Creating Works-Like Prototypes of Mechanical Objects

Bongjin Koo
Univ. College London

Wilmot Li
Adobe Research

JiaXian Yao
UC Berkeley

Maneesh Agrawala
UC Berkeley

Niloy J. Mitra
University College London

Abstract

Designers often create physical *works-like* prototypes early in the product development cycle to explore possible mechanical architectures for a design. Yet, creating functional prototypes requires time and expertise, which discourages rapid design iterations. Designers must carefully specify part and joint parameters to ensure that parts move and fit and together in the intended manner. We present an interactive system that streamlines the process by allowing users to annotate rough 3D models with high-level functional relationships (e.g., part A fits inside part B). Based on these relationships, our system optimizes the model geometry to produce a working design. We demonstrate the versatility of our system by using it to design a variety of works-like prototypes.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms.

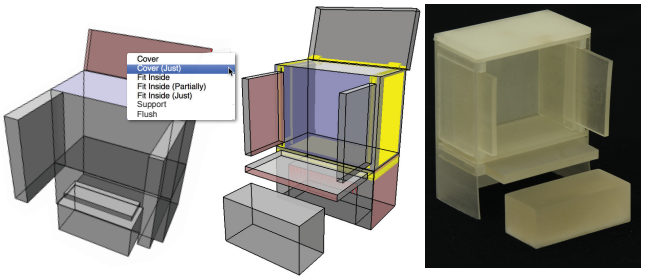
Keywords: fabrication, 3D printing, sketch-based modeling

Links: DL PDF WEB VIDEO

1 Introduction

Creating physical prototypes is an integral part of the product development process that helps designers evaluate and refine potential designs, explore multiple approaches in parallel, and communicate designs to others [Kelley 2001; Eissen and Steur 2009; Hallgrimsson 2012]. Designers create different prototypes for different purposes. Early in the design cycle, they often create *works-like* prototypes like the one shown above that embody the functional aspects of a design. Such prototypes typically contain working mechanical joints but simplified part geometry so that designers can focus on the mechanical "architecture" (i.e., how parts move and fit together) of the product. Later in the cycle, designers may create *looks-like* prototypes that convey the detailed shape and/or colour-material-finish (CMF) of a design. Such prototypes help designers (and clients) understand the intended appearance of the product.

In this work, we focus on the task of creating works-like prototypes. Designers are increasingly turning to 3D printing as a tool for fabricating physical prototypes with mechanical joints that approximate the intended functionality of a design. Yet, creating working mechanisms requires expertise and time [Ulrich and Eppinger 2007; Hallgrimsson 2012]. Designers must carefully specify part proportions and joint parameters to ensure that all moving parts fit together in the intended manner without interference. This task often involves non-trivial geometric calculations, even when the parts are composed of simple primitives (e.g., cuboids). Moreover, designers almost always iterate and refine prototypes by modifying certain parameters (e.g., size of a part, joint types), which requires updating the part and joint parameters to ensure a working prototype. Due to these challenges, designers often have to work with skilled CAD engineers to help them create physical works-like prototypes. This added friction in the design-prototype-evaluate cycle significantly limits the number of iterations designers can make and often leads to shallow exploration of the design space.



(a) Rough model (b) Optimized model (c) Works-Like Prototype

Figure 1: Creating works-like prototypes. *Users start by creating a rough 3D model of a design and then specifying the desired functional relationships between parts (a). Our system optimizes part and joint parameters to generate a working model (b) that can be fabricated as a physical prototype (c).*

To address this problem, we present an interactive system that helps designers create functioning works-like prototypes. The user starts by creating a rough 3D model of all the parts in the design and specifying the types of joints that connect the parts. Then, instead of tweaking part and joint parameters to produce a working model, our system allows users to directly specify high-level *functional relationships* between parts (e.g., part A fits inside part B, parts C and D support part E, etc.). Based on these relationships, our system automatically adjusts part proportions and joint parameters to produce a physically realizable working model (Figure 1c). To aid in design exploration, the system propagates edits (e.g., users may add/remove parts or modify part proportions) throughout the design to ensure that all the specified functional relationships are preserved. By allowing users to work primarily at the level of functional relationships rather than low-level joint and part geometry, our system helps designers create new prototypes more quickly and experiment with variations of existing designs.

