Computational Design of Reconfigurables

Akash Garg Alec Jacobson Eitan Grinspun

Columbia University

Figure 1: *Reconfigurables—objects that transform between multiple configurations—are found in many walks of life, from mechanisms such as (from left to right) folding bicycles, transforming furniture such as table/bench, architectural advances such as micro-kitchens and -apartments and their specialized cabinetry, to interlocking puzzles.*

Abstract

A reconfigurable is an object or collection of objects whose transformation between various states defines its functionality or aesthetic appeal. For example, consider a mechanical assembly composed of interlocking pieces, a transforming folding bicycle, or a space-saving arrangement of apartment furniture. Unlike traditional computer-aided design of static objects, specialized tools are required to address problems unique to the computational design and revision of objects undergoing rigid transformations. Collisions and interpenetrations as objects transition from one configuration to another prevent the physical realization of a design. We present a software environment intended to support fluid interactive design of reconfigurables, featuring tools that identify, visualize, monitor and resolve infeasible configurations. We demonstrate the versatility of the environment on a number of examples spanning mechanical systems, urban dwelling, and interlocking puzzles, some of which we then realize via additive manufacturing.

Spatial-temporal information about collisions between objects is presented to the designer according to a cascading order of precedence. A designer may quickly determine when, and then where, and then how objects are colliding. This precedence guides the design and implementation of our four-dimensional spacetime bounding volume hierarchy for interactive-rate collision detection. On screen, the designer experiences a suite of interactive visualization and monitoring tools during editing: timeline notifications of new collisions, picture-in-picture windows for tracking collisions and suggestive hints for contact resolution. Contacts too tedious to remove manually can be eliminated automatically via our proposed constrained numerical optimization and swept-volume carving.

CR Category: I.3.0 [Computer Graphics]

Keywords: Computational design

1 Introduction

Using traditional computational tools, the design of a transforming object or a collection of objects transitioning between configurations requires laborious, time-consuming and expensive iterations. This is especially true if relying on physical fabrication merely to determine feasibility. For reconfigurable designs, collisions and interpenetrations pose an editing problem. The solution requires a synergy across space and time lacking in current tools.

We consider the problems unique to the design of objects or collections of objects whose functionality or aesthetic appeal is defined by both the static geometry of participating rigid bodies and their transformations into different configurations, or states. We call such objects *reconfigurables* (see Figure 1).

Modeling in isolation one (spatial) component, or one ("temporal") state, of a reconfigurable opens a Pandora box of tempting modifications that invalidate physical realization. For example, exploring appliance and cabinetry choices of a reconfigurable space-saving

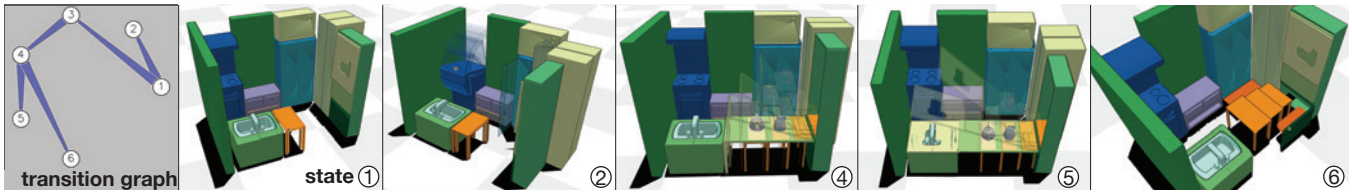


Figure 2: *A reconfigurable kitchen saves space and maximizes utility. Operating on a graph of seven states (left), our interactive environment enables the designer to add deployable, hidden features, such as a hidden countertop above the range (2), cabinet doors that do not interfere (2), a hidden appliance tabletop (4) with over-sink counterspace (5), and a telescoping eat-in table with swing-out benches (6).*

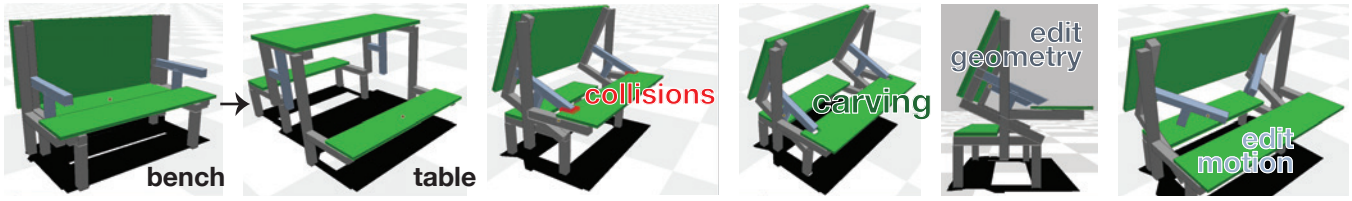


Figure 3: *Alternative solutions for a reconfigurable that starts as a bench and ends as a table. Solution (a) carves the seat to make room for the arm-rests, (b) reshapes the arm-rests to avoid collisions during transition and (c) arm-rests swing inward during the transition.*

kitchen, without carefully considering how they affect mutual clearance, could yield an expensive disappointment (see Figure 2). Likewise, adding armrests to a transforming bench/table might induce interferences to the transition between states (see Figure 3).

We reimagine the typical computer-aided design interface. Rather than optimize for editing static geometries, we treat the transitions of objects between configurations as first class citizens. A designer may interleave edits to an object's geometry in space and edits to its transition through a fictitious graph-based time dimension. All the while, our interactive tools help identify, visualize, monitor and resolve infeasible configurations that may occur along the way.

The first challenge to feasible spacetime editing arises when a seemingly valid edit made in one configuration state creates a new collision in a different state or during a transition. For example, stylizing the handlebars of a bicycle in its unfolded state seems possible, but animating the folding transition reveals new collisions (see Figure 5). Relying on manual scrubbing along the timeline and searching via camera controls is far too tedious to work effectively.

Since collisions might happen at a different time than the designer's current working time selection, the most pressing information is *when* the problem is occurring. Only then is the spatial extent of interpenetrations useful. This order of precedence guides the design of our collision notifications and spacetime collision detection.

The timespans of collision events are identified interactively as the designer makes edits, visualized on each transition's timeline corresponding to edges of a state-transition graph view (see inset). We traverse a four-dimensional bounding volume hierarchy biased toward honing the temporal extent so that notifications immediately appear first on the timeline view. The designer may then jump to this time or preview the collision in a picture-in-picture window, at which point collisions are localized spatially and highlighted in the 3D view.

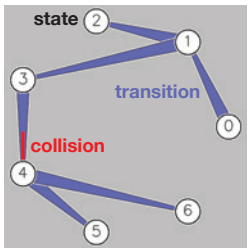


Fig. 4: *Collision alerts appear on the state-transition graph view.*

Some collisions inevitably occur outside the designer's current field of view. The automatic view selection tool transports the designer to an optimal view of the interference. Stroboscopic "ghosts" summarize the reconfigurable's motion near the contact time.

As the designer resolves the collision by editing an object's geometry or configuration, collision notifications interactively update. The designer can track collisions at one moment in time via the picture-in-picture, while making edits at another moment.

Resolving nearly-imperceptible collisions manually can be tedious and ungratifying. If a designer is not sure how to resolve a collision, we provide on-screen hints of edits that would encourage feasibility. Often a designer must make many small or obvious edits only to find that they create new problems elsewhere in spacetime.

Going further than a hint, we can also assist by optimizing over some of the reconfigurable's spatial or temporal degrees of freedom, resolving all slight collisions across spacetime. Our optimization builds on a novel formulation of contact normals, which we derive by extending to 4D a recent contour minimization approach to cloth untangling.

As an alternative to optimizing existing degrees of freedom, the user may also decide that the transformation of one object takes precedence over another object's shape. We propose a novel tool, which combines swept-volume computation with constructive solid geometry to carve away the volume of one object swept along its transformation from the rigid geometry of another.

Through a variety of designs, we demonstrate how the novel

- graph-based time dimension,
- lazy time-then-space collision detection,
- automated picture-in-picture view,
- collision-aware camera optimization,
- collision normal based on spacetime untangling,
- collision resolution hints,
- collision resolution in spacetime, and
- swept volume carving

collaborate harmoniously to afford a fluid interactive design environment. Rapid combinations of our visualization and resolution aids enable fast and creative editing of reconfigurables not possible with traditional tools.

2 Related Work

Though computer-aided design (CAD) and computer animation packages are mature and powerful, each on its own falls short when trying to design a reconfigurable.

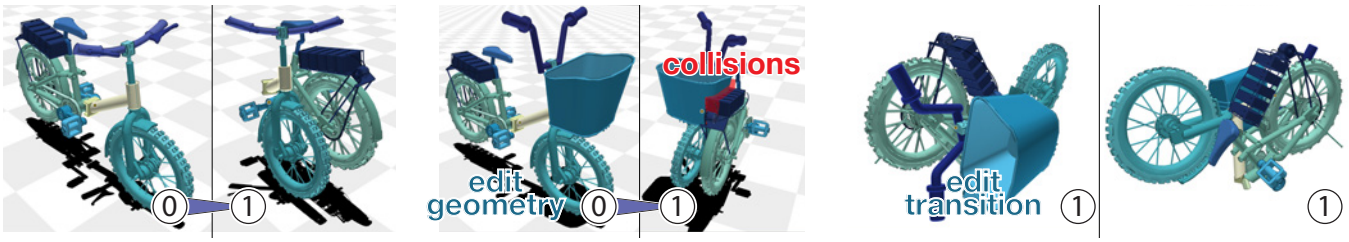


Figure 5: *The original folding bike is collision free, but adding new handlebars and a basket invalidate this type of folding. The designer changes to a swiveling hinge and finds a collision-free transition.*

Optimized for creating intricate static objects, traditional CAD packages offer only simple motion previewing, e.g., a screw twisting into a bolt or a spinning collection of gears. Some commercial tools incorporate interference detection, simply alerting the user if a prescribed motion has created an overlap. For example, a user of SOLIDWORKS can click a button to detect collisions and is alerted via an audible bell sound effect. The detection is not interactive as edits are made, nor does it cluster related collisions. The software does not offer auto-correct or polishing tools. The transition and reconfiguration of an object is not given importance on the same level as the object's surface quality or material strength.

