Eurographics Symposium on Geometry Processing 2016
Maks Ovsjanikov and Daniele Panozzo
(Guest Editors)

Volume 35 (2016), Number 5

Interactive Modeling of Mechanical Objects

Francisca Gil Ureta[†], Chelsea Tymms[‡], and Denis Zorin[§]

New York University

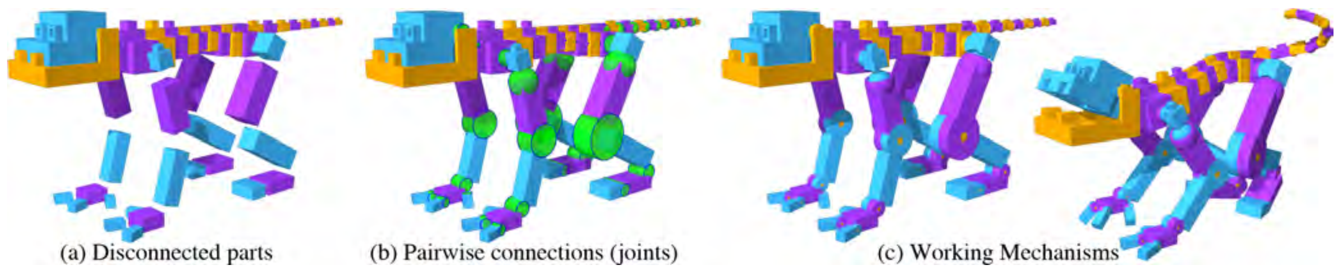

Figure 1: *Our system enables users to create working, printable mechanisms from a set of disconnected parts by adding pairwise connections (joints). The model shown here has more than 30 hinge and sphere joints added using our system in about 15 minutes. Original model "Blocko" by Mikkel_bf.*

Abstract

Objects with various types of mechanical joints are among the most commonly built. Joints implement a vocabulary of simple constrained motions (kinematic pairs) that can be used to build more complex behaviors. Defining physically correct joint geometry is crucial both for realistic appearance of models during motion, as these are typically the only parts of geometry that stay in contact, and for fabrication. Direct design of joint geometry often requires more effort than the design of the rest of the object geometry, as it requires design of components that stay in precise contact, are aligned with other parts, and allow the desired range of motion. We present an interactive system for creating physically realizable joints with user-controlled appearance. Our system minimizes or, in most cases, completely eliminates the need for the user to manipulate low-level geometry of joints. This is achieved by automatically inferring a small number of plausible combinations of joint dimensions, placement and orientation from part geometry, with the user making the final high-level selection based on object semantic. Through user studies, we demonstrate that functional results with a satisfying appearance can be obtained quickly by users with minimal modeling experience, offering a significant improvement in the time required for joint construction, compared to standard modeling approaches.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Computational Geometry and Object Modeling—[Geometric algorithms, languages, and systems];

1. Introduction

Mechanical objects are one of the most ubiquitous types of man-made objects. Many of the objects with which we interact during the day are part of a mechanism, including doors, faucets, coffee machines, cars, desks, computers, lamps, and tools. Unsurprisingly, creating models of mechanical objects is a common task. With standard modeling approaches, this task requires considerable effort and expertise. At the same time, as 3D printing and other technologies for personalized manufacturing become commonplace, the number of novice users creating 3D models, and particularly mechanical models, is rapidly increasing.

Constructing operational mechanical models is often difficult. For example, many of the models offered by Shapeways, one of the largest online services for 3D printing, *look like* mechanical objects but do not include functional mechanical joints. We conjecture that

[†] e-mail: gilureta@cs.nyu.edu
[‡] e-mail: tymms@cs.nyu.edu
[§] e-mail: dzorin@cs.nyu.edu

part of the reason for this is that modeling mechanisms, with actual moving parts, is far more difficult than modeling static shapes. The goal of our work is to develop a method to simplify and streamline the creation of basic mechanisms for users with moderate or no modeling experience as well as for more advanced users.

Formally, a mechanism is a structure composed of multiple rigid bodies (*parts*) interconnected at specific areas (*joints*). The geometry of the joint limits the relative motion of the connecting parts, thus defining the behavior of the mechanism [MGD94]. Consider, for example, the object in Figure 1: the ball-and-socket joints in the shoulders and the tail allow three-dimensional rotation while the hinges in the knees and fingers rotate about a single axis. As we can see, the kinematic behavior of a mechanism is intimately related to the geometry of the joints. Therefore, to create a good-quality mechanism, an artist must be able to understand how the geometry of a joint affects its motion.

In traditional 3D modeling tools, the geometry and motion of the joints are defined separately. In a typical scenario, the artist first models the geometry of all the mechanism's parts as independent components. Then, the artist creates a system of virtual joints (*rig*, or skeleton) which defines the kinematic behavior. Finally, the artist bounds each part of the model to a joint, with the part inheriting its relative motion.

We observe several major challenges with applying the traditional modeling workflow to mechanisms. First, the position, orientation, and proportions of the joint need to be computed by the artist, who must ensure the joint implements the desired motion. This is accomplished by a combination of time-consuming experimentation and geometric calculations, which are not always simple [KLY*14]. Second, since the behavior of the mechanism can only be tested after the geometry is created and associated with a separately-created skeleton, errors are often found late in the process; parts that look good statically might intersect or become disconnected during motion. And third, finding and fixing these errors is not a trivial task for complex shapes and may require many iterations.

In this paper, we present an approach to modeling mechanical objects that addresses these challenges. The two main principles that we use are (1) integration of the geometry definition with the rig construction, which defines motion, and (2) inferring the geometry of joints from the geometry of parts semi-automatically, allowing the user to seamlessly explore the space of joint configurations.

The contributions of our work include a semi-automatic technique to find a set of feasible joint positions, orientations and sizes for a pair of parts; a user interface design for exploring these configurations; and a method for adapting part geometry, when necessary, to the desired motion ranges.

We demonstrate our approach by converting a number of example 3D shapes into functional mechanisms with aesthetically satisfactory joints. Our evaluation shows that inexperienced users can efficiently use our interface and that overall performance is significantly better than what can be achieved with traditional modeling tools.

2. Related Work

Modern commercial CAD software for designing mechanisms and assemblies is tailored for professional machine designers and mechanical engineers (eg. SolidWorks, SolidEdge, Inventor). To properly use these systems, the user must have a good understanding of mechanics and modeling, which makes them unfit for hobbyists and novice users. In contrast, with our system, a user can create mechanisms, with physically realizable joints, without having to manipulate low-level geometry of joints.

The closest work, in terms of goals, is [KLY*14]. This paper addresses the problem of creating works-like prototypes of mechanical objects. The goal is the physical realization of a particular functionality defined by high-level functional relations between parts, with focus on converting these high-level relations to low-level kinematics. The work is restricted to parts shaped as cuboids and does not consider user-interaction or aesthetic aspects of joint placement; this makes it possible to use a minimalistic algorithm for joint placement, orientating and sizing.

A similar problem, that of creating articulated characters, is solved by [BBJP12] and [CCA*12]. These papers describe a pipeline for converting a character shape (typically, a single-component organic shape) into an articulated model by automatic segmentation and joint insertion. Both methods rely, in a fundamental way, on placing the joints in the interior of the object and on the availability of a skeleton or skinning weights to determine joint size and placement (with orientation of hinges in [BBJP12] determined manually). Although our targeted output models share similarities, we target input models consisting of separate parts, as is typical for mechanical objects, and do not assume a skeleton as input. We only require specification of pairs of parts to be connected by joints, with no geometric information as input, and focus on inferring a set of plausible options for joint geometry from part geometry.

