candidate trajectory $\Delta\mathbf{q}^{(n)}$, and apply response to $\mathbf{p}^{(n)}$ to obtain the intersection-free, final configuration $\mathbf{q}^{(n)}$. Figure 3 shows the entire editing workflow. Thus, we define our energy to be $E(\mathbf{p}^{(n)}) = \frac{1}{2}\|\mathbf{f}(\mathbf{p}^{(n)}) - \tilde{\mathbf{q}}^{(n)}\|^2$.

Our interference resolution algorithm is summarized in the pseudocode of Algorithm 1, which gives the solution to Equation 2 subject to a single constraint. The procedure computeSTIV is discussed in §4. Note that the function $\mathbf{f}$ in Line 1 is a black box operation representing the transformation from a geometric subspace $\mathbf{p}^{(n)}$ to the high-dimensional space $\mathbf{q}^{(n)}$. Response is also performed on this modeling subspace $\mathbf{p}^{(n)}$, ensuring that generated surfaces retain the desired formulation (§5.1). The STIV gradients can be found in the appendices.

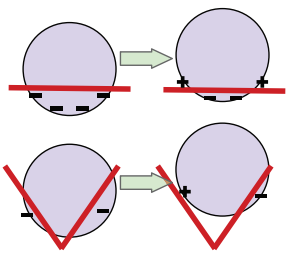
Algorithm 1 Resolving interference through STIV constraints

1: $\Delta\mathbf{q}^{(n)} = \mathbf{f}(\mathbf{p}^{(n)}) - \mathbf{q}^{(n-1)}$
2: **while** $\mathbf{V} = \text{computeSTIV}(\mathbf{q}^{(n-1)}, \Delta\mathbf{q}^{(n)}) \neq 0$ **do**
3: $\quad\lambda = -\mathbf{V}/\nabla\mathbf{V}\nabla\mathbf{V}^T$
4: $\quad\mathbf{p}^{(n)} = \mathbf{p}^{(n)} + \nabla\mathbf{V}^T\lambda$
5: $\quad\Delta\mathbf{q}^{(n)} = \mathbf{f}(\mathbf{p}^{(n)}) - \mathbf{q}^{(n-1)}$
6: **end while**
7: $\mathbf{q}^{(n)} = \mathbf{q}^{(n-1)} + \Delta\mathbf{q}^{(n)}$

This algorithm is iterative, continuing until all intersections are resolved. Due to the linearization of a non-linear 4D volume, the STIV may not be completely removed in a single iteration. However, it is guaranteed to continuously decrease in magnitude. Additionally, by altering the trajectory, new intersections may be introduced that must be detected and resolved (imagine a car applying brakes to prevent a frontal collision and being rear-ended). By requiring that $S(0)$ be free from intersections, a solution (albeit extreme) is guaranteed by eliminating all deformation, in which case $\mathbf{q}^{(n)} = \mathbf{q}^{(n-1)}$.

Comparison to related techniques. Allard *et. al.* [2010] formulated a similar constraint on intersection volumes in three dimensions, rather than in space-time, in the context of elastic object simulation. In this case, they also perform interference detection on static configurations, using their GPU-based technique of Layered Depth Images (LDI) [Heidelberger et al. 2004]. They constrain disjoint intersection volumes, computed by summing intersecting pixel areas in the LDI, to be 0. Unfortunately, LDIs only work for watertight volumes; boundaries violate the algorithm's assumptions. Furthermore, since intersection tests are static, *i.e.*, performed at fixed moments in time, the motion of objects is limited to small deformations. For closed surfaces and small motions, STIVs closely approximate intersection volumes.

3.3 Multiple STIVs



In Equation 1 the contribution of each interference point is negative, resulting in a negative integral. However, we resolve interference by following gradient directions, which may move an individual point into a state where its differential volume is positive. This is why the algorithm is iterative; a "solution" may involve some separated elements (positive integrand) and some intersecting elements (negative integrand), which average out to zero: a numerical solution that still contains interference (see inset figure).

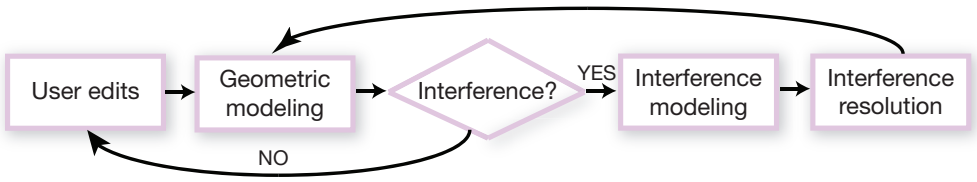


Figure 3: *Our workflow reacts to user edits, checking for interference and responding when necessary.*