In a complementary manner, computer animation packages *do* place a primary emphasis on the time-synchronized motion of characters and their environments, but time is linear and unidirectional. Our transitions between states are in general a combinatorial graph connecting many states with distinct, potentially bidirectional stretches of time (see Figure 2). Animation, by intent, worries about physical accuracy and feasibility of configurations only in so far as they affect perception. For example, MAYA will detect and respond to collisions during a rigid-/elastic-body simulation, but not during geometry modeling or keyframe animating. Indeed, self-interference of an unfolding bicycle or transformer robot may be perfectly acceptable for a film or video game if unnoticeable.

While we do marry concepts from both CAD and computer animation, our primary investigation asks what *new* tools are needed in the context of computational design of *reconfigurables*.

Static shape design The computational design of *static* shapes also benefits from concepts developed for computer animation or computer graphics. A variety of recent works help users find a physically realizable static output shape that achieves a certain functionality (e.g., [Bharaj et al. 2015]) or aesthetic appeal (e.g., [Igarashi et al. 2012; Garg et al. 2014; Schüller et al. 2014]). The focus of this class of works is on providing an interactive tool for finding a somehow "optimal" yet feasible static shape in a single configuration. The quality metrics and feasibility considerations vary depending on the specific application. Some may employ static physics to tighten the design loop, e.g., for designing spinning, balancing toys [Bächer et al. 2014] or structural-sound furniture [Umetani et al. 2012].

In other scenarios, a shape's quality is determined also by its dynamic behavior, e.g., designing optimal paper airplanes requires consideration of aerodynamics [Umetani et al. 2014]. The interactive garment design tool of Umetani et al. [2011] previews draping dynamics, detecting and resolving collisions as garments contact a mannequin. Unlike our proposal, these tools either ignore inter-object collisions or resolve them in a weak sense à la computer animation: e.g., collisions are important to garment design in so far as they help capture or predict the draping behavior, but collisions are not used explicitly to determine feasibility of a design.

Other works consider a static arrangement of static shapes according to their aesthetic or functional appeal without worrying about the feasibility of possible transitions between arrangements. For example, the furniture in the arrangements of [Merrell et al. 2011; Yu et al. 2011; Liu et al. 2015] are expected to remain in their assigned place, unlike our reconfigurable apartment design in Figure 20.

Fabricating animations While our work is not the first to consider physical constraints during computational design, previous works primarily focus on a very specific class of animated objects. Some methods simply ignore collision during feasibility analysis: e.g., when designing toys [Bächer et al. 2012; Ceylan et al. 2013; Skouras et al. 2013] or mechanical linkage assemblies [Thomaszewski et al. 2014; Bächer et al. 2015]. Similar to the use of collision resolution for static garment design [Umetani et al. 2011], Skouras et al. weakly resolve collisions to design balloons that inflate from a flat configuration [2012; 2013]

Existing works in the realm of reconfigurables consider and take advantage of a heavily constrained space. For example, Li et al. deform shapes to stack vertically on other instances of the same shape [2012], and Zhou et al. decompose and transform arbitrary objects into rectilinear boxes [2014]. Interlocking and twisting puzzles are an interesting subclass of reconfigurables [Xin et al. 2011; Sun and Zheng 2015]. For both types of puzzles, managing collisions during design is important, but also simpler due to the limited design space.

We are inspired by the prototyping tool for hinge-based reconfigurables [Koo et al. 2014]. By working only with simple box-based shapes, Koo et al. can dramatically simplify collision handling and contact relationships. We expand this scope by examining computational design of reconfigurables composed of arbitrary rigid parts.

Interactive motion editing At a technological level, our spacetime collision detection and resolution shares components and motivation with the fundamental path-planning problem in robotics (see e.g., [Kavraki et al. 1996]). However, we are not interested in the fully automatic computation of the path of objects to achieve a certain high-level goal, but rather to provide an interactive tool for editing objects and their transitions between configurations simultaneously.

In computer animation, the spacetime constraints paradigm attempts to reduce animator effort for creating realistic looking animations of a character hitting certain configurations at certain moments in time [Witkin and Kass 1988]. Advanced spacetime methods incorporate contacts and collisions [Popović et al. 2000], but the goal is inevitably to achieve a desirable animation arc. Recently, interactive motion editing also finds interesting applications in the physical world, e.g., designing paths for drone cinematography [Joubert et al. 2015]. While this interface incorporated physical flight constraints, possible collisions are ignored.

During reconfigurable design, our notion of time is artificial. Helping the user find a clear transition path is important, but there is no required concept of momentum, velocity or forces.

Modeling with collisions While we do not intend to model true dynamics, our fictitious time dimension is *not* to be confused with the use of interaction time to exploit collisions for modeling. For example, Harmon et al. essentially treat each user edit during free-form modeling as the next time step of a simulation [2011]. Modeled objects respond to new collisions according to (possibly non-physical) energy-minimizing forces. Similarly, Bernstein and Wojtan exploit the hysteresis between each edit to conduct constructive-solid geometry operations on artifact-ridden geometry [2013]. In contrast, our collision resolution operates directly on a 4D object–the spacetime trajectory—while these earlier works operate on a 3D, spatial object. Their works incorporate (quasi-)time by sequencing spatial collision resolutions. By operating directly on a 4D object, and avoiding the "orientation" of time in a sequential process, our response maintains temporal symmetry. We show that computing a resolution normal for a spacetime trajectory can be achieved exactly, robustly, and simply by integrating over time the contour minimization normals proposed in earlier work for the purely spatial setting.

Rather than liken our work to those that treat modeling as a time-integrated simulation, we interpret our problem as high-dimension analogue of "untangling cloth" [Baraff et al. 2003; Volino and Magnenat-Thalmann 2006]. These works attempt to remove intersections of surfaces in $\mathbb{R}^3$ (presumably resulting from a cloth simulation) *without* knowledge of velocities or previous configurations. One can interpret a standard surface modeling tool, such as MAYA, as a tool suitable for the interactive, albeit manual, untangling of surfaces. Under this philosophical analogy, our work may be viewed as a conceptualization of how modeling packages might be extended to treat spacetime hypersurfaces in $\mathbb{R}^4$. Going beyond manual interactive tools and aids, we also propose automatic spacetime untangling.

To achieve this we construct a four-dimensional *spacetime* bounding volume hierarchy for intersection detection. In contrast, there are many previous works for detecting collisions instantaneously [Allard et al. 2010] or in *continuous time* over short localized spans of time [Tang et al. 2011; Wang et al. 2012].

3 Kinematics and Collision Intervals

The reconfigurable consists of a collection of objects (or parts) that *transition* between various *states*. Abstracted as a graph, states and transitions are nodes and edges, respectively. A visualization of this graph provides a quick overview of the entire reconfigurable (see Figure 4). The graph view quickly indicates to the designer if new collisions appear and allows the designer to monitor collisions within one transition while editing another.

Because we are not interested in dynamic effects such as inertia, we parameterize the motion of an object during a transition along a fictitious time interval $[0, 1]$. Each object is assumed to be a solid bounded by a triangle mesh. In our implementation, objects are limited to rigid motions, i.e., all vertices of one object follow the same translations and rotations during a transition.

Borrowing from early work in collision detection [Cameron 1990], each solid in motion can be viewed instead as a static object in spacetime, occupying a spacetime volume (STV) in $R^3 \times [0, 1]$.

To simplify collision detection, we discretize each vertex path piecewise linearly between evenly sampled time intervals Δt. We may choose Δt sufficiently small to control the tolerance between the true continuous path (orange in inset) and the discretized path (green). The resulting piecewise-linear trajectory serves as the *spacetime proxy* for collision detection.

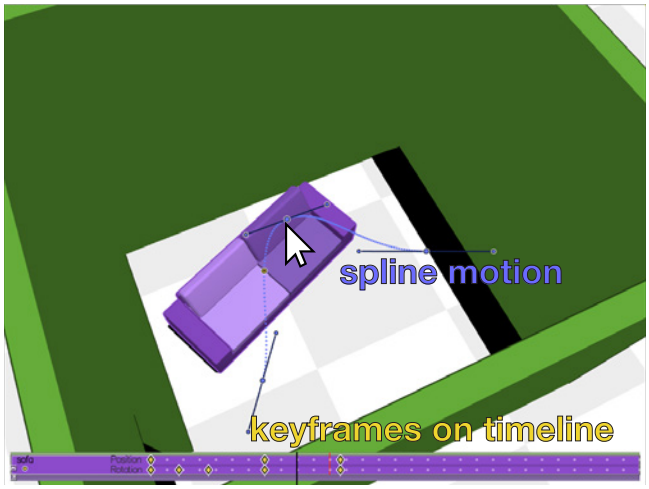


Figure 6: *Using our modeling interface, a designer choreographs the motion of a couch moving inside a reconfigurable apartment (green walls).*

Our collision detector accepts as input spacetime proxies for all objects of a transition. It identifies pairs of triangles that interfere in spacetime. Each such pair is recorded by its spacetime axis aligned bounding box [Cameron 1990], and a reference to the two involved objects. We call such a record a *collision interval* (CI). In general, multiple CIs output from the detector may overlap each other in spacetime. We simplify further by merging overlapping CIs, until the CIs are sufficiently coarse (see §4.1). A merged CI may now reference multiple involved objects.

CIs lay the foundation for all visualization and resolution tools that our design framework provides. The following sections will make use of CIs extensively in order to support the development of tools necessary to visualize collisions in both space and time as well as help the user to resolve those collisions.