Mechanism Design. A number of papers in graphics are concerned with design methods that facilitate the creation of functional objects. For instance, [HL15] turns a 2D mechanism into a working 3D model, but is restricted to planar mechanisms. [KSS*15] describes a method for automatically creating 3D-printable connectors; however, they use only rigid rod connectors.

Other works propose methods for designing mechanisms from a purely functional point of view: [ZXS*12] is concerned with an automatic design and placement of driving mechanisms, attached to an existing articulated character, to match simple motion trajectories created by a user. A related problem is solved in [CTN*13], handling a larger space of possible motion trajectories. [TCG*14] and [BCT15] present a system for the design and editing of linkage-based characters, allowing the creation of characters actuated by a single motor that achieves the desired motion while maintaining aesthetic appeal. All these methods require an existing virtual mechanical character as input, whereas our system creates such a mechanical character as the output.

More recently, [MTN*15] presents a system for the design of mechanical robots. Their approach uses the skeletal structure of the robot as input and adds motors at joint locations. The geometry for the body parts is automatically generated to connect the motors. The user can't easily modify the shape of the parts; augmenting the geometry is allowed, but the user must ensure there is no interference during motion. With our system, the user has greater control of the aesthetics of the model, while never having to manipulate low-level geometry.

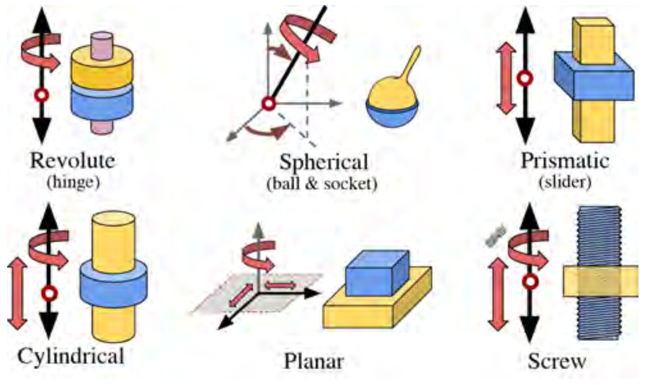

Figure 2: *Six standard kinematic pairs.*

Several works, [LOMI11, UIM12, LHAZ15], focus on furniture design. In particular, [LOMI11] describes a system for furniture assembly based on grammars, including adding parametrized joints from a library. The goal of this work is similar to ours, but it focuses on a narrower domain, which allows the use of grammars and a higher degree of automation. However, this approach restricts the possibilities of joint placement and sizing.

[SSL*14] demonstrates how conceptual sketches can be converted to manufacturable objects by adding the required connectors and joints based on a database of examples. In principle, this type of approach could be viewed as the ultimate goal for the problem we are solving, allowing near-complete automation, but in its current form, it requires a large and precisely-annotated collection of domain-specific examples.

A number of papers, e.g. [MYY*10] and [XWY*09], consider the complementary problem of estimating joint kinematics from surface geometry. These methods use the geometry of the joint, already present in the model, to determine the nature of the connections using primitive fitting [MYY*10] or slippage analysis [XWY*09].

Swept Volume. An essential feature of our method is the adaptation of part geometry to allow for desired joint motion, for which sweeps are required. Many general algorithms exist to compute swept volumes, such as [AMYBJ06] and more recently [vDH11], but these are relatively slow. We use an efficient special-case algorithm described in Section 7 to create the sweep envelope, using a method similar to that of [PPSZ05].

3. Simple Mechanisms and Joints

Mechanical joints between parts are described in terms of kinematic pairs of primitives. In this paper, we are primarily concerned with *lower pairs*, [DH64], i.e. joints that constrain a point, line, or plane of one part to a point, line, or plane in another part.

There are six standard kinematic pairs (Figure 2):

1. *A revolute pair (hinge)* requires two collinear lines in the connected parts, with no motion along these lines; that is, planes on each part, perpendicular to the aligned lines, maintain contact with each other. It has one rotational degree of freedom.
2. *A spherical joint* constrains two points in the connected parts to have the same location. It has three rotational degrees of freedom.
3. *A prismatic joint (slider)*, just as the hinge, requires the alignment of two lines in the connected parts and contact between two planes, but now the planes are *parallel* to the common line. It has one translational degree of freedom.
4. *A cylindrical joint*, like the hinge and slider, requires two collinear lines in the connected parts, but it does not add plane constraints. It has two degrees of freedom, one translational and one rotational, and it can be viewed as a combination of a hinge and a slider.
5. *A planar joint* requires that a plane in one part remain in contact with a plane in the other part. It has three degrees of freedom, two translational and one rotational.
6. *A screw joint* requires that helical lines (threads) on two parts remain aligned; it has one degree of freedom, corresponding to a mutually constrained translation and rotation.

In this paper, we consider linked assemblies using the first three pairs on the list (hinge, sphere, and slider), but effectively all except the screw joint can be handled: construction of a cylindrical joint is easily reducible to a hinge joint with a free translational axis; and the plane joint, although rarely used, can be composed by two sliders and a hinge.

While it would be simple to add the screw joint, we have not found many examples of its use; this type of joint is primarily used in linear actuators as a part of more complex mechanisms involving higher pairs (gears or cams), which we do not consider. The principles we developed for interactively placing and determining dimensions of the joints can also be applied to more complex joints, such as gimbal, six-degrees-of-freedom joint, universal joint, etc. All these joints can typically be decomposed into the primitive joints described here.

4. Overview

Our approach to modeling mechanical objects is based on several principles:

- The user specifies only the hierarchy of the parts and the type of kinematic pair they form; the kinematic skeleton, which defines part motion, is inferred from the joint type.
- The system identifies part *interfaces* (regions where the parts can be connected) and determines plausible joint *configurations* (consisting of *position*, *orientation*, and *dimensions*). The user interaction is mostly reduced to choosing among a small number of these configuration options and setting ranges of motion.
- The system creates the geometry of the joints and modifies part geometry when needed.

We describe the user perspective of the workflow in greater detail in Section 5. A number of algorithms are needed for this workflow; most are either independent of the type of joint or are easily adaptable to different joint types. One key algorithm extracts interfaces from a pair of parts (Section 6.1); another determines configurations from the interface geometry (Section 6.2). Finally, in Section 7, we describe our technique for creating joint geometry and updating part geometry to make the object physically realizable.