Nevertheless, this is not a problem since in the next iteration we include only active interference points, reducing the set over which we integrate. In physical simulation we would form a single constraint per intersection and resolve as a Linear Complementarity Problem (LCP) or a linear projection [Harmon et al. 2008]. Line 3 in Algorithm 1 would be replaced by a call to an LCP solver or a linear solver, respectively. The LCP conditions [Cottle et al. 1993] are

$$0 \leq \lambda \perp \mathbf{V}(\mathbf{p}^{(n)}) + \nabla \mathbf{V} \nabla \mathbf{V}^T \lambda \geq 0.$$

The motivation behind partitioning into multiple constraints is physical. In particular, intersection response involving friction often requires handling such local constraints for physically accurate solutions. We have no such requirements. In fact, the extent to which we need a "physical" solution is only as much as required for intuitive editing of surfaces. As such, we obtain our solution by resolving only a single constraint, with no adverse effects in how the algorithm feels to a user, *e.g.*, no artificial sticking or random "bumps."

One practical concern is that by integrating over a single (potentially large) region we slow down the overall algorithm by requiring additional iterations, each carrying expensive collision detection. For comparison, in §6 we include results with a single STIV as well as multiple STIVs, one per disjoint interference region of the mesh. Further refinement of STIVs into additional constraints would improve physical accuracy of the solution, but in our experience offers no practical advantage for geometric modeling. Between a single constraint and one constraint per region, we find that the difference is negligible, both in feel and in the total number of iterations; this data is presented in Table 1. Intuitively, this can be explained since while the total integral may penalize some intersecting subsets, it may help others by encouraging separation where otherwise exact contact would be enforced.

4 Computing interference volumes

Equation 1 defines a STIV for a continuous surface; our algorithm works on a mesh approximating the surface, which we assume consists of triangular faces. We do not make assumptions about the number of connected components, or manifold property, but we do require that the initial meshes have no self-intersections. At the same time, our technique, in combination with a skeletonization process, can be used to eliminate pre-existing self-intersections (§5.2), although without a guarantee of success.

We follow a natural discretization of Equation 1 where the summation is performed over vertices. Our discrete approximation has the following form:

$$V \approx \sum_{i \in S_I^{(n)}} \left[(1 - t_i) \Delta \mathbf{r}_i^{(n)} \cdot \hat{\mathbf{n}}_i(t_i) \right] \frac{1}{3} \sum_{k \in N(i)} |A_{ik}|. \quad (3)$$

A_{ik}, is the area of the k-th triangle connected to vertex i. Each entry in the summation can be thought of not as a tube, but a prism surrounding each vertex, whose base is the barycentric region around that vertex.

Interference prisms. $S_I^{(n)}$ is the discrete surface subset composed of points $\mathbf{q}_i$ involved in surface interference. We include in $S_I^{(n)}$ all vertices whose barycentric region A_i contains some point $\mathbf{r}$ involved in interference during a deformation. Furthermore, we use the time, trajectory, and normal corresponding to the earliest interference event in the region of $\mathbf{q}_i$ to correspond to t_i, $\mathbf{r}_i$, and $\hat{\mathbf{n}}_i$. Concretely, define the barycentric region corresponding to area A_i as the set of points

$$R_i = \{\mathbf{r}_j = \sum w_k \mathbf{q}_k, \text{ s.t. } w_i \geq w_k, \forall\, 1 \leq k \leq N\}.$$

Note that $A_i = \int_{R_i} d\mathbf{r}$. With this notation in hand, we use the pair

$$(\mathbf{r}_i, t_i), \text{ s.t. } t_i = \min(t_j(\mathbf{r}_j)), \forall\, \mathbf{r}_j \in R_i,$$

for the trajectory and time to represent the i-th region.

4.1 Detecting interference

We have yet to compute each intersecting point $\mathbf{r}$ and its time of intersection. Triangle meshes can come into contact in one of two ways: either a vertex strikes a face, or two edges meet. A vertex intersecting a vertex or an edge is considered a special case of the former. We must find all such events to construct the sets R_i.

Computing the discrete volumes, then, relies on our ability to quickly detect the points along trajectories, t_i, where intersection occurs. These tests are performed on the high-dimensional surface whose shape is defined by $\mathbf{q}_i$. Fortunately, this sort of collision detection is standard, and we can use off-the-shelf algorithms with minimal modifications for our purposes. We have tested our system with the Self-CCD library [UNC 2010] as well as the hash grid approach of Teschner *et. al.* [2003], with complete interchangeability. For low-level tests between vertex-triangle and edge-edge pairs we solve the cubic polynomials presented in Provot [1997]. The roots of these polynomials provide our times of intersection t_i. Along with the point trajectories $\mathbf{r}_i$, we can compute the STIV of Eqn. 3. Where multiple, disjoint STIVs are used, we partition the interference surface based on the 1-ring connectivity of interference points.

The system attempts to exactly resolve each STIV, $V(\mathbf{p}^{(n)}) = 0$. However, due to numerical imprecision, primitives may be left touching or slightly intersecting. To remedy this, we return a value of t_i sooner than the actual value, to account for error in the root finding and response computation. In our examples we return the maximum of 0 and $(t_i - \frac{1}{100})$.