We used our system to fabricate works-like prototypes for a variety of objects, including articulated devices (folding tablet), appliances (printer), and furniture (kiosk cabinet, sofa bunk). Figure 1 shows the printed prototype of a cabinet design generated by our system, and Figure 10 shows all of our examples with the moving parts in different configurations. It took less than 10 minutes of interaction time in our system to create working 3D models for these results.

2 Related Work

There is an extensive body of literature on product design theory and practice that discusses the benefits of physical prototyping [Kelley 2001; Ulrich and Eppinger 2007; Eissen and Steur 2009; Hallgrimsson 2012]. While this material offers important motivation for our goal of helping designers create works-like prototypes, we focus here on related work that proposes computational tools for analyzing product designs and generating physical objects.

Analyzing designs. There has been a significant amount of work that leverages sketches to produce a 3D representation of a design. Bae et al. [2008] and Schmidt et al. [2009] propose methods for authoring 3D curves based on gestures and standard techniques for constructing perspective drawings, while Shao et al. [2012] and Xu et al. [2014] extract 3D information from existing sketches by leveraging drawing conventions. However, these techniques focus on creating static 3D geometry, whereas we aim to produce models with moving parts. In this respect, the most relevant previous work is by Shao et al. [2013], who develop a system for creating interactive 3D models from a set of concept sketches that depict an object from different viewpoints and with different configurations of moving parts. However, their approach does not produce a single consistent 3D model, which is necessary for creating a physical realization of the design.

Tools for digital fabrication. With the recent advances in digital fabrication, researchers have started to explore methods for transforming 3D models into physically realizable objects. Some of these techniques optimize properties such as stability [Umetani et al. 2012], robustness [Stava et al. 2012], and the ability to balance [Prévost et al. 2013]. However, these methods focus on the design of static objects. In the domain of moving objects, Bacher et al. [2012] and Cali et al. [2012] propose techniques for adding printable mechanical joints to 3D models, while other efforts explore the design of linkage and gear mechanisms to drive toys or automata [Zhu et al. 2012; Coros et al. 2013; Ceylan et al. 2013]. Although all of these methods produce working, physically realizable models, they focus on the problem of adding joints, gears or linkages to existing 3D models. In contrast, our work helps designers create working models from scratch, which requires optimizing both joint parameters and part geometry. Of course, commercial CAD packages like AutoCAD and SolidWorks also enable users to create working physical prototypes from scratch. However, these tools require users to manually specify part and joint geometry to produce a working design.

Constraint-based modeling. Our work is also related to the extensive body of constraint-based modeling research in the graphics and CAD communities. In particular, we adopt the same high-level approach as previous graphics research that tries to automatically determine the relevant geometric relationships between parts to enable editing and synthesis of 3D models [Gal et al. 2009; Xu et al. 2009; Bokeloh et al. 2012; Zheng et al. 2012]. Our approach is also similar in spirit to previous mechanical engineering research that proposes declarative methods for specifying the relevant geometric constraints for a given mechanical design [Daniel and Lucas 1997; Yvars 2008]. However, to our knowledge, we are the first to focus specifically on the design of articulated, works-like prototypes, and a key part of our contribution is defining a set of functional relationships that are useful for this design task. We also note that some professional CAD tools include constraint-based modeling features, but they require users to manually specify low-level geometric relationships between part/joint parameters. In contrast, our approach automatically converts high-level user-specified functional relationships into the relevant low-level constraints.