5. Workflow

The overall workflow is shown in Figure 3. To create a joint in our system, the user starts by selecting two parts of the model and

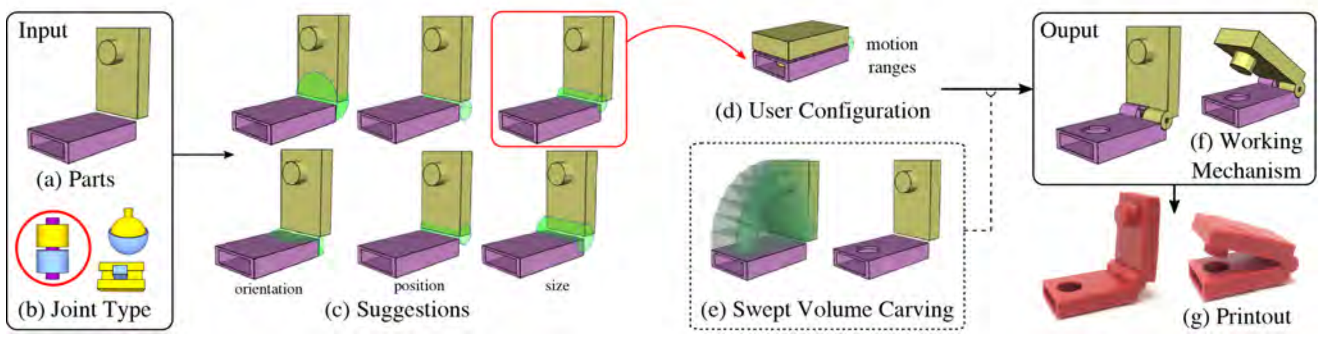

Figure 3: *User workflow.*

specifying the type of mechanical joint that will connect the pair. Based on the selection, our system calculates valid options for interfaces and joint configurations (position, orientation, and dimensions). The user then selects one of the suggested configurations. Using this configuration, our system creates a fabricable version of the design.

Connecting parts. In our system, a part consists of one or multiple combined surfaces that behave as a rigid body. Parts can be connected to each other pairwise. When two parts are connected by a joint, the system automatically adds the needed bones to the skeleton and attaches the parts to it, defining their relative movement. Thanks to the incrementally constructed skeleton, the user can immediately assess the behavior of the mechanism.

Suggestions. For a given kinematic pair, our system automatically computes a set of joint configurations (Figure 3c). Each configuration includes options for position, orientation, and dimensions that control the kinematic behavior and appearance of the joint. Using our interface, the user can iterate over all the configuration options and choose the one that best fits his intended design. When the user changes an option, our system automatically updates the joint and skeleton.

Our interface displays a joint using a simplified version of its geometry (i.e. a *proxy*), which provides a visual representation of position, orientation, and size. We find that using a proxy of the geometry, instead of the full geometry, is better for depicting the kinematic behavior of the joint. For example, for a hinge, we use a cylinder that effectively shows rotation axis, position, radius, and length (see Figure 4a,c).

Controlling motion ranges. Our interface offers the user an option to specify the bounds of motion and visualize the related swept volume (Figure 3e-left). This visualization helps the user decide suitable bounds for the intended mechanism behavior.

After setting the motion bounds, the user can use our system to remove any intersecting geometry (Section 7). This step is optional and serves as an additional tool to our system: a user can decide to execute it at any point during the modeling session and as many times as required. With this tool, the user can easily carve out complex geometries from connected parts. For example, the lock mechanism in Figure 3 needs a hole in one of its parts to fit the cylinder

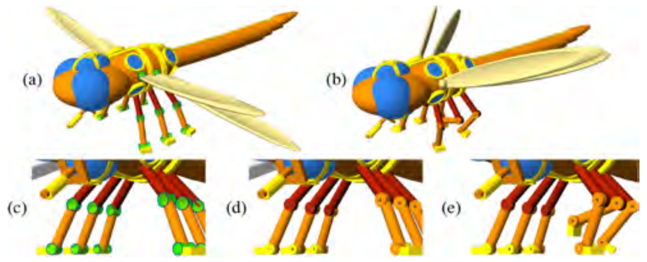

Figure 4: ***Joint Proxies.*** *(a) Dragonfly model with joint proxies in green. (b) Model with actual joint geometry and posed joints. (c-e) Close-up views of the legs. Original model "Dragonfly fighter" by Alexis Zephyrian.*

of the lock. However, the size of the hole is hard to calculate: the cylinder sweeps the volume of a torus with a minor radius equal to the radius of the cylinder, and the major radius depends on the position of the hinge.

As we show in our user studies (Section 8), calculating bounds and swept volumes manually is time-consuming and requires nontrivial modeling proficiency. In contrast, with our tool, even a novice user can create working joints.

6. Algorithms

Our workflow for modeling mechanisms is supported by a set of algorithms that calculate the configuration alternatives offered to the user. These algorithms, with minor variations, are the same for all joint types.

The core of our suggestion algorithm is based on three observations. First, joints are often located in regions proximal to both parts (the parts's *interfaces*). Second, the motion defined by the joint is usually aligned to features of the interface. And third, the overall size of the joint must fit the space defined by these regions. All of these observations are somewhat ambiguous, and typically a number of possibilities exist, with the best choice related to the semantics of the object.

An interface is a region of a part where other components can be connected. In the case of polyhedral parts, it is natural to use polyhedron facets as our elementary unit for interfaces. Following

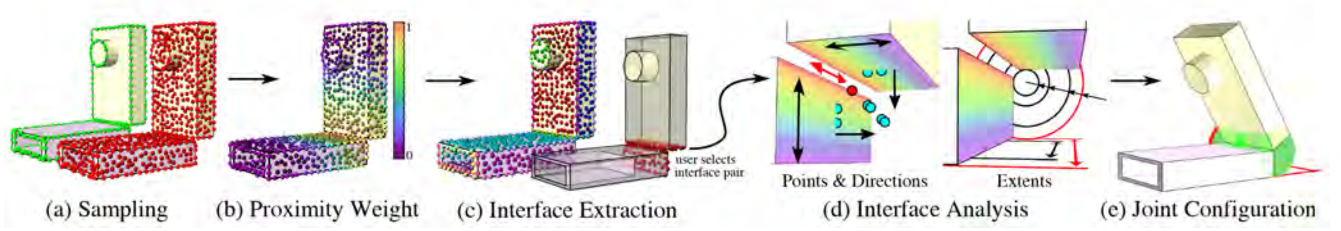

Figure 5: ***Pipeline overview.*** *(a) sharp features samples (green) and surface samples (red); (b) colored proximity weight; (c) partition of interfaces, each with a unique color, and example of an interface pair selected by the user (in red); (d) points, directions, and extents calculated with our algorithm; (e) joint configuration given the point, direction and extents highlighted in red.*

the three observations mentioned above, we can find interface candidates, for example, by identifying faces on both parts that are close to each other, setting the joint orientation with respect to the normals of the facets, and deciding the joint size from the face size. This approach can easily be generalized to parts consisting of smooth patches bounded by sharp curves. However, this approach works poorly for all but the simplest shapes: elements like small details, protruding sharp parts, rounded edges, curvature variation, and position or normal noise all make this simple approach unusable.

Our concept of interfaces generalizes facets. We define an interface as a set of points sampled from mesh triangles and "sharp" edges (cf. [GSMCO09]). Each triangle sample is assigned a normal, and each edge sample a direction. *Intuitively, the interface is a face-like connected subset of the surface of a part, that has a boundary defined by rapid changes of normal direction, and is close to the other part.* The choice of interfaces is described more precisely below.

Pipeline overview. The input to our suggestion algorithm is a pair of parts and a joint type. The algorithm proceeds in several steps:

- The surface and sharp features of each part are sampled.
- Sample sets are partitioned into interface regions (possibly multiple regions per part, as different areas in a pair of parts may be in proximity).
- For a given interface pair, a set of attributes (positions, orientation axes, and extents) that characterize the region is computed.
- Using these attributes, a set of joint configuration alternatives is computed and scored to determine the order of presentation to the user.