5 Editing

User interfaces implementing interference-aware geometric modeling require little modification from standard modeling software. We describe simple editing primitives and modes that we use in our examples. Most of them are standard, but with their power significantly enhanced by interference awareness.

Primitives. In our examples we use standard translation, rotation and scaling of control points or groups of vertices acting as handles. However, in our system, a simple rigid transform on the controls

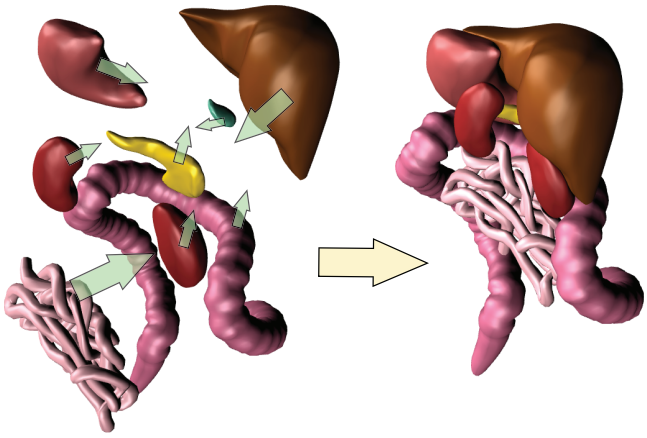


Figure 4: *The modeling task is to pack these bodily organs tightly together. Doing so without consideration for interference is likely to result in intersecting meshes, possibly disqualifying it for use in a medical simulation, for example.*

may result in a complex deformation due to interaction with other objects.

Less standard tools we found essential both for object positioning and removal of self-intersections are *contraction* towards its skeleton (§5.2), and *expansion* back to its original configuration.

Modes. Editing modes define the effect primitive operations have on geometry, and are specific to interference-aware modeling. We have three basic modes: we can enable / disable interference processing, enforce rigid response or allow the mesh to deform, and allow all surfaces to respond or only the selected subset.

We utilize these primitives and modes extensively in our examples, and refer to them in describing our results in §6.

5.1 Modeling subspace

Meshes representing complex shapes have many degrees of freedom and editing vertices directly is often impractical. Most modeling techniques reduces this space in various ways: the geometry is defined by a smaller number of degrees of freedom in a reduced subspace (subdivision surface control points or handles for variational surface editing). Restricting possible configurations of meshes often guarantees many useful properties (*e.g.*, smoothness or detail preservation). We do not wish our treatment of interference to disturb these properties, and thus we perform response in the modeling subspace by modifying the control vertices $\mathbf{p}^{(n)}$ rather than the fine degrees-of-freedom $\mathbf{q}^{(n)}$. Expressions for the necessary gradients are given in the appendices.

5.2 Controlling the behavior of response

Due to the variety of needs in different modeling systems, it is impossible to present a single solution that is "one size fits all." However, we present a variety of options and "add-ons" that increase the utility of interference-aware geometric modeling. Some are purely geometric, other are motivated by physical intuition, but are always formulated in purely geometric terms.

Weighted handles. The gradients ∇V describe how to modify the DOFs in $\mathbf{p}^{(n)}$ to avoid intersection. By appropriately weighing ∇V, we can capture various inertial effects.

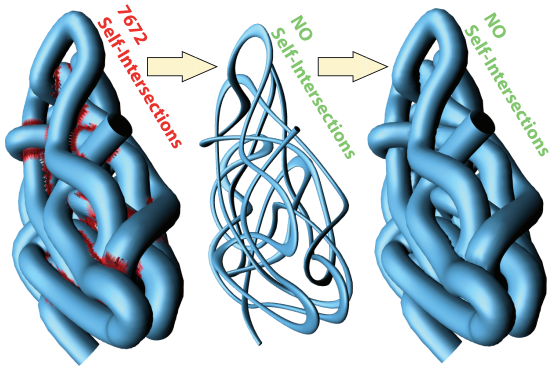


Figure 5: *With a simple extension, our algorithm is able to untangle complicated intersections within meshes.*

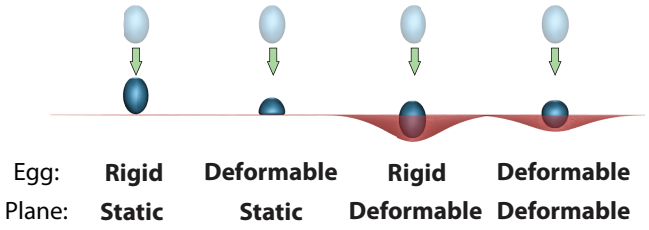
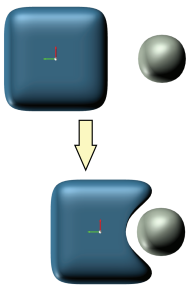


Figure 6: *We weigh the response per handle vertex to achieve many different effects, each useful under different circumstances.*

Let $\mathbf{W}$ be the $3M \times 3M$ identity matrix, and substitute $\mathbf{W}\nabla V^T$ for ∇V^T in Line 3. By modifying the (i,i)-th entry of $\mathbf{W}$, we can control the relative effect of response on the i-th DOF. This is most useful by inserting a 0, which removes the DOF from the system, ensuring that the corresponding vertex remains stationary. This allows us to restrict deformation due to response to only those vertices selected, only those not selected, or allow all to deform.