3 Overview

To create a works-like prototype in our system, the user starts by producing a rough 3D model of all the parts in the design and specifying the types of mechanical joints that connect the appropriate parts. The user then specifies the relevant functional relationships between parts. Based on these relationships, our system optimizes the part proportions and joint parameters to create a working version of the design.

Parts. In our system, each part consists of one or more axis-aligned connected cuboids. Since works-like prototypes focus on mechanical functionality rather than detailed appearance, representing parts with sets of cuboids is often sufficient. We provide two part modeling interfaces: (1) users draw 2D boxes from one of three orthogonal viewpoints (top, front, side) and then extrude them into cuboids; (2) users annotate an input concept sketch by clicking on feature points (e.g., corners) to create cuboid parts (as described in Shao et al. [2013]). In addition to modeling solid parts, users can also create cuboid or cylindrical cavities within a part. We refer to the initial positions and orientations of all the extracted proxies as the *base configuration* of the object.

Joints. Our system supports four types of mechanical joints (Figure 2). *Hinges* are attached to the faces of two different parts and enable rotation around an axis, *sliding joints* allow two parts to translate linearly along a sliding vector with respect to one another, *sliding hinges* enable both rotation and translation, and *double pivot joints* allow rotations around two axes separated by a rigid link. To add joints, the user first selects two adjacent parts and then does one of the following: add a hinge by selecting any cuboid edge that touches both parts; add a sliding joint by selecting a cuboid face whose normal defines a sliding vector; add a sliding hinge by selecting both an edge and a face; add a double pivot by clicking points on two coplanar faces (one on each part) that define the pivot positions. We represent the pose of a joint j as $j(\theta)$. For hinges, θ represents the angle between the pair of attached faces (Figure 2a). For sliding joints, θ is the signed offset along the sliding vector between the two connected parts with respect to the base configuration (Figure 2b). For sliding hinges, θ is a tuple that includes both the rotational and translational parameters of the joint (Figure 2c). For double pivots, θ is a tuple that includes the rotations around both pivots (Figure 2d). We write the combined pose of a set of joints $J = (j_1, j_2, \ldots, j_n)$ with corresponding joint parameters $\Theta = (\theta_1, \theta_2, \ldots, \theta_n)$ as $J(\Theta)$.

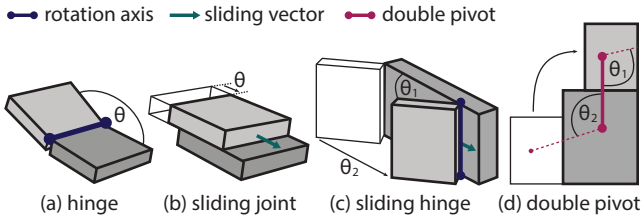


Figure 2: Joints. *We support hinges, sliding joints, sliding hinges, and double pivot joints.*

4 Defining Functional Relationships

To determine the types of functional relationships to include in our system, we examined many product designs and consulted with three professional product designers: a former IDEO employee who now works for Proteus, a startup that makes wearable sensors; and two partners who run Anvil Studios, a Seattle-based product design firm. Based on this formative research, we identified four functional relationships that support a wide range of products with rigid mechanical interactions between parts: cover, fit-inside, support and flush. These relationships impose specific geometric dependencies between the relevant parts. The remainder of this section describes how we formulate these dependencies in terms of constraints on the part dimensions.

4.1 Cover

In many products, certain parts are designed to cover either other parts or cavities. For example, the top half of a clamshell phone must cover the bottom half, and the lid of a container covers its opening. The relationship stipulates that specific faces of the covering part must be the same size or larger than specific faces of the covered part or cavity (Figure 3). While the simplest examples involve a single face covering another single face, in general, cover relationships can involve a set of faces covering another set of faces, based on a *corresponding faces graph* that indicates which subsets of faces correspond. We say that the faces are in their *covered configuration* when all corresponding faces lie in the same plane, the covered faces lie within the covering faces, and there are no gaps or overlaps between the covering or covered faces.