Sampling. As input meshes may have highly nonuniform triangle size and shape, we start by resampling the surface uniformly, with additional samples on sharp-edges. Our sampling method is based on the Poisson disk algorithm proposed by [BWWM10] and is extended to edges. This algorithm first computes a large set, S_i, of random points uniformly distributed on the surface. Then, it draws Poisson disk samples from S_i, generating the set $S \subset S_i$. We refer the reader to [BWWM10] for implementation details. Here we only state that we can efficiently draw a Poisson disk sample set S from a sufficiently large S_i.

To compute S_i for a part p, our algorithm first triangulates the mesh and extracts sharp-edges (edges that have sharp dihedral angles or lie on the boundary [GSMCO09]. Then it generates two

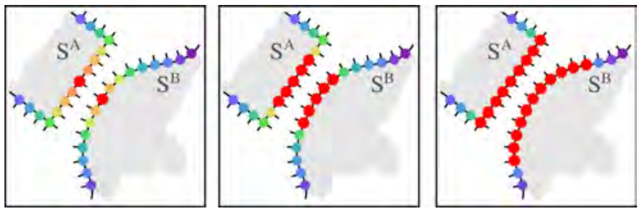

Figure 6: ***Interface extraction.*** *From left to right: iterations 1, 3, and 6 of BFS in each sample set.*

sets of points, S_i^T and S_i^E, drawn from triangles and edges respectively. To generate S_i^T (resp. S_i^E) we repeatedly select a triangle (resp. edge) with probability proportional to its area (resp. length) and then uniformly sample the selected element. For a triangle, the sample point is defined in barycentric coordinates as $u = 1 - \sqrt{\xi_1}, v = \xi_2\sqrt{\xi_1}$, where ξ_1, ξ_2 are two uniform random numbers. Likewise for edges, the point is given by $u = \xi_1$. The union set $S_i = S_i^T \cup S_i^E$ is passed to the Poisson disk algorithm to compute the set S (see Figure 5a).

In addition to its position, each point is assigned a normal direction; for a point drawn from a triangle, we use the surface normal as normal direction; for a point drawn from a sharp-edge we generate two samples per point, each with a different normal (corresponding to the normals of the triangles associated with the sharp-edge). Each edge-point is also assigned an edge-direction, corresponding to the vector defined by the two vertexes of the edge.

Although we sample each part p individually, we use the same Poisson disk radius, so the densities of both sample sets are equivalent. Let a^A, a^B be the surface areas of each part, and N a desired upper bound on the size of the sample sets; then, the sample radius r is given by $r = \sqrt{(a^A + a^B)/(N\pi)}$.

6.1. Interface Extraction

We extract interfaces from a sample set S one by one, using a breadth-first growth for each (see Figure 6). To favor expansion towards regions close the other part, each point is assigned a weight using the proximity weight function defined below. The first interface is initialized with the sample point with highest weight. Starting points for following interfaces are chosen to be the points with highest weight, not yet assigned to any existing interface. Our algorithm iteratively expands the interface to neighboring points that keep the interface relatively flat.

Let us denote the set of points that compose the interface in the

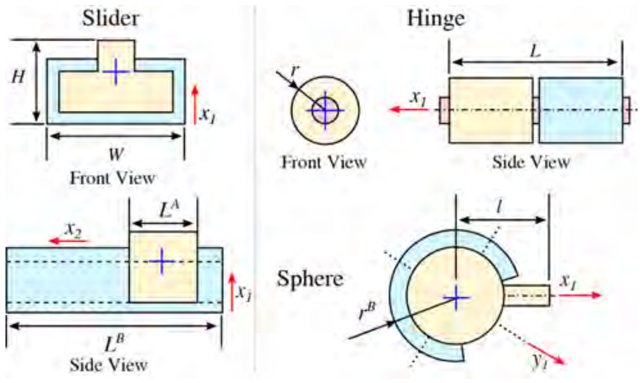

Figure 7: *Schematics for joint models. Our system automatically calculates values for the position (indicated by a blue cross), part orientations (red arrows), and part dimensions. Dimensions not included in this figure are derived from these main dimensions. Such is the case of the radius of the ball in the sphere joint, calculated as a fraction of r^B. Complete schematics of the models are included in the supplementary material.*

k-th iteration by I_k. We define the set of "neighboring points" of that iteration as all the points in $S \setminus I_k$ at a distance less than $2r$ from any point in I_k. To keep the interface planar, our algorithm accepts a point p only if its normal direction is close to the normal of the best-fit-plane defined by I_k. We calculate this normal as the weighted average normal of the points in I_k.

The extraction of interface I ends when no neighboring points are found that satisfy the plane requirement. The algorithm continues extracting new interfaces until all points have been processed and the whole surface is segmented.

Proximity weights. Motivated by the observation that joints are typically located in regions close to both parts, we weight points of a sample set by their distance to the other set. Due to their good localization properties, we choose these weights to have a similar form as weights used in moving-least-squares constructions. For a point $p \in S^A$, its weight with respect to S^B is given by:

$$w_d(p, S^A, S^B) = \exp(-c_1 \|p - q(p, S^B)\|^2 / a_{min}), \quad (1)$$

$$a_{min} = \pi r^2 \min\{|S^A|, |S^B|\} \quad (2)$$

where $q(p,S)$ corresponds to the closest point on set S, and r is the sampling radius. The parameter c_1 controls the falloff of the weight function. All the examples in this paper use $c_1 = 100$.

To accelerate the subsequent stages, we remove most of the resulting interfaces for each pair of parts from consideration, keeping only the ones that have significant weight and size: given an interface set $I \in S$ we use a threshold on the total weight of its points (10% of the total weight of points in S) and a size threshold (1% of the area of S).

6.2. Computing Positions, Axes and Extents

For a pair of interfaces, our algorithm computes a set of positions, axes, and extents that are used to configure the parameters of the joint (see schematics in Figure 7). The sets of points, axes, and extents we use are based on observations of a gallery of joints used in current manufacturing (see examples in Figure 8).

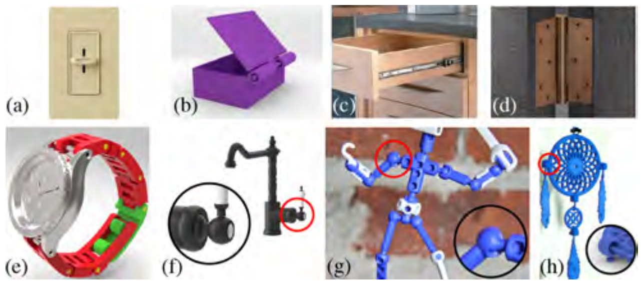

Figure 8: *Gallery of mechanisms from online repositories: (a) light switch with slider joint, (b) box with hinge joint, (c) drawer with sliding joint, (d) door hinge joint, (e) watch band with multiple hinge joints, (f) faucet with sphere joint, (g) toy, ModiBot, with multiple sphere joints, and (h) dream-catcher with sphere joints.*

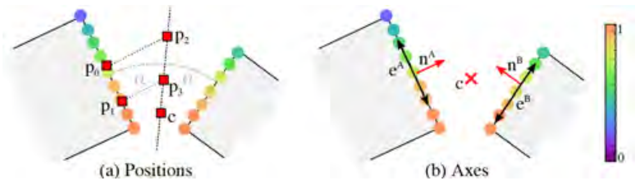

Figure 9: *Positions and axes calculated by our system.*

Positions. A position is a possible location of the center of a joint. We extract two types of positions, *surface* points and *bisecting* points, according to their distance to the interface planes. Surface points lie on the plane defined by the interface and are used to position the joint inside a part (see the slider joint in Figure 8a). Bisecting points are equidistant to both interfaces and are particularly relevant to symmetric joint models (see hinge joint in Figure 8d).