4 User Tools

Our investigation is driven by the aspiration to facilitate an interactive, nonlinear, free flowing design process. As the designer alters an object's geometry, state configuration or transition motion, the design environment provides interactive feedback about possible invalid states or transitions. The design environment also guides the designer to resolve these errors by an array of tools that range from informative and unintrusive, yielding control of alterations to the designer, to automatic intervention, alleviating tedium. Our research goal is to understand both the interaction metaphors as well as the under-the-hood infrastructure best suited for interactive, assistive design. Discarding potential tools is just as important as retaining the most effective ones. We present the set front-end tools that we have found to be most effective and generalizable to a broad class of reconfigurable design problems.

As depicted in Figure 6 and the accompanying movie, our design tool builds on keyframe animation to sculpt motions (translations and rotations) of objects, using a timeline and 3D viewer metaphor. This keyframe timeline corresponds to a single edge/transition in the graph view seen in Figure 4. Object transitions within the timeline are represented by keyframed translations along a cubic Beziér spline and spherically interpolated keyframed rotations.

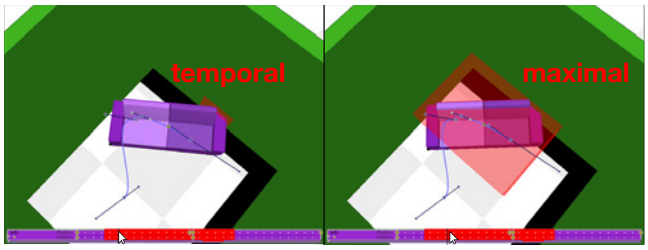


Figure 7: *Left: consolidation of spatial extents over small stretches in time give us more context over collisions compared to, Right: maximal consolidation in space. Both: The timeline always enjoys maximal consolidation of CIs.*

A typical design session begins with an arbitrary arrangement of objects. Additional objects can be added and removed from the scene dynamically. In our research implementation, we externalize those static-shape modeling tasks that are well established and easily accessible in commercial tools, e.g., MAYA's modeler. In the following, we describe the tools that make up the design environment roughly ordered from least to most interventionistic.

4.1 Visualizing Collision Intervals

Complicated reconfigurables are difficult to navigate virtually. As objects transition between states, their relative spatial arrangements are functions of time. Therefore detecting overlaps and collisions between objects *manually*, i.e., via visual inspection, is a tedious process of view manipulations and timeline scrubbing.

We automate the process of identifying and isolating collision intervals (CIs). This process runs continuously in a background thread while the designer makes edits (§5). To direct attention to problem areas, the design environment's 3D viewer displays optionally-pulsating transparent boxes over the spatial extent of each CI (see Figures 7, 9) and the timeline highlights the corresponding temporal extent (see Figures 7, 8). The 3D viewer highlights only those CIs that overlap the selected moment in time, corresponding to the time marker on the timeline.

If a CI is not easy to interpret from the current viewpoint, a hotkey provides invocation to view optimization (§4.3) for rapid optimal focus on a CI of interest. The corresponding spatial and temporal highlights, combined with view optimization, enable the designer to quickly navigate, reason, and act on problems that arise during design revision.

We found that displaying the raw CIs produced by the collision detection routine produces a picture that is in general too busy for a designer to parse. We therefore consolidate CIs that overlap in spacetime: CIs with overlapping or adjacent temporal extents, and spatial extents, merge into one (conservative) CI.

Although considerable consolidation is useful for visualizing temporal extents on the timeline, the same degree of consolidation produces spatial extents that are too conservative (large) to provide any useful context over colliding objects. Thus we consolidate to different extents for the timeline and 3D viewer. While the former enjoys maximal consolidation, the latter is limited to consolidations that keep the temporal extent no longer than a small stretch of time. This ensures that all visualized pulsating highlights encompass colliding regions that occur at the frequency of a single visual frame (see Figure 7).

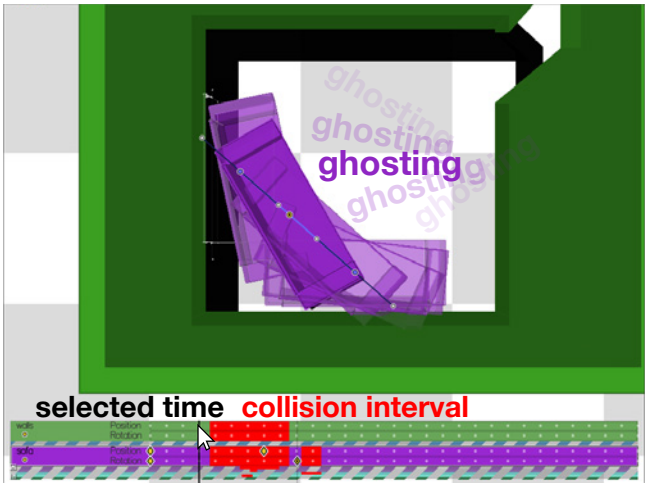
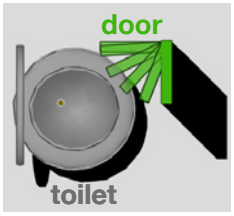


Figure 8: *The designer makes use of "ghosting" to relocate the couch around a tight corner.*

4.2 Ghosting

Inspired by Snibbe [1995], we optionally display "ghosts" of any transitioning object. These allow the designer to see the object's transition without actively scrubbing the timeline. Formally, ghosting is computed by projecting isotemporal slices of object's spacetime volume onto the current 3D view at the selected time. We render the object at regular samples over time, with progressively decreased opacity away from the selected time [Everitt 2001].

Ghosting is particularly useful for laying out objects in tight-fitting spaces, allowing the designer to conservatively reason about potential collision states without repeatedly scrubbing the timeline (see Figure 8). In the inset figure, the designer reasons about the layout and clearance of a swinging door and toilet obstacle.



4.3 View Selection

Visual occlusions due to perspective projection may hide collisions in the scene. Other collisions may simply not be in the current field of view. To help the user inspect problem areas, we propose a method for selecting optimal camera views of collisions. Our interactive design loop requires a fast optimization that considers the dynamic objects. Though analogous in spirit to previous works on optimal view selection [Secord et al. 2011], we have neither the luxury of precomputation nor the convenience of a single, static shape.

We assume a rigid camera with three translational degrees freedom and three rotational degrees of freedom. Assuming a "fixed up axis" (e.g., as in AUTOCAD, MAYA, etc.), we may omit the rotational degree of freedom *twisting* about the viewing axis.

Given a collision interval selected via the timeline, we first identify the center of the collision's spatial extent. While one could imagine various weightings on the spatial contact points to compute a center position, we observe that simply taking the barycenter of the spatial bounding box $\mathcal{X}$ containing all contact points found within the collision interval is consistent, predictable and robust. The camera is constrained to place this center position along its viewing ray: two rotational parameters and the distance from the center remain unknown.

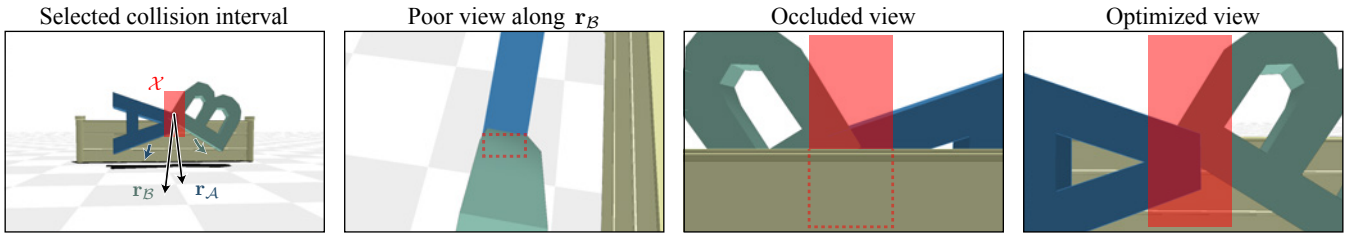


Figure 9: *After selecting a collision interval on the timeline, a user can jump to an optimized view by hitting a hot key. Views along the trajectory of an object (e.g., $\mathbf{r}_\mathcal{B}$) are poor. Views orthogonal to the movement of both objects are ideal. Of these two views, we choose the one with the least occlusion.*

We select the smallest distance containing the bounding box $\mathcal{X}$ completely in view for any angle.

Finally, to optimize the rotational degrees of freedom, we consider two scenarios: 1) collisions involving two objects and 2) collisions involving more than two objects.

Two objects Brief collisions involving two rigid objects $\mathcal{A}$ and $\mathcal{B}$ are well approximated by their relative translation. Given the *rigid* motions of $\mathcal{A}$, we define the direction of approximation motion during the collision interval with temporal extent $[t_0, t_1]$ to be simply the difference in center of mass at the end and beginning of the interval: $\mathbf{r}_\mathcal{A} = \mathbf{A}(t_1) - \mathbf{A}(t_0)$, and equivalently for $\mathbf{r}_\mathcal{B}$.

A very degenerate (non-optimal) view would be along $\mathbf{r}_\mathcal{A}$ or $\mathbf{r}_\mathcal{B}$: in this view one object occludes its own motion and likely also the collision with the other object. The ideal view places the approximate motion of both objects in the view plane: the view direction should be perpendicular to both $\mathbf{r}_\mathcal{A}$ and $\mathbf{r}_\mathcal{B}$. In other words, the viewing direction should be parallel to the cross product of these two directions: $\pm(\mathbf{r}_\mathcal{A} \times \mathbf{r}_\mathcal{B})$.

In this case, the optimization boils down to choosing between two views (see Figure 9). This decision is made based on a measure of the collision's occlusion. The exact occlusion induced by these and other objects meshes is difficult and expensive to measure. Our view selection is only useful if instantaneous. Therefore, we approximate occlusion via the traditional real-time rendering pipeline.

We render the scene with an opaque bounding box $\mathcal{X}$ around the collision's spatial extent and count the number of visible pixels of the bounding box. We implement this by rendering the scene once to set the depth buffer and then rendering only the bounding box again while counting pixels using the hardware "occlusion queries" (`GL_SAMPLES_PASSED` in OpenGL). In our experimental setup, this outperformed rendering with depth culling and counting pixels (even when using a GPGPU parallel reduction).