Minimum separation. In Equation 3, t_i is the parametric value at which an intersection occurs. We can modify the computation of t_i to return the time when two surfaces enter within some *proximity*, rather than exactly touching. This is useful for simulation pipelines, where a minimum distance between all surfaces is needed.



We can express this with the following degree six polynomial:

$$(\mathbf{x}_3(t) - \mathbf{x}_0(t)) \cdot [(\mathbf{x}_1(t) - \mathbf{x}_0(t)) \times (\mathbf{x}_2(t) - \mathbf{x}_0(t))]^2 - h^2 \| [(\mathbf{x}_1(t) - \mathbf{x}_0(t)) \times (\mathbf{x}_2(t) - \mathbf{x}_0(t))] \|^2.$$

Finding the roots of this polynomial gives the points along the trajectory where the vertex $\mathbf{x}_3$ is exactly a distance h apart from the plane spanned by the triangle $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$. By projecting onto the plane, we can confirm that it lies within the region of the triangle. A similar polynomial is constructed from the cubic polynomial for edge-edge intersections [Provot 1997].

We employ a series of optimizations, not unlike those in the cubic case, that vastly reduce the number of polynomials that must actually be solved. Nevertheless, the increased proximity increases the amount of primitives that make it to low-level intersection tests.

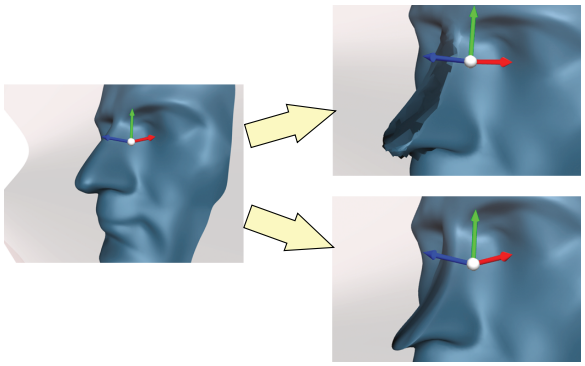


Figure 7: *One response candidate from physical simulation, Bridson* et. al. *[2002] (top), has no consideration for the underlying geometric model, so while it robustly resolves the interference, in the process it ruins the continuity of this subdivision face colliding with a wavy surface. In contrast, our response (bottom) modifies the control mesh, preserving the smoothness of the subdivided surface.*

Self-intersecting meshes. Utilizing Algorithm 1, we are able to resolve self-intersections that *already* exist in many meshes. Our formulation requires $\mathbf{q}^{(n-1)}$ for detection, and is thus *history dependent*. In particular, our algorithm requires a previous configuration that is intersection-free. This is fine for modeling from scratch or simple initial shapes, but poses a challenge for meshes with pre-existing intersections.

Our algorithm for removing existing intersections is based on the observation that if we have *any* non-self intersecting shape $\mathbf{q}^0$, as long as there is one-to-one correspondence to the final shape, we can run our algorithm between $\mathbf{q}^{(0)}$ and a desired configuration $\tilde{\mathbf{q}}^{(1)}$, with intersections repaired automatically.

While the task of constructing such shape in full generality is formidable, a natural candidate in many cases is a shape closer to the skeleton of the mesh. Intermediate steps of the method of Au *et. al.* [2008], based on constrained Laplacian smoothing, yield exactly such meshes. Using this method, we contract the self-intersecting mesh until it is free of intersections. When none are detected, we have found our intersection-free configuration $\mathbf{q}^{(0)}$. We then run an unmodified Algorithm 1 to obtain $\mathbf{q}^{(1)}$, the mesh closest to the original but free of intersections. Clearly, this method is not guaranteed to succeed, but for many models we have tried (*e.g.*, Figure 5), it resolves self-intersections successfully.

6 Results

We tested our algorithm on a variety of scenarios that stress different parts of the system. The results demonstrate the robustness, speed, generality, and controllability of our method, as seen here and in the accompanying video. All modeling sessions were performed single-threaded on a 3.6GHz Intel Core i5.