We compute the following alternatives for positions (Figure 9):

- Points p_0 and p_1 lie on the interface and correspond to its unweighted and weighted center of mass respectively (eg. Figure 8a,c,f,h).
- Points p_2 and p_3 lie in the bisecting plane and correspond to the intersection of the plane and the line defined by p_0 (resp. p_1) and the interface normal (eg. Figure 8e,g).
- Point c corresponds to the weighted center of mass of both interfaces and usually lies on the bisecting plane (eg. Figure 8b,d).

Axes. An axis is used to orient a natural body-aligned axis of a joint (e.g., the rotation axis of a hinge, or the slider direction). An axis can be *tangent* or *perpendicular* to the interface average plane. For example, in the slider model in Figure 7, the axis x_1 is perpendicular to the interface while x_2 is tangent.

We use the following directions for a given interface pair:

- Best-fit-plane normal of each interface: for axes perpendicular to an interface (e.g., sphere joints on Figure 8f,g, and axis x_1 of slider joints on Figure 8a,c));
- Cross product between best-fit-plane normals: for axes tangent to both interfaces. This is particularly useful when the initial pose is already rotated, as it indicates the preferred rotational axis (e.g., Figure 8 b,d,e,g).
- Sharp-edge directions: for axes tangent to one of the interfaces (e.g., travel axis x_2 for slider joints on Figure 8a,c).

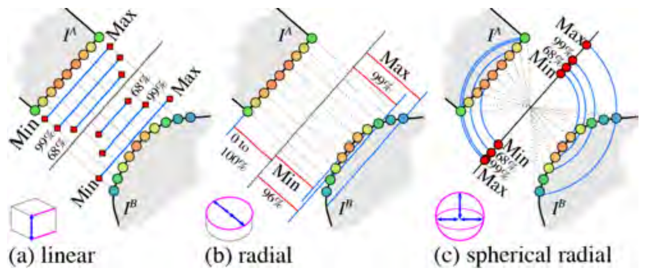

Figure 10: ***Extents Measure.*** *Interface I^A represents an interface with a sharp bound, and I^B one with soft bounds. For a given extent, distances are measured (a) along a line, (b) to a line, or (c) to a point, and used to obtain representative values for the extent.*

To compute salient sharp-edge directions, we use histogram-based clusters of directions. Using the edge direction, our method projects all sample edge points to a 3D cube grid. It then picks two cells (bins) with the highest total weights. When no relevant bins are found, it uses PCA to compute two principal directions for the interface. We find this approach adequate, since we need only a relatively coarse resolution.

Extents. An extent is a range of distances given by a lower and upper bound. We use the following types of extents (see Figure 10): linear, radial and spherical radial. For example, models for sliding joints include mostly linear extents (e.g., Figure 8a,c); hinges include linear and radial extents (Figure 8b,d,e); and ball-and-sockets include mostly spherical radial extents (e.g., Figure 8f,g,h).

A *linear* extent is associated with an axis a and a position p, and is obtained by projecting all interface points to a line l in direction a passing through p and measuring the distances to p.

A *radial* extent is also associated with an (axis, position) pair, but the distances from sample points to the line l are measured.

A *spherical radial* extent is obtained from distances from sample points to a position p.

For each set of distances, we compute the ranges that cover the 68%, 95% and 99% of the interface total weight, with distances weighted by the weight of corresponding points. We also obtain the extreme points, min/max, along the line.

Interfaces in which weights are similar (see I^A in Figure 10), have very similar 99% and maximum ranges. In contrast, interfaces where the weights decay slowly (e.g., I^B) present a significant difference between the 99% range and the extreme options. In the first case, the extreme range often offers a better solution, working as a "snap to border" option. In the second case, the extreme range is not representative of the interface (having a low weight). To offer appropriate solutions to the user, our algorithm evaluates the difference between the 99% range and the extreme range: if the difference is smaller than twice the sampling radius, only the extreme options are offered. Otherwise, the percentage options are used.

The values calculated are used to modify the parameters of the joint's model (Figure 7). In most cases, the value of the parameter is calculated directly from a single extent. However, some parameters are affected by multiple extents. Such is the case of the number of knuckles in the hinge joint, as shown in Figure 11. Here the number of knuckles is an integer, calculated as a relationship between the length and radius.

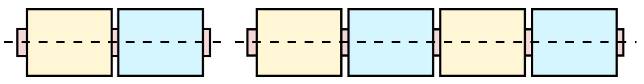

Figure 11: ***Indirect parameters.*** *In the hinge model, the number of knuckles is calculated as a factor of the ratio between the length and the radius of the joint.*

We show the complete parametric models and the relationships between the attributes in the supplementary material.

To add new models into our system, it is sufficient to specify how to calculate the parameters in terms of our main dimensions (see Figure 7). For models with different main dimensions, each dimension needs only to be classified as linear, radial, or spherical radial along a pair of position/axes.

6.3. Computing Configuration Alternatives

When the user selects a pair of parts, our system automatically calculates a set of interface pairs (Section 6.1) and alternatives for positioning, orienting and sizing the joint (Section 6.2). These alternatives are intuitive to the user, as they have a direct relation to the common 3D manipulation tasks of translating, rotating and scaling. Before the options are presented to the user, each set of alternatives is sorted and the option with highest score is used as default. While position and axis choices are only dependent on the interface pair, the extents depend on both and so they need to be computed separately for each (axis,position) combination.

Interface pairs are ranked by the proximity between their points. For a pair (I^A, I^B), the weight is given by:

$$w_I(I^A, I^B) = \frac{1}{|I^A||I^B|} \sum_{p \in I^A} w_d(p, I^A, I^B) \sum_{p \in I^B} w_d(p, I^B, I^A) \quad (3)$$

where w_d is defined as in Equation 1.

Similarly, position alternatives are ranked by their proximity to both interfaces, with the weight of a position p given by:

$$w_P(p, I^A, I^B) = w_d(p, I^A, I^B) w_d(p, I^B, I^A), \quad (4)$$

Positions originating from different interfaces may be close; we consider a position p to be a duplicate if there exists a position q such that the distance between them is less than a tenth of the sampling radius. Only the position with the highest weight is retained.

Axis alternatives of different types are sorted differently: for tangent axes, we offer first the cross product (if any) and then the sharp edge directions; for perpendicular axes, we offer the normal directions. Axes with angular difference of less than 0.1 radians are viewed as identical.

Extent alternatives are sorted by length, from shortest to longest. We consider a extent $[t_1, t_2]$ to be a duplicate if there is another extent $[l_1, l_2]$ whose difference between corresponding endpoints is less than a tenth of the sampling radius. Duplicates are handled by choosing arbitrarily between the options.

7. Generating Physically Realizable Geometry

Once the user is satisfied with the configuration of the joint, our system creates its geometry and connects it to the mechanism. This involves three steps: generating working joint geometry, adding gaps between the parts, and computing the swept volume to ensure the parts can move.