Many objects Now, let us consider the multi-object collision scenario. In this case, the approximate directions of all objects will not define a unique ideal viewing axis: any view axis will be non-perpendicular to some direction. We experimented with deriving a least-squares optimization to find an "as-perpendicular-as-possible" viewing direction, but this frequently resulted in degenerate views. Instead, we propose that in multi-object scenarios the prevention of occlusions outweighs the importance of finding a perfect direction relative to the object motions.

Optimizing over all views is infeasible, so we opt for a Monte-Carlo approach: we sample many (>1000) random views and choose the best according to the number of visible pixels from the collision box as described above. For scenes with an obvious "ground" (e.g., the space-saving kitchen in Figure 2), we restrict the sampling to the northern hemisphere. If the camera control is a trackball (i.e., without a fixed up axis), we uniformly sample random orbiting rotations as quaternions [Shoemake 1992]. If using "fixed up" camera controls, we uniformly sample directions via points on the unit (hemi-)sphere [Shao and Badler 1996].

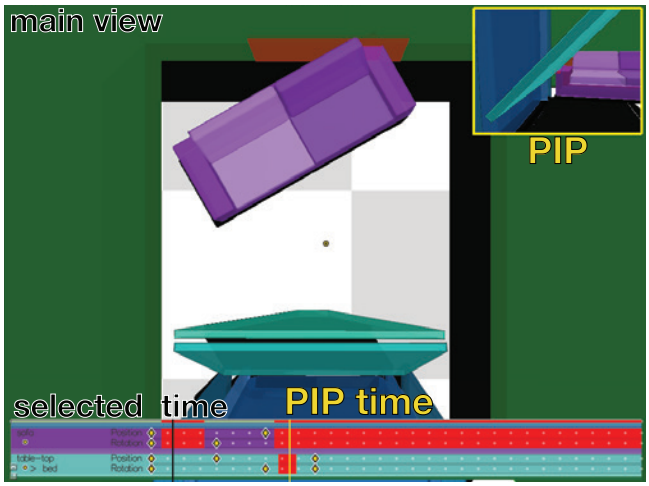


Figure 10: *As the designer attempts to resolve the current collision of the sofa against the wall, another collision is triggered and shown in the PIP view.*

4.4 Picture in Picture

Side effects are a common occurrence when trying to resolve collisions of a reconfigurable. While the designer focuses on repairing a particular CI, a new collision may form elsewhere in spacetime. Recognizing this, then scrubbing back and forth between collisions and their side effects hinders productivity.

Our tool actives a picture-and-picture (PIP) view when a side effect collision occurs. As seen in Figure 10, the PIP is view-optimized (see §4.3) on the side effect CI. The corresponding temporal position of the PIP is marked by a secondary yellow marker on the timeline. As the designer makes edits in the design the PIP view updates accordingly. When the designer clicks on the PIP window, the main and PIP views are swapped, accelerating the typical back-and-forth workflow required to achieve multiple conflicting spacetime corrections.

Triggering PIP Modification of the reconfigurable triggers background collision detection (see §5), which yields a set $\{I_1, I_2, \ldots\}$ of spacetime CIs, $I_i \in \mathbb{R}^3 \times [0, 1]$. By comparing the sets before and after the modification, $\{\hat{I}_i\}$ and $\{I_i\}$, respectively, we determine which CIs (i) remain unchanged, (ii) disappeared, (iii) appeared, or (iv) changed spatiotemporal extent. Whereas cases (i)

and (ii) do not call for the designer's attention, cases (iii) and (iv) correspond to side effects that the designer must review.

We first considered displaying multiple PIPs for the multiple CIs, but this clutters the user interface. Instead, we prioritize CIs and alerting the designer to the most critical side effect via a single PIP. This decision follows and reinforces the typical design workflow of attending to the most critical problems first.

We first prioritize *new* CIs, case (iii), over *altered* CIs, case (iv). Among new CIs, we prioritize *longer* time extents over *shorter* durations Among altered CIs, we prioritize *extending* over *contracting* durations. The sorting of PIPs based on duration is intended to match the intended workflow of honing in on CI time and then space.

The PIP, like all 3D views, is a slice through spacetime at a so-far-undetermined fixed time. Keeping with our design philosophy of attending to the most critical sections first, the time instance chosen corresponds to *when* the collision is *most severe* within the CI. Classically, one would use a measure of penetration depth to measure the severity of a collision region. However, measuring penetration depth usually requires the use of distance field methods, which introduce substantial computational overhead and are not suitable for interactive applications when geometry deforms significantly [Teschner et al. 2004]. Thus, we rely on our notion of intersection energy (1), detailed in §6.2, as a measure of severity. Having selected a most severe time during the CI, the PIP view shows the spatial collision configuration using the view selection algorithm (see §4.3).

Finally, the design can trigger the PIP view manually via a hotkey while hovering over a CI on the timeline. This enables easy navigation between CIs (see accompanying video).

5 Collision Detection

As the user edits and revises the design, our tool continuously runs a spacetime collision detection thread. This spacetime collision detection thread enables the user tools in the previous section to bring attention to any design trajectories that may interfere with one another, highlighting first *when* in time collisions occur before computing *exactly* where they occur.

Given a reconfigurable, our goal is to identify collision intervals (CIs) in spacetime where pairs of objects overlap. The following basic algorithm identifies all overlaps between piecewise-linear spacetime proxies. In order to achieve interactive rates for continuously modified designs, we use a spacetime k-DOP bounding volume hierarchy (BVH). For complex designs, recomputing the hierarchy on every edit would prohibit interactivity. We exploit the locality of design edits by *updating* and *traversing* only modified portions of the BVH.

5.1 Spacetime k-DOP

A wealth of bounding volume (BV) types have been explored [Klosowski et al. 1998]; we adopt k-DOPs [Klosowski et al. 1998], whose popularity stems from their easy construction, efficient update and comparison operations. A k-DOP bounds a volume by taking the intersection of tight fitting half-spaces associated with a fixed family of k axes. The question arises, how large should k be? In three dimensions, popular choices are 6 (an axis-aligned bounding box) and 26 (choosing the corners, edge midpoints, and face barycenters of the unit cube to form axes). If we interpret $\mathbb{R}^3$ instead as two spatial dimensions plus time (e.g. $\mathbb{R}^2 \times [0,1]$), then we may understand the effect of this choice in spacetime (see Figure 11).

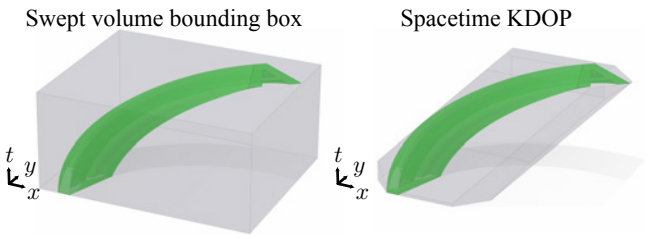


Figure 11: *The 2D object A rotates over time. The bounding box of the swept volume extruded into time is very conservative (left). A k-DOP in spacetime hugs the spacetime volume tighter.*

For our spacetime k-DOP, we consider a variety of generalizations. We could take the axis-aligned bounding box in four-dimensional spacetime: 8-DOP. On the opposite end of the spectrum, we could take an 80-DOP with halfspaces corresponding to all combinations of spatial and temporal axes. A good balance is to augment the generous spatial 26-DOP halfspaces with two temporal halfspaces, resulting in a 28-dop.

We implemented our bounding volume hierarchy generically to these choices. Our experiments reveal that the 28-DOP slightly outperforms the 80-DOP, and both greatly out perform the 8-DOP. Although the 80-DOP provides fewer false-positives, the spacetime proxies themselves only span small segments of time where false positives are rare. Therefore, we avoid extra computational cost of testing so many plane equations, using only the 28-DOP (on all our examples).

5.2 Constructing the Bounding Volume Hierarchy

We construct binary tree BVH for each object once at initialization. We employ a top-down tree construction strategy that recursively splits our BVH until each leaf contains a single spacetime proxy (a spacetime triangle). At each non-leaf node, the splitting procedure creates a splitting plane all the 4D vertices of the spacetime proxies. This plane is chosen orthogonal to the axis of the coordinate with maximum extent, centered at the median value. Note that this splitting does not necessarily create two disjoint children, i.e., it may result in children whose BVs overlap if a poxy is assigned partially to each halfspace. We experimented with increasing the branching factor to the tree: from binary to ternary and octonary, but we did not experience any performance gains.

5.3 Local, Lazy Refitting

As the designer edits the reconfigurable, the structure of each object's BVH remains intact, while the extents of the affected k-DOPs are updated. Since the designer typical makes structured, localized and continuous edits, we found that refitting is usually much faster than rebuilding the tree from scratch.

The animation keyframe user interface inherently leads to a temporal locality of revisions. Further, the use of a (mouse-)pointer-based interface typically results in spatially-localized editing. In general, only a small subset of the entire spacetime reconfigurable is altered with each designer gesture. Marking the affected spacetime proxies as "dirty," we exploit the locality of editing operations by refitting and "cleaning up" only those dirty k-DOPs impacted by dirty proxies. This local refitting allows for interactive collision detection, even for many objects in large scenes.