The following examples range from a few thousand triangles to over 75K, with vertex counts ranging from a few hundred to 38K. Table 1 contains full example data, including various measures for evaluating performance, both for the case of a single STIV constraint and one per disjoint region of the interference surface. The quantitative measures typically used to evaluate performance of contact algorithms for physically based simulation are not entirely appropriate for our task (interactive modeling). For this task, the frame rate profiles (see Figure 12) capture a more practically relevant performance measure. During all operations, we maintain an interactive

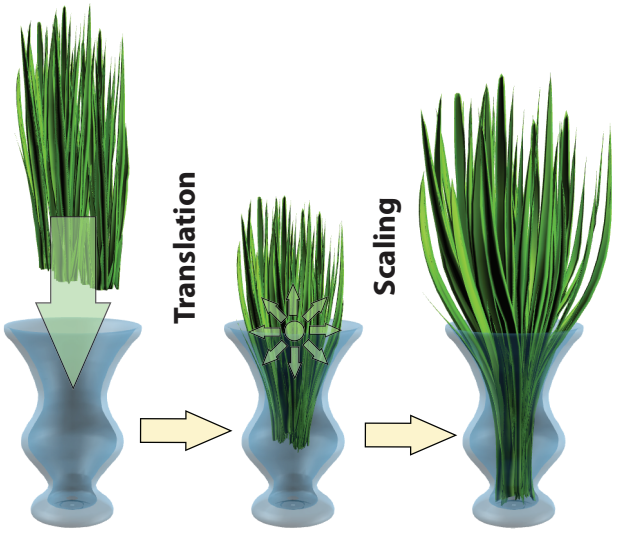


Figure 8: *STIVs do not introduce any artificial friction, so these plant shoots freely slide into the vase, without intersecting it.*

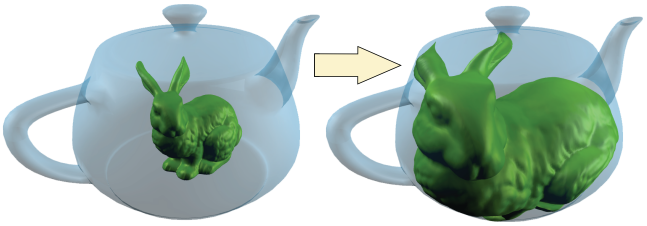


Figure 9: *We enlarge the bunny trapped in the teapot until it is tightly pressed against the sides. Such large regions of consistent intersection are challenging for collision detection.*

frame rate on average, with occasional momentary dips only during the most stressful of operations.

Plant in vase. In this example we wish to place a tight bunch of grass-like shoots into a vase (Figure 8). The vase is immovable, while the shoots are allowed to deform through a simple tri-cubic FFD lattice. We translate the entire plant down into the vase. As it first intersects, the bottom of the lattice pinches inward, allowing the shoots to slide freely into the vase. Once we have reached the end of the vase, we scale the lattice to increase its size, allowing the bunch to freely expand outwards, while the bottom remains constricted inside the vase.

Bunny in teapot. This stress-test was designed to push our system to its limits. The teapot is fixed, while the high-resolution bunny is enclosed in a tri-cubic FFD lattice. We select the entire bunny and scale up, continuing even after intersections occur. Eventually the bunny is tightly pressed against the teapot interior (Figure 9). The entire meshes, both bunny and teapot, are in contact throughout. This stresses timings because there are fewer false positive intersections, and thus fewer options for culling candidate intersections; all low-levels tests must be done. Despite these setbacks, we maintain steady frame-rates and the user receives consistent feedback, even when the frame-rate eventually drops to below 5 frames per second.

Tree in corner. Similar to the bunny, this example grows a tree mesh in a contained region. It also uses a tri-cubic FFD lattice,

Model	Vertices	Triangles	Collisions	Regions	Iterations	Time per iteration (ms)	Total time (ms)
Plant	13759	26782	36.21	4.07	1.56 / 2.20	48.55 / 40.64	75.52 / 89.31
Bunny	38045	75943	46.33	1.42	1.08 / 1.75	630.3 / 524.0	678.7 / 917.0
Tree	32937	18745	16.16	2.49	2.21 / 2.64	56.59 / 56.81	125.0 / 150.2
Knot	5808	11616	23.25	1.83	2.30 / 2.16	65.43 / 57.39	150.0 / 124.0
Ogre	13318	26060	50.14	2.60	8.59 / 11.8	36.84 / 27.52	316.4 / 326.0

Table 1: *For five examples we give, from left to right, the number of mesh vertices, the number of mesh triangles, the average number of collisions between primitives, the average number of disjoint regions, the average number of iterations with multiple constraints (one per region) / one constraint, the average time per iteration (ms) with multiple constraints / one constraint, and the average runtime per edit with multiple constraints / one constraint. Contributions to the average are only taken when the number of collisions is non-zero.*

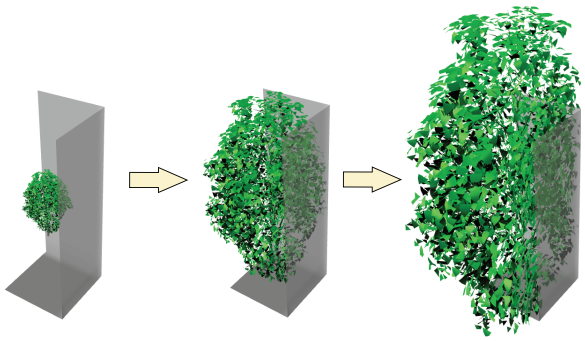


Figure 10: *This tree "grows" in the corner, firmly pressed against the walls without penetration.*