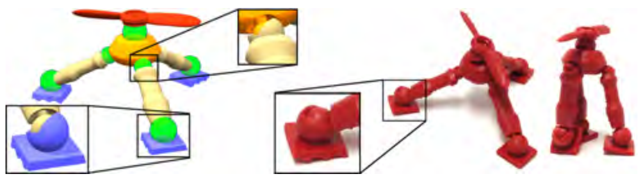

Figure 12: ***Printable Geometry.*** *Propeller Head model. Geometry is carved out of the parts to allow joint motion.*

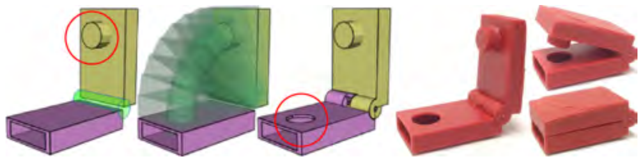

Figure 13: *Lock model. Our system automatically carves the purple part to fit the cylinder on the yellow part.*

Generating working and printable geometry. Before connecting the geometry to the mechanism, it is necessary to remove any volume that interferes with the pair's relative motion (see Figure 12). Let us denote the joint geometry that will be attached to part A as J^A (resp. J^B for part B). To connect J_A to A, our algorithm first computes the volume V^B traversed by J_B (see below) and carves it off A using Boolean difference. Finally, J_A is connected using Boolean sum. The same operation is done for B, carving off V_A and adding J_B, with V_A computed for the original J_A. Our algorithm adds a separation between the parts (clearance) by inflating V_A and V_B before carving. An appropriate clearance value depends on the printer's precision and is therefore provided by the user.

Given the arbitrary shape of the parts, some remaining geometry, outside the vicinity of the joint, might still interfere with the part's movement (e.g., the highlighted cylinder in Figure 13). Because our tool is meant to be used as part of the modeling process we do not automatically remove this geometry, but allow the user to run a sweep analysis at any time during modeling and selectively remove unneeded regions.

Swept Volume Computation. Several generic algorithms exist to obtain a swept volume generated by arbitrary transformations of a mesh over time. We found that the existing code available to us to compute swept volumes was not robust and was therefore unsuitable for our system. Thus, we implemented our own simplified algorithm similar to the method described in [PPSZ05]. Their method finds characteristic or silhouette points in the direction of motion at each time step and sweeps them in that direction at the step intervals, creating a mesh from this set of boundary points. We tailor this method for the simpler, constant-velocity trajectories which mechanical objects typically create, e.g. lines, circular arcs, and helixes. As in most swept volume algorithms, the most computationally demanding aspect of our approach is the use of robust Boolean operations [ZGZJ16] to identify the inside and outside of the shapes.

We explain our approach using the case of linear one-dimension translation. The algorithm first partitions the mesh of the moving part along the silhouette in the direction of the translational sweep, separating it into regions of front-facing and back-facing components (components whose vertex normals have a positive or negative dot product with the direction of the sweep). Then, for each separate patch of connected front-facing or back-facing components, we construct the swept volume: two copies of the patch are created and then translated respectively to the starting and ending position of the sweep, and the corresponding boundary vertices are connected. If we take the Boolean union of all patch sweeps and the original shape, we can create the entire swept volume. However, for efficiency, we consider only those patches whose sweeps intersect with the static part. Note that if the starting shape is manifold, then this method, by construction, produces closed manifold meshes.

This method can be used for other types of one-dimensional parameterized motion using a simple change in coordinate system. For example, for the hinge, we remap the circular rotation to a simple linear translation, mapping the Cartesian coordinates to polar coordinates relative to the rotation axis. The sweep is created using the same process described above, with the addition that, in order to create a smooth curve, multiple vertices are added to interpolate between the boundary vertices of each pair of swept patches. Robust boolean and mesh operations are used to find the union of the patches, resolve self-intersections, and compute an outer hull of the resulting mesh. For more complex types of motion with multiple independent degrees of freedom, such as the motion allowed by a ball-and-socket joint, the sweep must be computed successively for each degree of freedom: the shape is first swept around one axis, the resulting shape is swept around the next axis, and that result is swept along the final axis.

8. Discussion and Results

To evaluate our system, we conducted pilot experiments with novice users and applied our system to create mechanical objects of varying complexity. In order to facilitate the evaluation, we implemented our system as a plug-in for Maya 2015 — which we will release online to support and foster research in this area.

8.1. Pilot Experiments

We conducted the experiments on a MacBook Pro equipped with a 15-inch screen. The test group consisted of eight subjects: four Computer Science students and four architects. None of them had extensive experience with graphics modeling software. Half of them reported having low to no experience with 3D modeling and rigging.

Task I. To validate our suggestion-based approach, we asked users to connect the Lamp model (Figure 14) using our system and to repeat the task using standard modeling tools present in Maya. For the standard workflow, we provided participants with parametric models of the joint's geometry.

When using our system, the user would choose one of the offered suggestions for position, orientation and dimensions of the joints, with the possibility of minor adjustments of joint parameters at the end. In the standard modeling trial, the user was free to use standard tools for translating, rotating and sizing the parametric model and skeleton. In both cases, participants were allowed to change the camera viewpoint. The trial was considered completed when the user determined that the model and skeleton of each joint "looked correct." Before the test, participants were briefed on the

Manipulation	Standard	Option-based	Speedup
Translation	185.1s	36.4s	5.1 x
Rotation	161.3s	25.5s	6.3 x
Scaling	138.5s	85.1s	1.6 x
Camera	613.3s	196.5s	3.1 x
Edit Time	484.9s	146.9s	3.3 x
Total Time	1099.3s	384.0s	2.9 x

Table 1: *Average performance on Task 1, for standard and option-based systems.*

kinematics of the joints, and they practiced with both systems until they felt comfortable with the interface.

To compare the performance of the two systems, we measure the total completion time and the edit time. Edit time considers the time spent modifying the configuration of the joint: choosing one of the alternatives in our system, and translating, rotating and scaling in the standard system. Total completion time includes the additional time spent manipulating the viewpoint. For usability comparison, we use the System Usability Scale (SUS) questionnaire [BKM09].

Results. The results show that with our system users took considerable less time to finish the task, with a speedup of 3.3 times on edit time and 2.9 times on total time (see Table 1). The improvement derives not only from the time spent directly configuring the joint but also from the time spent manipulating the camera (speedup of 3.1 x). Camera changes during the standard trial were common when the user switched between editing the joint and checking the overall aspect of the mechanism. For example, the user would zoom in to change the length of the hinge and then zoom out and orbit to see how it looked. This behavior was frequent in the standard trial and subdued in the option-based trial. We conjecture this is due to needing few different viewpoints to evaluate our options. Additionally, changes of previous decisions were more common for the standard system, most likely due to errors discovered while inspecting the joint from a new viewpoint. This data supports our belief that the standard pipeline needs multiple iterations to solve issues, while our option-based approach solves them in fewer steps. Furthermore, it is important to point out that most of the resulting

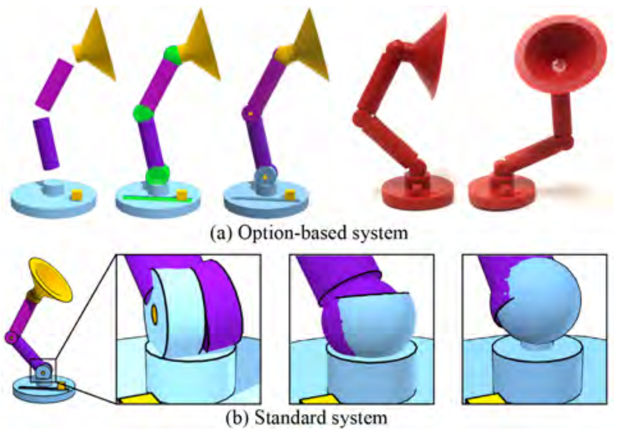

Figure 14: ***Task I.*** *Lamp model (a) Result using our option-based system. (b) Results using traditional modeling tools.*