Correspondingly, pairwise BVH traversal need not be carried out over the complete set of all BVHs. Rather, in the spirit of *kinetic data structures* [Guibas 1998], only comparisons involving

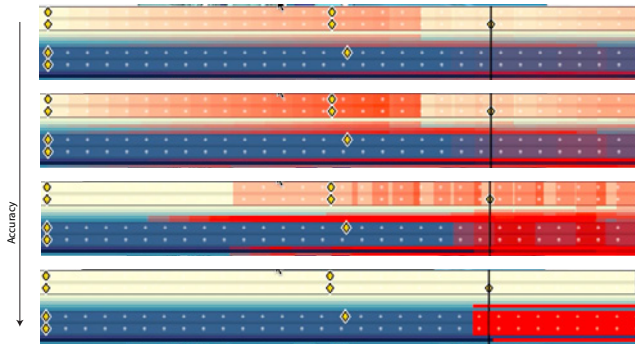


Figure 12: *Accuracy of the visualized CIs increase as deeper nodes in the BVH are processed. Opaque CIs are the most accurate.*

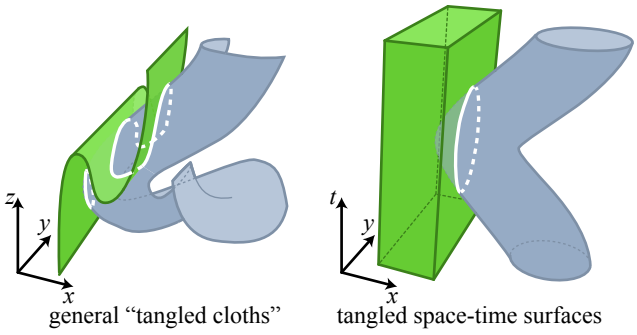


Figure 13: *Left: intersection of general 2D surfaces in 3D. Right: intersecting spacetime volumes of 2D curves.*

the "dirtied" subtrees of a BVH need to be checked against the clean subtrees.

5.4 Visualizing Collision Intervals

In order to provide the designer with the most relevant information immediately, we give precedence to first visualizing *when* collisions occur, so that the design process may benefit from immediate high-level feedback when edits invalidate trajectories of objects in the scene.

When Collisions Occur Although BVH traversal schemes typically proceed top-down based on a stack, we opt for a breadth-first traversal based on a queue. Each level of the BVH that is traversed in this way provides an initial "preview" to where potential collisions lurk. These previews are displayed on the timeline, with progressively increasing opacity at deeper tree levels (see Figure 12). While the local, lazy refitting typically allows for nearly instantaneous completion of all collision detection, the breadth first traversal with previews further reinforces the impression of immediacy even for operations that less localized, such as the automatic resolution described in §6.6.

Where Collisions Occur After honing in on the time extent of the CI, the designer can jump via the timeline or picture-in-picture to a moment within the CI. The BVH hones the spatial extents and returns a set of candidate spacetime primitives, involving two triangles a and b over a time step Δt. The final CIs are built by conducting continuous collision detection (CCD) over the time interval Δt. The solution to the CCD procedure provides a $\delta t \subset \Delta t$, where δt is the time-extents when a and b are overlapping. The CI stores this δt as well as the spatial extents of the collision region between a and b within δt. We store mappings between CIs and the objects to which the primitive a and b belong, allowing for fast queries of CI and their objects.

6 Spacetime Collision Resolution

Our automatic resolution of collisions is set apart from earlier work on collision response by its focus on altering spacetime configurations, rather than the more common goal of computing forces or displacements in the context of forward time integration.

To develop an automatic collision resolution method we must first define a notion of a four-dimensional constraint gradient or *contact normal*. For a spacetime point on our spacetime shapes this is the direction of the admissible region of configuration space: no interpenetrations.

6.1 Less Decent Descent Directions

Before embarking on our normal derivation, we summarize our early failures with various alternatives. An obvious approach is to move objects according to the spatial normal evaluated at the first instant of contact. Similar to our chain rule differentiation with respect to the UI, this direction can be projected onto the UI degrees of freedom. Beyond the philosophical question of what is a "first" instant of contact in a fictitious time dimension (and why is "first" better than "last," see Figure 15), we found this normal to be very sensitive, or "noisy," with respect to small perturbations to position or shape.

To restore temporal symmetry we considered averaging the contact normals at the first and last instants contact, but noise remained. To reduce the noise, we considered averaging over all contact normals throughout the collision duration, but the result was not useful: contact normals at a pair of overlapping mesh primitives, belonging to two already overlapping objects, can point in essentially any direction, and do not provide a useful direction for resolving the intersection.

Another alternative we considered was not a definition of the contact normal per se, but a different approach to collision response altogether. We considered treating the collision interval as a physical simulation, *plowing* through the collisions by integrating forward in time. To do this, we experimented with a traditional linear complementary problem (LCP) formulation of collision response, but found that the position-based variant did not always have a feasible solution [Erleben 2004] while the velocity-based variant was not guaranteed to resolve all collisions nor stay close to the designer's intended edit. Ultimately, we realized that our problem is best approached not as a collision *prevention/resolution*, rather as an existing spacetime mess that must be *untangled*.

6.2 Untangling Spacetime Cloth

We extend the untangling cloth method of [Volino and Magnenat-Thalmann 2006] to spacetime. Although we consider reconfigurable composed only of *rigid* objects moving in space, the corresponding trajectory spacetime volumes are highly deformable and thus susceptible to "tangling," analogous to how cloth tangles in $\mathbb{R}^3$.

Volino and Thalmann [2006] consider the global problem of identifying and minimizing the amount of overlap between deformable cloth surfaces in space. They do so by identifying the one-dimensional intersection contours and minimize their lengths via gradient descent.

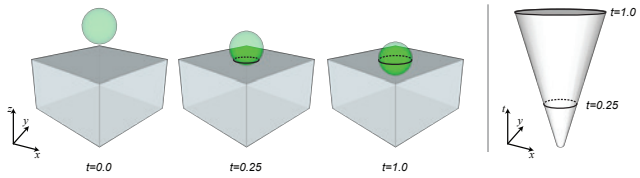


Figure 14: *Intersection between sphere and box. We illustrate intersection contours for three instants in time (three images, left), and depict the surface swept by all of intersection contours throughout the duration of the intersection (one image, right).*

6.3 Intersection Surface

At first, it appears that we require a full generalization of this approach from three to four dimensions. Resolving intersections between our spacetime trajectories indeed requires minimizing the *two-dimensional surface area* of the intersection of two three-dimensional volumes in spacetime.

We are saved however by the grace of the monotonicity of time. Time serves as a natural independent variable for parameterizing the spacetime trajectory and obtaining a 3D "slice" at an instant of time. A corollary is that spacetime trajectories may fold over in space but cannot fold over in time (see Figure 13). We exploit this crucial fact to simplify the spacetime generalization of the contour minimization method.

Suppose that the time interval $[t_0, t_1]$ encloses a collision between a pair of objects (i, j). At any instance $t \in [t_0, t_1]$, there exists at least one intersection contour (curve) between object i and object j (see Figure 14). Integrating these contours over the entire intersection interval sweeps the entire 3D intersection surface immersed in spacetime. Generalizing the earlier work on cloth untangling, we wish to minimize an *energy* measuring the area of this intersection surface $|I_S|$,

$$|I_S| = \int_{t_0}^{t_1} \|I_c(t)\| \mathrm{d}t \tag{1}$$

where, $I_c(t)$ is the intersection contour at a time t and $\|\cdot\|$ measures the length of the contour.

6.4 Contact Normals

For untangling two-dimensional cloths in $\mathbb{R}^3$, Volino and Thalmann [2006] use gradient descent to minimize the length of the one-dimensional intersection curve between the two surfaces: they move the n surface mesh vertices $\mathbf{V} \in \mathbb{R}^{n\times 3}$ along the gradient vector $\nabla_{\mathbf{V}} \|I_c\|$. This gradient vector is referred to as the *contact normal*.

We are dealing with "three-dimensional cloths" in $\mathbb{R}^3 \times [0, 1]$. For now, we analogously consider moving spacetime mesh vertices $\mathbf{V} \in \mathbb{R}^3 \times [0, 1]$ by minimizing the intersection volume's *surface area* $|I_S|$ in Equation (1), moving along $\nabla_{\mathbf{V}} |I_S|$. Because integration and differentiation commute, this gradient is simply a sum of the gradient of one-dimensional intersection curves for each moment along the collision interval's temporal extent:

$$\nabla_{\mathbf{V}} |I_S| = \int_{t_0}^{t_1} \nabla_{\mathbf{V}} \|I_c(t)\| \mathrm{d}t \tag{2}$$

Therefore, we directly employ their gradient calculation sampled regularly in time.

Since each one-dimensional intersection curve I_c is independent of time, the temporal component of our spacetime contact normal is zero. As a concrete example, consider the spatial component of the spacetime contact normal depicted in Figure 15. Naturally, the gradient of the *spatial* intersection curve length ($\|I_c(t)\|$) at a fixed time (t) with respect to the mesh vertex *spatial positions* at that time ($\mathbf{V}_t \in \mathbb{R}^{n\times 3}$) only reveals *spatial* information: all slices of the mesh through time seemingly stay in their time slice and do not influence each other.

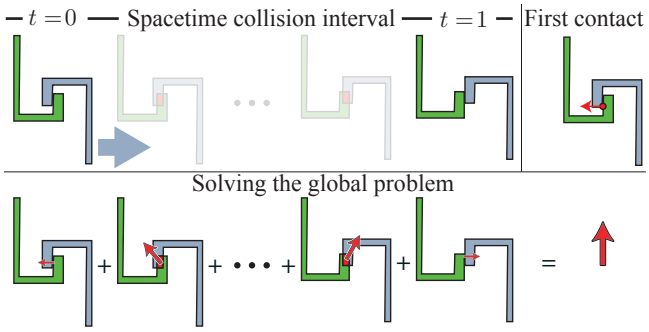


Figure 15: *Consider a 2D example of a grey hook moving horizontally over a green hook. The spatial normal at first (or last) contact has no vertical component; moving in this direction will only lead to more contact. In contrast, our method effectively solves the* global *problem resulting in a normal pointing in the vertical direction, successfully resolving the contact.*

However, this apparent independence between temporal slices is not as it initially appears. The spacetime mesh is coupled across time via our design user-interface degrees of freedom. We only consider rigid objects, so a change to an object's geometry will affect all moments in time. For example, changing a "rest pose" vertex position $\mathbf{p}_i$ will effect the vertex's position $\mathbf{v}_i(t)$ at every time t. Similarly, the keyframes used for UI elements (cf. §4) effect long stretches of time. Generally speaking, the gradient $\nabla_{\mathrm{UI}} |I_S|$ with respect to the UI will have a non-zero temporal action. Letting $\mathbf{G}_t = \nabla_{\mathbf{V}_t} \|I_c(t)\|$ and applying the chain rule, we can determine the gradient of the intersection surface area with respect to the user-interface (cf. [Harmon et al. 2011; Umetani et al. 2011]):

$$\nabla_{\mathrm{UI}} |I_S| = \mathrm{diag}\left((\nabla_{\mathrm{UI}} \mathbf{V}_0)\mathbf{G}_0, \ldots, (\nabla_{\mathrm{UI}} \mathbf{V}_1)\mathbf{G}_1\right), \tag{3}$$

that is, a block diagonal matrix involving the gradient of the mesh vertex positions with respect to the UI elements at each discrete time sample, $\nabla_{\mathrm{UI}} \mathbf{V}_t$. Although the temporal component of $\mathbf{G}_t$ is set to zero due to the time independence of $\|I_c(t)\|$, the chain-rule when applied to UI elements results in non-zero changes to the timing of UI keyframes. This follows since the timing of keyframes influence the object's motion in space. We apply the chain rule to this term for our most common UI elements (spherically interpolated quaternion and spline translation keyframes) in the Appendix, application to other user interactions follows analogously.