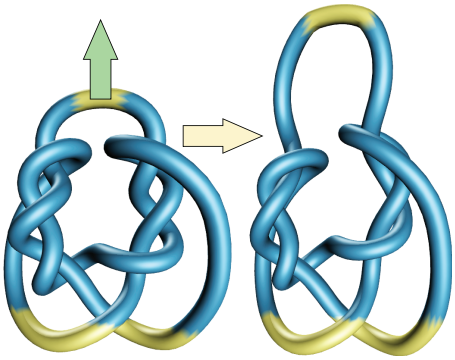


Figure 11: *This tangled knot has no hope of becoming untangled with interference-aware geometric modeling preventing intersections, even for large deformations.*

however the mesh almost entirely consists of "triangle soup". As the tree grows, it pushes against the wall, eventually creeping over the edges.

Knot. This tangled knot is deformed using Laplacian surface editing, with the bottom region fixed, and the top arch manipulated as a handle. It requires little movement to instigate intersections and tighten the knot configuration. By allowing the non-handle mesh to move, the knot becomes further entangled.

Ogre. This examples combines many operations and editing modes into a single, practical example (Figure 1). The task is to dress the ogre using off-the-shelf models. The shirt, for example, is a woman's shirt that clearly will not fit easily, and intersects the ogre in many places. All meshes are Loop subdivision surfaces.

We begin by eliminating intersections with the shirt, by contracting the ogre until intersection-free, then expanding her with interference enabled. This gives an intersection-free shirt, but it clearly was not made for this model. We finish the shirt by scaling the entire mesh down until it intersects the ogre, this "shrink-wrapping" removes the feminine cut of the cloth and gives a more ogre-ish shape. We move the glasses rigidly against the ogre's head until they are properly positioned. We then enter deformable mode and press the glasses on its head, allowing it to freely expand to the proper size. We repeat these operations with the hat to conclude the dressing of the ogre. This is a non-trivial editing task that is made painless, even for non-experts, by interference-aware geometric modeling.

Timings. We give the frames per second values over time for the ogre shirt-fitting and the plant in vase examples. We experience dips in frame-rate during intersection, with more severe intersections causing more significant dips. The important thing to note is these dips are momentary, and quickly recover to a steady state. Overall time is spent in a few operations. An average of 15% (0.1 SD) of runtime is spent processing UI events, 15% (0.21 SD) in geometric modeling, 51% (0.2 SD) is spent in interference detection, and 16% (0.14 SD) in response. Integrating STIVs and computing gradients takes a negligible amount of time. Table 1 gives more detailed timing, with total costs per iteration of our algorithm.

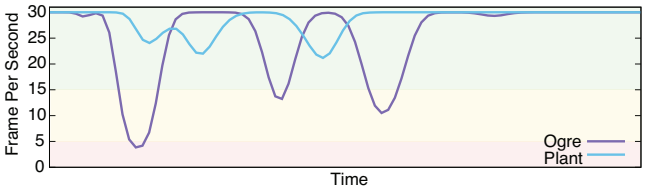


Figure 12: *The frames per second for two examples during an editing session. Dips are momentary, and are little interruption to our algorithm's responsive feedback.*

7 Conclusion

We presented a method for responding to interference in geometric modeling sessions. This method is fast, enabling interactive editing sessions, it is general, not limited by geometry or surface representation, and it is controllable, equipping the user with a wide array of expressive ability. Interference-aware geometric modeling can fit into practically any existing modeling system. It can easily be enabled and disabled, allowing artists to guarantee absence of intersections during critical editing moments, while editing freely during other times.

Implementing specialized methods to handle every case is tedious and increases chance of error. Our aim was to build an algorithm that was general enough, yet had the performance capabilities to work with any scenario encountered in the domain of geometric modeling. Our resulting algorithm follows from physical principles, enough to guide intuitive behavior, but is relaxed enough to edit in real-time.

Limitations. Unfortunately, our method still has strict requirements on the input geometry. In particular, it does not handle degeneracies well. Zero area triangles and zero length edges disrupt the interference detection. Along these same lines, our algorithm also requires input meshes be free of intersection. Using our skeleton-based approach we are able to repair self-intersections in many meshes, but a few are impossible to repair in this way.

Future work. Currently, interference detection is a major bottleneck during processing. This is an active research area, and continued development in continuous detection methods will directly improve our results. There is, in particular, opportunity for utilizing parallel processing to improve interference-aware geometric modeling.

In addition, specialized interference detection algorithms show promise for geometric modeling. There are specialized factors that could motivate the design of new detection algorithms. For example, only subsets of meshes are usually edited at a time, in contrast with simulation, where everything moves each timestep. Furthermore, algorithms could leverage specific knowledge of the modeling scheme for faster processing.