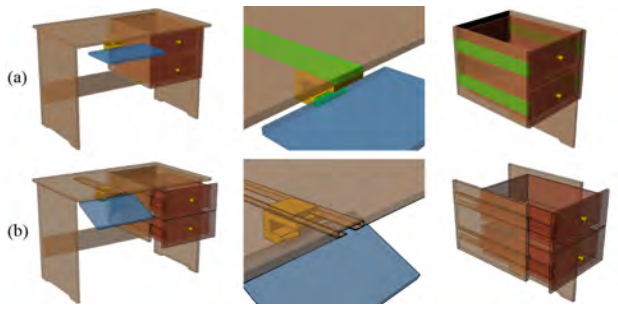

Figure 15: ***Task II.*** *(a) Desk model and close-up with joint proxies; (b) Results obtained with our option-based system. Model derived from "Wooden Desk" by Carlos Folch.*

joints modeled with the standard system had mismatching geometry and skeleton (see Figure 14b), which could cause errors during animation (parts separate or intersect) and unexpected motion or collisions in fabricated models.

On the usability questionnaire, the option-based system has an average score of 86.9 pts on a scale of 0 to 100 pts. According to [BKM09], this score translates to an adjective rating between Excellent and Best Imaginable (B in grade scale). In contrast, the standard system, with an average of 37.8 pts, is qualified as Poor (F in grade scale).

Task II. This was an informal study to retrieve more feedback on the perceived accuracy of our options. Participants were asked to add joints to the Desk model (Figure 15) using our option-based system.

Results. The feedback from the users indicated that our system offers good solutions for position, orientation and size. All subjects agreed that they found alternatives similar to what they wanted and that the final model looks close to what they had envisioned. They also agreed that most of the solutions offered made sense and that they were likely to check all possible options.

8.2. Mechanical Objects

Figures 1,4,12-19 show a range of mechanical objects of varying complexity with joints modeled using our system. Some included a large number of joints and long chains (e.g., Figure 1 and 16a), others contained faces with noise, small details, or intersections (e.g., Figures 4,16b,c), and others had relatively simple geometry with few joints (e.g., Figures 14a,15,19). The models in Figures 12,13,14,17,18,19 were created from scratch by novice users with inspiration from real-life mechanisms. The rest were obtained from online stores, like Shapeways and Turbosquid.

For each object, we used our system to generate, adjust, and implement joint geometry. In cases where the model included non-operational joints (Figure 16), we removed that geometry beforehand. Using our system, we were able to create joints that look similar to the original design. For the Retro Robot model, we manually adjusted the shoulders joint (moved the joints up, and extend their length). For models with loops (Figures 17,18), it was necessary to break the rig, as Maya's rig system doesn't support loops.

To demonstrate that the results are physically realizable models,

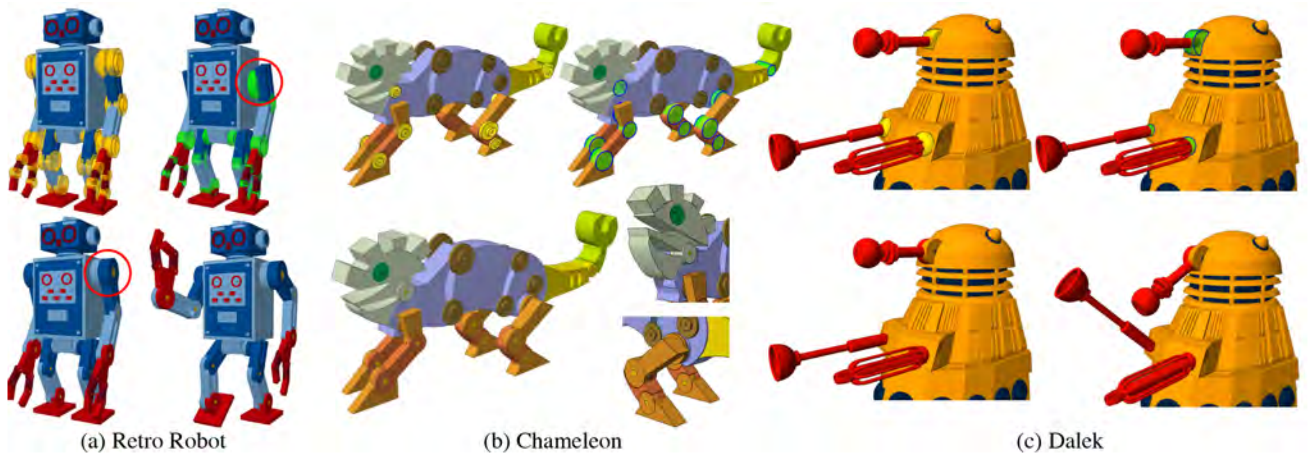

Figure 16: *Models with non-operational joints, which are shown in yellow in the first image of each set. Using our system, we were able to create similar-looking joints. Original models (a) "Retro Robot 1" by Forpost D6, (b) "Dino-Robot" by Ms. McClure, and (c) "Dalek" by squidinc3d.*

we fabricated six of the mechanisms using a B9Creator DLP/SLA printer with Cherry Red resin, and post-assembled the models. The printing process tends to cause inflation, and although our models included a clearance between the parts, the inflation eliminated it. Therefore, the fabricated parts are in contact, creating friction. To control the degree of friction, we altered the models' clearance values: 0.2mm (high friction) to 0.4mm (low friction). For pre-assembly printing, other techniques (eg. [CCA*12]) can be used to provide friction. Incorporating these additional joint models is straightforward: the same parameters that configure our ball-and-socket are used to configure the cage-and-socket (see Figure 20).

Limitations We restricted our implementation to three joint types. However, the same approach can be applied to other kinematic pairs or even higher pairs such as gears.

The major limitation of our system is that joint configurations are calculated using attributes of only two interfaces. While this is sufficient for many joints, artists sometimes want to match *global* attributes of the mechanism. For example, in the Dalek model, an artist might want to force all joints to have the same radius. Our current system can certainly be extended to handle attributes extracted from other regions, but this would require new shape analysis methods.