6.5 Resolution Hints

The contact normal computed in §6.4 immediately provides useful insight that the designer can harness to resolve interferences. The designer can explicitly request a hint for selected object of the current collision interval (CI).

The chain rule derived in the Appendix effectively projects the energy descent direction on a particular UI element's degrees of freedom, providing exactly the appropriate information necessary to draw a hint. For example, the components corresponding to a spline control point are interpreted as 3-vector in space rendered on screen as a large yellow arrow (see Figure 16). The 3D interface can make

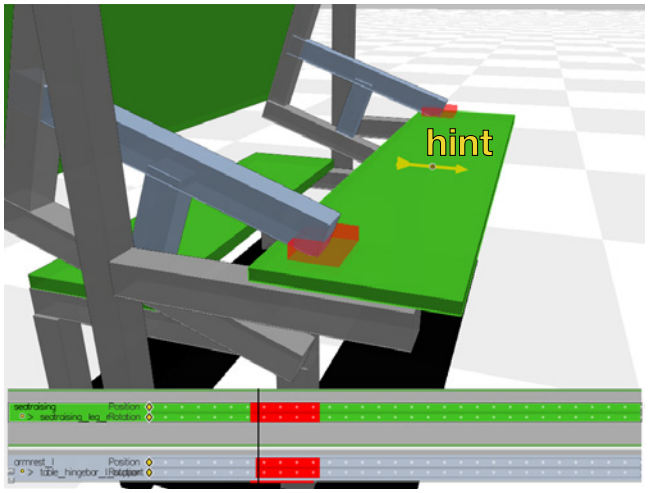


Figure 16: *The yellow arrow suggests how the designer could move the seat to avoid interference with the armrests.*

it difficult to precisely move in the direction of the hint, so we also provide a slider to move the UI element along the hint direction. The interface and the designer work symbiotically to solve the intersection.

Often however this magnitude can be determined automatically and the direction may change as result. This leads naturally to a gradient descent resolution scheme that is fully automatic.

6.6 Automatic Resolution

In many instances during the design process the designer is required to make several tedious edits that can often be automatically handled by the system. We formulate this automatic resolution scheme as a non-linear optimization problem that attempts to minimize the energy formulated as the surface area of the collision interface, as described in Equation (1). Our implementation of gradient descent employs the Golden Section Search Algorithm [Press et al. 1992] and assumes that the local energy landscape has a single global minimum nearby.

Our experiments showed higher success resolving multi-object interferences in pairs independently rather than as a simultaneous optimization over all pairwise interferences. We employ a greedy approach, resolving the collision between the pair of objects contributing most to the intersection energy (1). Resolving the most severe collisions first mimics the approach a designer might take, focusing first on severe collisions and later on grazing cases.

6.7 Spacetime Geometry Carving

In the previous sections we considered ways to mitigate collisions by altering objects via the available user-interface elements. We now consider altering an object's geometry dramatically by *carving* away the relative swept volume of another interfering object.

Ideally we would like to *subtract*—in a constructive solid geometry sense—an offset of the relative swept volume of the carving object B from the carved object A:

$$A \leftarrow A \setminus \text{offset}\left(\text{sweep}\left(B, f_A^{-1} \circ f_B\right), \varepsilon\right) \tag{4}$$

where $\text{offset}(X, \varepsilon) \subset \mathbb{R}^3$ computes an ε-offset of a given volume $X \subset \mathbb{R}^3$ and $\text{sweep}(X, f) \subset \mathbb{R}^3$ computes a swept volume of a given volume X undergoing a rigid motion $f(t)$.

If B and A are each undergoing rigid motions $f_A(t)$ and $f_B(t)$ respectively, then we consider the swept volume of B according to its motion *observed* in the reference frame of A.

Computing offset volumes and swept volumes of triangle meshes *exactly* poses computational and representational problems. The *exact* swept volume of a solid bounded by a triangle mesh undergoing a rigid motion is a piecewise-ruled surface [Weld and Leu 1990] (i.e., in general not representable with mesh of flat triangles). Similarly the *exact* offset surface of a solid bounded by a triangle mesh is a piecewise quadric surface [Pavic and Kobbelt 2008]. However, to fit into the rest of our pipeline, we need the output of this carving sub-routine to produce a new triangle mesh. Therefore, previous works on exact offsets and swept volumes are inappropriate in our contexts due to their output respresentation.

Previous tools for computing triangle-mesh approximations of swept volumes (e.g., [Peternell et al. 2005]) and surface offseting (e.g., [Campen and Kobbelt 2010]) focus on accuracy over performance and simplicity. Instead, we propose directly computing a conservative triangle-mesh approximation of an offset to the swept volume by contouring a signed distance field. We then subtract this approximation from the exact geometry of the static object to ensure its details remain in tact.

Approximate offset of swept volume Our approach adapts and accelerates the implicit method of [Schroeder et al. 1994]. Without loss of generality, the input to this subroutine is an object B, a rigid motion $f(t)$ for $t \in [0, 1]$, and a desired offset amount ε. The output is a triangle mesh approximating the ε-offset surface to the swept volume of B moving along $f(t)$ (see Figure 17).

First we lay a grid over the bounding box containing the spatial extent of B transformed by $f(t)$ for all $t \in [0, 1]$ padded by 2ε. The grid step size h is an exposed parameter trading off between computation time (larger is coarser, faster) and accuracy (smaller is finer, more accurate). The step size should be chosen smaller than the smallest relevant feature on B.

For each grid vertex, we will approximate the signed distance to the swept volume's surface. The swept volume S is the union of B moving along $f(t)$:

$$S = \bigcup_{t \in [0,1]} f(t)B. \tag{5}$$

This is easily re-written in terms of signed distances (assuming negative distances inside a solid):

$$d(S, \mathbf{p}) = \min_{t \in [0,1]} d\left(f(t)B, \mathbf{p}\right), \tag{6}$$

where $d(X, \mathbf{p})$ is the signed distance from the surface of some solid X to a query point $\mathbf{p}$.

Given a signed distance field to S, the surface of S (or any offset) can be extracted as the zero (ε) level set. We approximate this on our grid by taking small discrete steps in time and using marching cubes to extract the level set at ε. Because the signed-distance field is only useful insofar as it reveals the ε level set, we cull grid vertices determined far enough away ($> \sqrt{3}\varepsilon$) or too far inside ($< \varepsilon - \sqrt{3}\varepsilon$) during acceleration tree evaluation.

Finally, we subtract our triangle mesh approximation of S from the triangle mesh representation of A using the robust boolean library within LIBIGL [Jacobson et al. 2013].

It is tempting to conduct this final boolean subtraction also on the signed distance grid, using the signed distance to A. This would

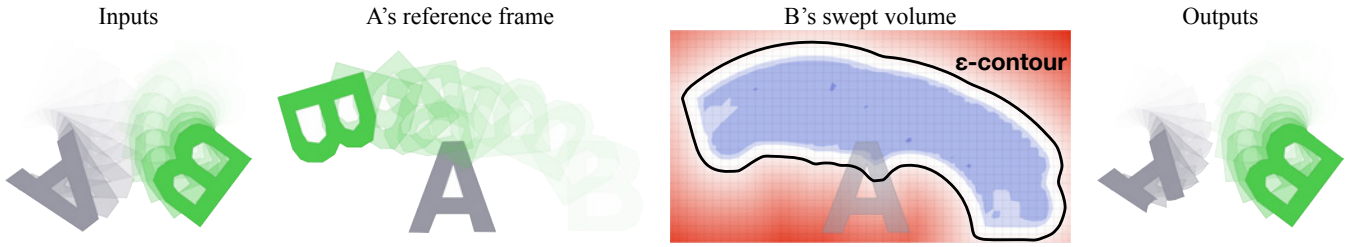


Figure 17: *Two objects A and B overlap in spacetime (ghosting projection, left). We consider the relative motion of B in A's reference frame. Sampling densely in time we aggregate the signed distance to B on a grid as B transitions. We contour the ϵ-offset surface to the swept volume and subtract this mesh from A. The new A does not overlap B in space time.*

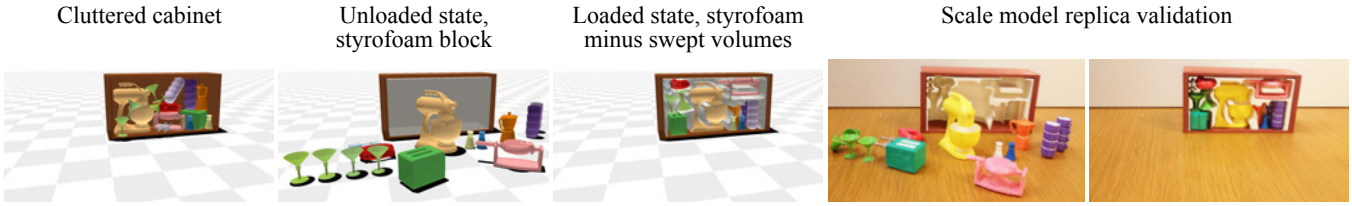


Figure 18: *A cluttered kitchen cabinet is tidied up by inserting a styrofoam block and subtracting the swept volumes of each object placed into place. Using a 3D printer, we validate this design.*

certainly be more efficient, but unfortunately forfeits the sharp details and sparse representation of A.