While debugging we observed that the response was fairly insensitive to errors in the STIV computation. This motivates inquiry in fast algorithms for approximating STIVs without performing expensive low-level interference detection.

Acknowledgements

We thank Optitex for providing the ogre shirt mesh. This work was supported in part by the NSF (awards DMS-0602235 and IIS-0905502) and Adobe Research. The first author is supported by a CRA Computing Innovation Fellowship.

References

ALDRICH, G., PINSKIY, D., AND HAMANN, B. 2011. Collision-driven volumetric deformation on the GPU. *Eurographics 2011*.

ALLARD, J., FAURE, F., COURTECUISSE, H., FALIPOU, F., DURIEZ, C., AND KRY, P. G. 2010. Volume contact constraints at arbitrary resolution. *ACM Trans. Graph. 29* (July), 82:1–82:10.

ANGELIDIS, A., CANI, M., WYVILL, G., AND KING, S. 2006. Swirling-sweepers: Constant-volume modeling. *Graphical Models 68*, 4, 324–332.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27* (August), 44:1–44:10.

BARAFF, D. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Proc. SIGGRAPH*, 223–232.

BARAFF, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH*, 23–34.

BARBIČ, J., AND JAMES, D. 2008. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, 39–52.

BOTSCH, M., AND SORKINE, O. 2007. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 213–230.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, 11–20.

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press, New York, NY, USA.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3, 594–603.

CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6, 350–355.

COTTLE, R., PANG, J., AND STONE, R. 1993. *The Linear Complementarity Problem*. Academic Press, New York, NY.

ERICSON, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann, December.

FAURE, F., BARBIER, S., ALLARD, J., AND FALIPOU, F. 2008. Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008*.

GAIN, J., AND DODGSON, N. 2001. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 289–298.

HARMON, D., VOUGA, E., TAMSTORF, R., AND GRINSPUN, E. 2008. Robust treatment of simultaneous collisions. *ACM Trans. Graph. 27*, 3, 23:1–23:4.

HARMON, D., VOUGA, E., SMITH, B., TAMSTORF, R., AND GRINSPUN, E. 2009. Asynchronous contact mechanics. *ACM Trans. Graph. 28*, 87:1–87:12.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3, 145–152.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph. 24*, 946–956.

LÖTSTEDT, P. 1984. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal on Scientific and Statistical Computing 5*, 370–384.

MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. SIGGRAPH '99, 401–408.

MIRTICH, B., AND CANNY, J. 1995. Impulse-based dynamic simulation. In *WAFR: Proceedings of the workshop on Algorithmic foundations of robotics*, A. K. Peters, Ltd., Natick, MA, USA, 407–418.

MOORE, M., AND WILHELMS, J. 1988. Collision detection and response for computer animation. ACM, New York, NY, USA, SIGGRAPH '88, 289–298.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garments. In *Proc. Computer Animation and Simulation*, Springer Verlag, 177–189.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. ACM, New York, NY, USA, SIGGRAPH '86, 151–160.

SELLE, A., LENTINE, M., AND FEDKIW, R. 2008. A mass spring model for hair simulation. *ACM Trans. Graph. 27*, 3, 64–64.

SNYDER, J. M. 1995. An interactive tool for placing curved surfaces without interpenetration. ACM, New York, NY, USA, SIGGRAPH '95, 209–218.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. Symposium on Geometry Processing*, 109–116.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H. 2004. Laplacian surface editing. In *Proc. Symposium on Geometry processing*, 175–184.

STEWART, D., AND TRINKLE, J. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Intl. Journal for Numerical Methods in Engineering 39*, 2673–2691.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. ACM, New York, NY, USA, SIGGRAPH '87, 205–214.

TESCHNER, M., HEIDELBERGER, B., MÜLLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. In *Proc. VMV*, 47–54.

TESCHNER, M., KIMMERLE, S., ZACHMANN, G., HEIDELBERGER, B., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., AND STRASSER, W. 2004. State-of-the-art report: Collision detection for deformable objects. In *Proc. Eurographics*, 119–139.

UNC, 2010. Self-ccd: Continuous collision detection for deforming objects.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.

WRIGGERS, P., AND LAURSEN, T. A. 2007. *Computational contact mechanics*, vol. 498 of *CISM courses and lectures*. Springer.

A STIV gradient

Response is applied by directly modifying $\mathbf{p}^{(n)}$, the handle or control of the mesh. We present the gradient of the inner operation of Equation 3 with respect to a single vertex $\mathbf{q}_j^{(n)}$ and use the chain rule to express gradients with respect to $\mathbf{p}^{(n)}$ (see Appendix B for a few common subspace gradients).