The heuristics built into the system are only as powerful as the detected interfaces. To properly detect features, a suitable sampling radius r must be used: details smaller than r are overlooked, while larger noise might cause an interface to separate. Also, since we sample only the surface, we might fail to detect proper interfaces on parts with large intersection volumes. In such cases, user intervention might be required to modify or reposition the parts. Finally,

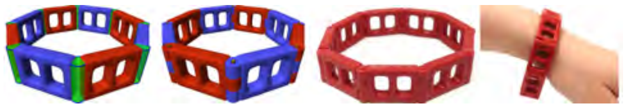

Figure 17: *Bracelet model. (left) Model with joint proxies; (center) Model with joint geometry; (right) Fabricated mechanisms.*

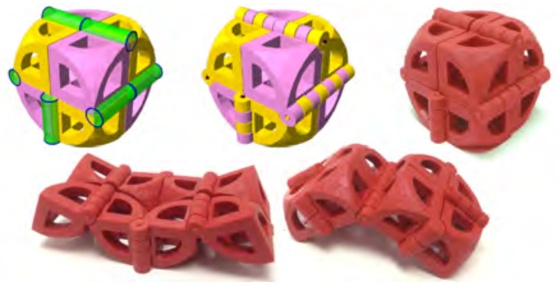

Figure 18: *Magic Cube model. (left) Model with joint proxies; (center) Model with joint geometry; (right-bottom) Fabricated mechanisms.*

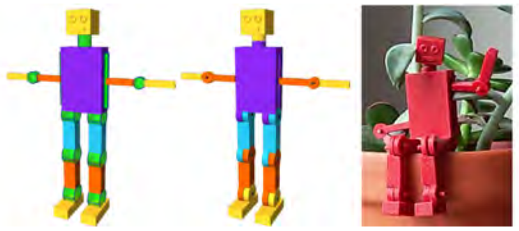

Figure 19: *Tiny Robot model. (left) Model with joint proxies; (center) Model with joint geometry; (right) Fabricated mechanisms.*

our algorithm requires parts to be placed in an initial pose that is allowed by the desired relative motion. Integrating our system with tools that assist in this task (e.g., [ATF12], Tinkercad, Gravity Sketch) is left as future work.

Last but not least, we do not take into account physical properties of produced models, only kinematics. Accounting for physical effects is important for fabrication applications. One approach for supporting physical effects would be to combine our method with stability and strength evaluation [PWLSH13, ZPZ13, BWBSH14, LSZ*14] to find, for example, structurally optimal choices in the parametric space of joint positions and dimensions.

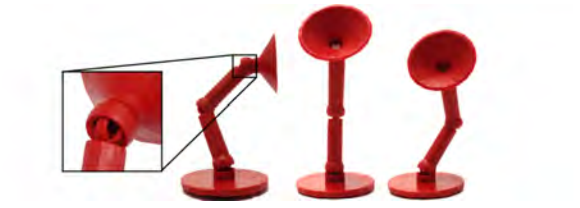

Figure 20: *Lamp model printed using joint model from [CCA*12].*

9. Conclusion

We have presented a system that allows inexperienced users to create articulated, physically correct objects easily. The basic approach of our system, presenting a small set of options to the user for each operation, proved to work quite well, either providing an immediate satisfactory answer or a close approximation point which could, with ease, be adjusted to get the final shape. Our system is based on a set of carefully selected heuristics that do not require manual or automatic model segmentation, except for partitioning the model into moving parts. The underlying computations are fairly simple and do not require an extensive database of sample objects or complex crowdsourcing and learning, yet they yield, in all cases we have tried, small sets of choices including several semantically natural ones.

Acknowledgements We thank David Benjamin (The Living, Autodesk Studio) and his team for helping with user testing, and the anonymous reviewers for their useful comments. This work was partially supported by the Rudin Foundation and NSF award DMS-1436591.

References

[AMYBJ06] ABDEL-MALEK K., YANG J., BLACKMORE D., JOY K.: Swept volumes: Fundation, perspectives, and applications. *International Journal of Shape Modeling 12*, 1 (2006), 87–127. 3

[ATF12] AU O. K.-C., TAI C.-L., FU H.: Multitouch gestures for constrained transformation of 3d objects. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 651–660. 10

[BBJP12] BÄCHER M., BICKEL B., JAMES D. L., PFISTER H.: Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 47. 2

[BCT15] BÄCHER M., COROS S., THOMASZEWSKI B.: Linkedit: interactive linkage editing using symbolic kinematics. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 99. 2

[BKM09] BANGOR A., KORTUM P., MILLER J.: Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies 4*, 3 (2009), 114–123. 9

[BWBSH14] BÄCHER M., WHITING E., BICKEL B., SORKINE-HORNUNG O.: Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 96. 10

[BWWM10] BOWERS J., WANG R., WEI L.-Y., MALETZ D.: Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 166. 5

[CCA*12] CALÌ J., CALIAN D. A., AMATI C., KLEINBERGER R., STEED A., KAUTZ J., WEYRICH T.: 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 130. 2, 10, 11

[CTN*13] COROS S., THOMASZEWSKI B., NORIS G., SUEDA S., FORBERG M., SUMNER R. W., MATUSIK W., BICKEL B.: Computational design of mechanical characters. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 83. 2

[DH64] DENAVIT J., HARTENBERG R. S.: *Kinematic synthesis of linkages*. McGraw-Hill, 1964. 3

[GSMCO09] GAL R., SORKINE O., MITRA N. J., COHEN-OR D.: iwires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (TOG) 28*, 3 (2009), 33. 5

[HL15] HERGEL J., LEFEBVRE S.: 3d fabrication of 2d mechanisms. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 229–238. 2

[KLY*14] KOO B., LI W., YAO J., AGRAWALA M., MITRA N.: Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics 33*, 6 (2014). 2

[KSS*15] KOYAMA Y., SUEDA S., STEINHARDT E., IGARASHI T., SHAMIR A., MATUSIK W.: Autoconnect: computational design of 3d-printable connectors. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 231. 2

[LHAZ15] LI H., HU R., ALHASHIM I., ZHANG H.: Foldabilizing furniture. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 90. 3

[LOMI11] LAU M., OHGAWARA A., MITANI J., IGARASHI T.: Converting 3d furniture models to fabricatable parts and connectors. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 85. 3

[LSZ*14] LU L., SHARF A., ZHAO H., WEI Y., FAN Q., CHEN X., SAVOYE Y., TU C., COHEN-OR D., CHEN B.: Build-to-last: Strength to weight 3d printed objects. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 97. 10

[MGD94] MALLIK A. K., GHOSH A., DITTRICH G.: *Kinematic analysis and synthesis of mechanisms*. CRC Press, 1994. 2

[MTN*15] MEGARO V., THOMASZEWSKI B., NITTI M., HILLIGES O., GROSS M., COROS S.: Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 216. 2

[MYY*10] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 58. 3

[PPSZ05] PETERNELL M., POTTMANN H., STEINER T., ZHAO H.: Swept volumes. *Computer-Aided Design and Applications 2*, 5 (2005), 599–608. 3, 8

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make it stand: balancing shapes for 3d fabrication. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 81. 10

[SSL*14] SCHULZ A., SHAMIR A., LEVIN D. I., SITTHI-AMORN P., MATUSIK W.: Design and fabrication by example. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 62. 3

[TCG*14] THOMASZEWSKI B., COROS S., GAUGE D., MEGARO V., GRINSPUN E., GROSS M.: Computational design of linkage-based characters. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 64. 2

[UIM12] UMETANI N., IGARASHI T., MITRA N. J.: Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4 (2012), 86. 3

[vDH11] VON DZIEGIELEWSKI A., HEMMER M.: High quality surface mesh generation for swept volumes. In *EuroCG Workshop on Computational Geometry* (2011). 3

[XWY*09] XU W., WANG J., YIN K., ZHOU K., VAN DE PANNE M., CHEN F., GUO B.: Joint-aware manipulation of deformable models. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 35. 3

[ZGZJ16] ZHOU Q., GRINSPUN E., ZORIN D., JACOBSON A.: Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG) 35*, 4 (2016). 8

[ZPZ13] ZHOU Q., PANETTA J., ZORIN D.: Worst-case structural analysis. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 137. 10

[ZXS*12] ZHU L., XU W., SNYDER J., LIU Y., WANG G., GUO B.: Motion-guided mechanical toy modeling. *ACM Trans. Graph. 31*, 6 (2012), 127. 2