Instead, our approach uses an approximate representation of the swept volume S, but conducts the boolean subtraction exactly. This maintains the original details of A away from the subtracted region, forfeiting details of the swept volume, but this is already abstracted from the designer and obscured by the necessary offsetting.

7 Results

We implemented our prototype in C++11 using a background thread (`std::thread`) to detect collisions as the designer makes edits. On a MacBook Air 2GHz Intel Core i7 8GB memory machine, our spacetime collision detection and resolution runs interactively for our examples (meshes with 100 to 10,000 triangles). For examples with multiple transitions, each transition runs an instance of our 3D viewer with keyframe timeline. End states and object geometries are shared and edits propagate immediately. Changes to the collision intervals (CIs) are announced to the graph view.

In Figure 3, the designer would like to add armrests to a park bench that reconfigures into a picnic table. The armrests do not cause problems at either end state, but do cause collisions along the way. The designer experiments with a variety of solutions available within our tool: (a) carving away the swept volume of the armrests from the bench seat, (b) interactively reshaping the armrests until collisions disappear, and (c) adding a hinge mechanism to stow the armrests beneath the table.

In Figure 5, the designer adds various accessories to a folding bicycle. Adding a basket and altering the handle bars creates collisions (red highlight) when folding laterally. After interactive experimentation, the designer finds that changing the folding mechanism will accommodate the additions.

The reconfigurable kitchen in Figure 2 involves a graph of six transitions between seven states. We show a few of the most interesting states with ghosting (see §4.2) to indicate transitions. In the clean up state (1), all deployable objects are hidden and there is plenty of space to walk around. In the food prep state (2), the stove vent lowers to reveal more counter space. In appliance state (4), medium-size machines appear ready for use via a Murphy-bed-style shelving system. If more counter space is needed when using the appliances, extra counter space unfolds over the sink (5). The designer employs the carving tool of §6.7 to leave space for the faucet. When cooking is done and the appliances are put away, the telescoping table deploys and benches swing out from under the cabinets (6). See the accompanying material for a video of the full length editing session of this example.

Reconfigurability helps tidy up a chaotic cabinet of appliances, cups, and glass in Figure 18. In this theoretical example, the designer considers filling the cabinet with a block of styrofoam and then carving from it the swept volume of each object as it is placed into a non-overlapping position in the cabinet. We validated the effectiveness of this idea with a 3D-printed scale model.

In Figure 20, the designer plans furniture arrangements for a reconfigurable studio apartment in Manhattan. By identifying and resolving impossible transitions, the apartment fits a wide assortment of furniture featuring a sleeping mode, bathroom mode, and entertainment mode.

Our tools function on the macro apartment-size scale and also on the micro scale. In Figure 19, the designer reshapes and adds complexity to a reconfigurable Burr puzzle. We validated this result using a 3D printer: all parts fit together neatly.

7.1 Limitations & Future Work

We focused the feature set of our prototype on visualization, monitoring and resolution aids unique to the problem of designing reconfigurables. We delegated advanced geometric editing to third party tools. Integrating advanced real-time mesh editing [Gal et al. 2009; Liu et al. 2014] into our tool should be possible.

We also assumed that the designer had geometric models available. While 3D scanning is becoming more common place, we imagine that integrated the wealth of online 3D data (à la [Schulz et al. 2014]) would be an interesting extension.

Finally, our interpretation of physical feasibility is limited to interpenetration during reconfiguration. We rely on the user's human intuition or domain expertise to prevent unnatural or mechanically impossible transitions (e.g., levitating couches). Adapting our con-

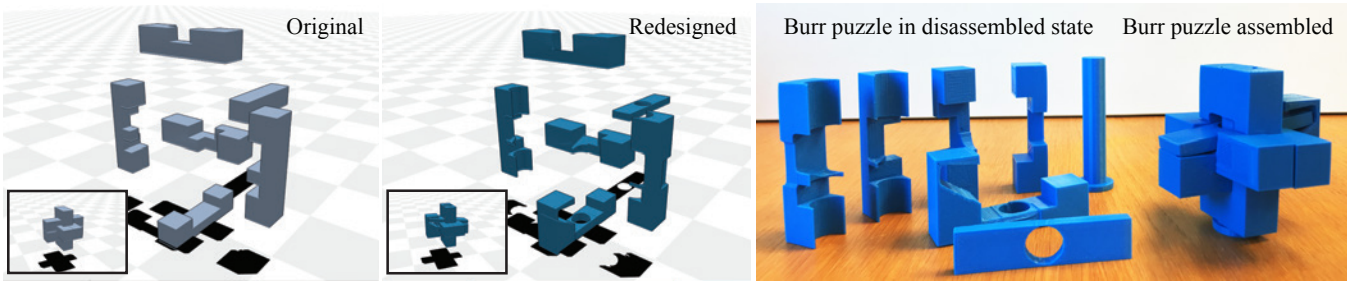


Figure 19: *We validated the feasibility of our Burr puzzle design by 3D printing the parts.*

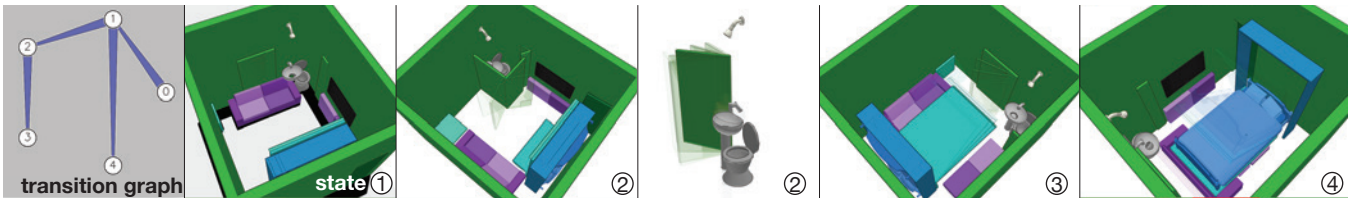


Figure 20: *A reconfigurable apartment. Transition graph (leftmost) summarizes four overall states. (1) An open floorplan is used for parties and housekeeping. (2) In daytime and evening, the sofa faces the TV, the WC is accessible via a swinging door, and for showering, the sink folds up, the undersink piping telescopes into the drain. (3) For dining, a table and benches swing into place; the sink remains accessible but not WC. (4) At night, a wall bed swings down.*

tact resolution optimization to account for mechanical constraints would be a challenging but exciting future work.

Acknowledgements

The Columbia Computer Graphics Group is supported by Disney Research, Pixar, Adobe, and Altair. We thank Keenan Crane and Henrique Maia for illuminating discussions. Funded in part by NSF grants CMMI-11-29917, IIS-12-08153, IIS-14-09286, and IIS-17257.

References

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*, 4 (July), 82:1–82:10.

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph.*.

BÄCHER, M., WHITING, E., BICKEL, B., AND SORKINE-HORNUNG, O. 2014. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph.*.

BÄCHER, M., COROS, S., AND THOMASZEWSKI, B. 2015. Linkedit: interactive linkage editing using symbolic kinematics. *ACM Trans. Graph.*.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *ACM Trans. Graph. 22*, 3, 862–870.

BERNSTEIN, G. L., AND WOJTAN, C. 2013. Putting holes in holey geometry: Topology change for arbitrary surfaces. *ACM Trans. Graph.*.

BHARAJ, G., LEVIN, D. I. W., TOMPKIN, J., FEI, Y., PFISTER, H., MATUSIK, W., AND ZHENG, C. 2015. Computational design of metallophone contact sounds. *ACM Trans. Graph.*.

CAMERON, S. 1990. Collision detection by four-dimensional intersection testing. *IEEE T. Robotics and Automation*.

CAMPEN, M., AND KOBBELT, L. 2010. Polygonal boundary evaluation of minkowski sums and swept volumes. *Comput. Graph. Forum*.

CEYLAN, D., LI, W., MITRA, N. J., AGRAWALA, M., AND PAULY, M. 2013. Designing and fabricating mechanical automata from mocap sequences. *ACM Trans. Graph.*.

ERLEBEN, K. 2004. *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, Univ. of Copenhagen.

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., nVidia Corp.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*.

GARG, A., SAGEMAN-FURNAS, A. O., DENG, B., YUE, Y., GRINSPUN, E., PAULY, M., AND WARDETZKY, M. 2014. Wire mesh design. *ACM Trans. Graph.*.

GUIBAS, L. J. 1998. Kinetic data structures–a state of the art report. *Proc. WAFR*.

HARMON, D., PANOZZO, D., SORKINE, O., AND ZORIN, D. 2011. Interference-aware geometric modeling. *ACM Trans. Graph.*.

IGARASHI, Y., IGARASHI, T., AND MITANI, J. 2012. Beady: Interactive beadwork design and construction. *ACM Trans. Graph.*.

JACOBSON, A., PANOZZO, D., ET AL., 2013. libigl: A simple C++ geometry processing library. http://igl.ethz.ch/projects/libigl/.

JOUBERT, N., ROBERTS, M., TRUONG, A., BERTHOUZOZ, F., AND HANRAHAN, P. 2015. An interactive tool for designing quadrotor camera shots. *ACM Trans. Graph.*.

KAVRAKI, L. E., ŠVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE TRA*.

KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE TVCG*.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph.*.

LI, H., ALHASHIM, I., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2012. Stackabilization. *ACM Trans. Graph.*.

LIU, S., JACOBSON, A., AND GINGOLD, Y. 2014. Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces. *ACM Trans. Graph.*.

LIU, T., HERTZMANN, A., LI, W., AND FUNKHOUSER, T. 2015. Style compatibility for 3D furniture models. *ACM Trans. Graph.*.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*.