Directly taking the derivative with respect to $\mathbf{q}_j^{(n)}$ gives

$$\frac{\partial V}{\partial \mathbf{q}_j^{(n)}} = \left(\frac{\partial w_j}{\partial \mathbf{q}_j^{(n)}}(1-t_i) + w_i \frac{-\partial t_i}{\partial \mathbf{q}_j^{(n)}} \right) \Delta \mathbf{r}_i^{(n)} \cdot \hat{\mathbf{n}} + w_i \left((1-t_i)\delta_{ij}\mathbf{I}_3\hat{\mathbf{n}} + (1-t_i)\Delta \mathbf{r}_i^{(n)} \frac{\partial \hat{\mathbf{n}}}{\mathbf{q}_j^{(n)}} \right),$$

with $\partial t_i / \partial \mathbf{q}_j^{(n)} = w_j \mathbf{n}$. Note that $\mathbf{n}$ is the un-normalized cross product of the two edges comprising the triangle.

$$\frac{\partial \hat{\mathbf{n}}_i}{\partial \mathbf{q}_j^{(n)}} = \sum_{k=0,1,2} \left(\frac{\mathbf{e}_k(t_i) \times \hat{\mathbf{n}}}{\|\mathbf{n}\|} \otimes \hat{\mathbf{n}} \right) \left(\frac{\partial t_i}{\partial \mathbf{q}_j^{(n)}} \otimes \Delta \mathbf{q}_j^{(n)} \right),$$

where $\mathbf{e}_k$ is the edge opposite vertex k, and $\partial \alpha_i / \partial \mathbf{q}_j^{(n)}$ are derivatives for the barycentric coordinates of a point in a triangle.

B Subspace gradients

Subdivision surfaces. With subdivision surfaces, the control vertices $\mathbf{p}^{(n)}$ are updated, followed by re-computation using the subdivision matrix

$$\mathbf{q}^{(n)} = \mathbf{S}\mathbf{p}^{(n)}.$$

In this case, the linear operator $\mathbf{S}$ is the function $\mathbf{f}$, so $\partial \mathbf{f} / \partial \mathbf{p}^{(n)}$ is simply $\mathbf{S}$. Left multiplying by the original gradients gives the subspace gradients.

Laplacian editing. Laplacian surface editing intrinsically captures surface shape using differential coordinates, $\delta^{(0)} = \mathbf{L}\mathbf{q}^{(0)}$. Then, for a particular edit of the handle $\mathbf{p}^{(n)}$, we solve for the new positions using the formula

$$\tilde{\mathbf{L}}\mathbf{q}^{(n)} = \begin{pmatrix} \delta^{(0)} \\ \mathbf{p}^{(n)} \end{pmatrix},$$

where $\tilde{\mathbf{L}}$ is the augmented Laplacian matrix, as described in Sorkine *et. al.* [2004]. Ignoring numerical instability for a moment, we can re-write this as

$$\mathbf{q}^{(n)} = (\tilde{\mathbf{L}}^T\tilde{\mathbf{L}})^{-1}\tilde{\mathbf{L}}^T\mathbf{S}_L \begin{pmatrix} \delta^{(0)} \\ \mathbf{p}^{(n)} \end{pmatrix}. \quad (4)$$

In this form, the derivative is $(\tilde{\mathbf{L}}^T\tilde{\mathbf{L}})^{-1}\tilde{\mathbf{L}}^T\mathbf{S}_L$, where $\mathbf{S}_L$ is a "selector" matrix containing the identity matrix in the subset corresponding to $\mathbf{p}^{(n)}$, and zeros elsewhere.

Rigid motion. We can write the gradients in terms of the six degrees of freedom representing the center of mass of an object to obtain rigid motion, implementing the *rigid* editing mode. Let $\mathbf{p}^{(n)} = (\mathbf{x}_{cm}\ \theta_{cm})^T$, represent these six degrees of freedom.

Any point i on a rigid body has its location in *body* coordinates specified by $\mathbf{r}_i = \mathbf{q}_i^{(0)} - \mathbf{x}_{\text{cm}}^{(0)}$. and the coordinates at any given point are given by $\mathbf{q}_i^{(n)} = \mathbf{x}_{\text{cm}} + \mathbf{R}_{\text{cm}}\mathbf{r}_i$, where $\mathbf{R}_{\text{cm}}$ is the 3 DOF rigid body rotation coordinates expressed in matrix form.

The partial derivative of a vertex i with respect to the center of mass, is then given by the 3×6 matrix

$$\frac{\partial \mathbf{q}^{(n)}}{\partial \mathbf{p}^{(n)}} = (\mathbf{I}_3 \quad -\mathbf{r}^*),$$

where $\mathbf{I}_3$ is the 3×3 identity matrix, and $-\mathbf{r}^*$ is the skew-symmetric cross product matrix.

Free-form deformations. Free-form deformations express every point of $\mathbf{q}^{(n)}$ as a convex combinations of the points in $\mathbf{p}^{(n)}$, that is:

$$\mathbf{q}^{(n)} = \mathbf{M}\mathbf{p}^{(n)}.$$

The linear operator $\mathbf{M}$ can be build using different basis functions, in our case we used tri-cubic B-splines. The subspace gradients are computed in the same way as for subdivision surfaces.