PAVIC, D., AND KOBBELT, L. 2008. High-resolution volumetric computation of offset surfaces with feature preservation. *Comput. Graph. Forum*.

PETERNELL, M., POTTMANN, H., STEINER, T., AND ZHAO, H. 2005. Swept volumes. *Computer-Aided Design Appl.*.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *Proc. SIGGRAPH*.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

SCHROEDER, W. J., LORENSEN, W. E., AND LINTHICUM, S. 1994. Implicit modeling of swept surfaces and volumes. In *Proc. of the Conference on Visualization*.

SCHÜLLER, C., PANOZZO, D., AND SORKINE-HORNUNG, O. 2014. Appearance-mimicking surfaces. *ACM Trans. Graph.*.

SCHULZ, A., SHAMIR, A., LEVIN, D. I. W., SITTHI-AMORN, P., AND MATUSIK, W. 2014. Design and fabrication by example. *ACM Trans. Graph.*.

SECORD, A., LU, J., FINKELSTEIN, A., SINGH, M., AND NEALEN, A. 2011. Perceptual models of viewpoint preference. *ACM Trans. Graph.*.

SHAO, M.-Z., AND BADLER, N. 1996. Spherical sampling by archimedes' theorem. Tech. rep., Univ. of Penn.

SHOEMAKE, K. 1992. Uniform random rotations. In *Graphics Gems III*. Morgan Kaufmann.

SKOURAS, M., THOMASZEWSKI, B., BICKEL, B., AND GROSS, M. 2012. Computational design of rubber balloons. *Comput. Graph. Forum*.

SKOURAS, M., THOMASZEWSKI, B., COROS, S., BICKEL, B., AND GROSS, M. 2013. Computational design of actuated deformable characters. *ACM Trans. Graph.*.

SNIBBE, S. S. 1995. A direct manipulation interface for 3d computer animation. *Comput. Graph. Forum*.

SUN, T., AND ZHENG, C. 2015. Computational design of twisty joints and puzzles. *ACM Trans. Graph.*.

TANG, M., MANOCHA, D., YOON, S.-E., DU, P., HEO, J.-P., AND TONG, R.-F. 2011. Volccd: Fast continuous collision culling between deforming volume meshes. *ACM Trans. Graph. 30*, 5 (Oct.), 111:1–111:15.

TESCHNER, M., KIMMERLE, S., ZACHMANN, G., HEIDELBERGER, B., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., AND STRASSER, W. 2004. Collision detection for deformable objects. In *Proc. Eurographics (STAR)*.

THOMASZEWSKI, B., COROS, S., GAUGE, D., MEGARO, V., GRINSPUN, E., AND GROSS, M. 2014. Computational design of linkage-based characters. *ACM Trans. Graph.*.

UMETANI, N., KAUFMAN, D. M., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment editing and modeling. *ACM Trans. Graph.*.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*.

UMETANI, N., KOYAMA, Y., SCHMIDT, R., AND IGARASHI, T. 2014. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.*.

VOLINO, P., AND MAGNENAT-THALMANN, N. 2006. Resolving surface collisions through intersection contour minimization. *ACM Trans. Graph.*.

WANG, B., FAURE, F., AND PAI, D. K. 2012. Adaptive image-based intersection volume. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4.

WELD, J. D., AND LEU, M. C. 1990. Geometric representation of swept volumes with application to polyhedral objects. *Int. J. Rob. Res.*.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. *Proc. SIGGRAPH*.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3d models. *ACM Trans. Graph.*.

YU, L.-F., YEUNG, S. K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*.

ZHOU, Y., SUEDA, S., MATUSIK, W., AND SHAMIR, A. 2014. Boxelization: Folding 3d objects into boxes. *ACM Trans. Graph.*.

Appendix: Chain-rule

We expand upon the gradient of the mesh vertex positions at a fixed time with respect to the user-interface elements, $\nabla_{\mathrm{UI}}\mathbf{V}_t$ in Equation (3).

We assume all objects of the reconfigurable to be rigid (their geometric shapes are constant over time up to rigid motion). The degrees of freedom exposed to the user are: movements of each mesh vertex's "rest pose" location, $\mathbf{p} \in \mathbb{R}^3$); cubic Bézier spline translational keyframes, each including backward, interpolated and forward spatial points $\mathbf{u}, \mathbf{t}, \mathbf{w} \in \mathbb{R}^3$ and a time $s \in [0, 1]$ on the timeline; and spherically interpolated rotational keyframes, each including a rotation $\theta \in SO(3)$ and a time on the timeline $\sigma \in [0, 1]$.

Exploding all degrees of freedom in our UI as a vector we have:

$$\mathrm{UI} = (\mathbf{p}_1, \ldots, (\mathbf{u}_1, \mathbf{t}_1, \mathbf{w}_1, s_1), \ldots, (\theta_1, \sigma_1), \ldots) \,. \qquad (7)$$

Now, considering the gradient of the mesh vertex positions at a time t with respect to the user-interface decomposes into partial derivatives:

$$\nabla_{\text{UI}}\mathbf{V}_t = \begin{pmatrix} \frac{\partial \mathbf{v}_1}{\partial \mathbf{p}_1} & \cdots & \frac{\partial \mathbf{v}_n}{\partial \mathbf{p}_1} \\ \vdots & \ddots & \\ \frac{\partial \mathbf{v}_1}{\partial(\mathbf{u}_1,\mathbf{t}_1,\mathbf{w}_1,s_1)} & & \\ \vdots & & \\ \frac{\partial \mathbf{v}_1}{\partial(\theta_1,\sigma_1)} & & \\ \vdots & & \end{pmatrix}, \tag{8}$$

where $\mathbf{v}_i$ is the vertex position at time t.

That vertex position $\mathbf{v}_i$ is determined by its rest pose $\mathbf{p}_i$, the object's center of mass $\mathbf{c} \in \mathbb{R}^3$, the current rotation $\theta(t) \in SO(3)$ about that center mass, and the current translation $\mathbf{x}(t) \in \mathbb{R}^3$:

$$\mathbf{v}_i = \theta(t)(\mathbf{p}_i - \mathbf{c}) + \mathbf{c} + \mathbf{x}(t). \tag{9}$$

Let us consider each type of partial derivative in turn.

Clearly moving a rest-pose vertex position does not influence some other vertex, so

$$\frac{\partial \mathbf{v}_i}{\partial \mathbf{p}_j} = \delta_{ij}\theta(t) \in \mathbb{R}^{3\times 3}, \tag{10}$$

where δ_{ij} is Kronecker's delta.

The current translation $\mathbf{x}(t)$ is defined by the cubic Beziér interpolation of the two keyframes immediately closest to t on either side, $s_j < t < s_{j+1}$:

$$\mathbf{x}(f) = (1-f)^3\mathbf{t}_j + \tag{11}$$
$$3(1-f)^2 f\mathbf{w}_j + \tag{12}$$
$$3(1-f)f^2\mathbf{u}_{j+1} + \tag{13}$$
$$f^3\mathbf{t}_{j+1} \tag{14}$$
$$f = (t-s_j)/(s_{j+1}-s_j). \tag{15}$$

Changing a translational spline keyframe has no effect outside its immediate neighborhood:

$$\frac{\partial \mathbf{v}_i}{\partial(\mathbf{u}_j,\mathbf{t}_j,\mathbf{w}_j,s_j)} = \frac{\partial \mathbf{x}(t)}{\partial(\mathbf{u}_j,\mathbf{t}_j,\mathbf{w}_j,s_j)}, \tag{16}$$
$$(= 0 \text{ if } t \le s_{j-1} \text{ or } t \ge s_{j+1}). \tag{17}$$

Otherwise, let us assume (by symmetry) that $s_j < t < s_{j+1}$, then

$$\frac{\partial \mathbf{x}}{\partial f} = 3(1-f)^2(\mathbf{w}_j - \mathbf{t}_j) + \tag{18}$$
$$6(1-f)f(\mathbf{u}_{j+1} - w_j) + \tag{19}$$
$$3t^2(\mathbf{t}_{j+1} - \mathbf{u}_{j+1}), \tag{20}$$
$$\frac{\partial \mathbf{x}}{\partial \mathbf{t}_j} = (1-f)^3, \tag{21}$$
$$\frac{\partial \mathbf{x}}{\partial \mathbf{w}_j} = 3(1-f)^2 f, \tag{22}$$
$$\frac{\partial f}{\partial s_j} = -1/(s_{j+1}-s_j) + (t-s_j)/(s_{j+1}-s_j)^2. \tag{23}$$

The current rotation $\theta(t)$ is defined by the spherical interpolation of the the two keyframes immediately closest to t on either side, $\sigma_j < t < \sigma_{j+1}$:

$$\theta(g) = \theta_j(\theta_j^{-1}\theta_{j+1})^g, \tag{24}$$
$$g = (t-\sigma_j)/(\sigma_{j+1}-\sigma_j). \tag{25}$$

where we now interpret θ_j as a unit quaternion.

Changing a rotational keyframe has no effect outside its immediate neighborhood:

$$\frac{\partial \mathbf{v}_i}{\partial(\theta_j,\sigma_j)} = \frac{\partial \mathbf{v}_i}{\partial\theta(t)}\frac{\partial\theta(t)}{\partial(\theta_j,\sigma_j)} \tag{26}$$
$$= (\mathbf{p}_i - \mathbf{c})\frac{\partial\theta(t)}{\partial(\theta_j,\sigma_j)}, \tag{27}$$
$$(= 0 \text{ if } t \le s_{j-1} \text{ or } t \ge s_{j+1}). \tag{28}$$

Again, let us assume that $\sigma_j < t < \sigma_{j+1}$:

$$\frac{\partial\theta(g)}{\partial g} = \theta(g)\log(\theta_j^{-1}\theta_j). \tag{29}$$

The final remaining term, $\frac{\partial\theta(g)}{\partial\theta_j}$, is left to the ambitious reader or to the industrious MATHEMATICA.