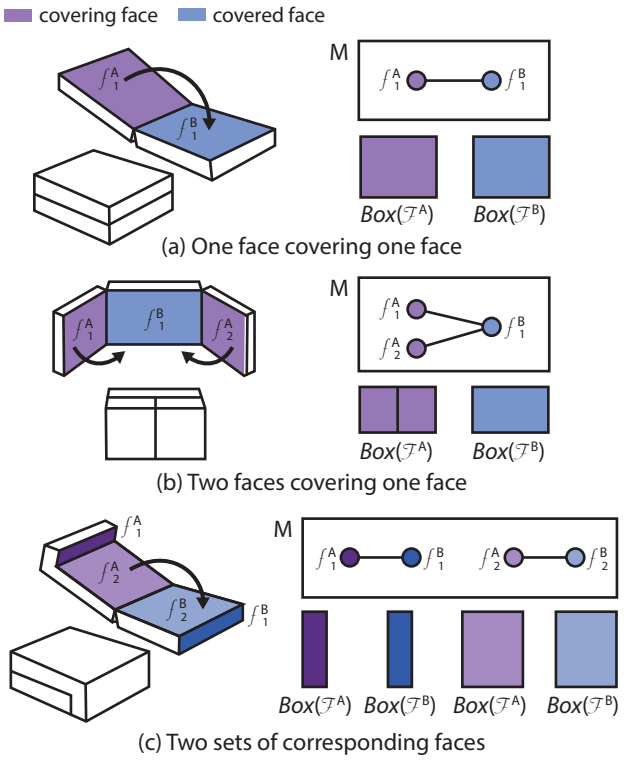


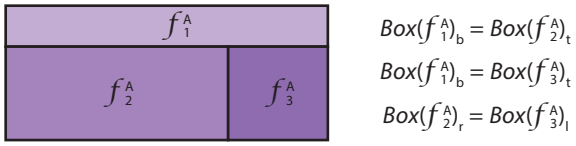
Figure 3: Example cover relationships. *Each example shows the covering faces F^A and covered faces F^B in their base and covered configurations, corresponding faces graph M, and the bounding boxes of the corresponding faces.*

We define a cover relationship as $Cover(F^A, F^B, M, J, \Theta)$, where the set of faces F^A cover the set of faces F^B, M is a corresponding faces graph linking each face in F^A to the corresponding faces in F^B that it must (fully or partially) cover, and $J(\Theta)$ are the set of joints and parameters that put the faces into their covered configuration. Under this definition, each connected component of M contains two sets of faces $\mathcal{F}^A \subseteq F^A$ and $\mathcal{F}^B \subseteq F^B$, and in the covered configuration, these corresponding faces lie in a plane where $\mathcal{F}^A$ covers $\mathcal{F}^B$. Thus, for the cover relationship to hold, these faces must meet the following geometric constraints:

$$\begin{aligned} Box(\mathcal{F}^A)_l &\leq Box(\mathcal{F}^B)_l & Box(\mathcal{F}^A)_r &\geq Box(\mathcal{F}^B)_r \\ Box(\mathcal{F}^A)_b &\leq Box(\mathcal{F}^B)_b & Box(\mathcal{F}^A)_t &\geq Box(\mathcal{F}^B)_t \end{aligned} \quad (1)$$

where $Box(\mathcal{F})$ represents the bounding box of the geometric union of the set of faces $\mathcal{F}$, and $Box(\mathcal{F})_l$, $Box(\mathcal{F})_r$, $Box(\mathcal{F})_b$, $Box(\mathcal{F})_t$ represent the left, right, bottom and top coordinates of the box. The bounding boxes are defined in 2D because $\mathcal{F}^A$ and $\mathcal{F}^B$ are in the covered configuration, which means that the faces all lie in the same plane. To compute the bounding boxes, we define a coordinate system by taking the largest surface $f \in \mathcal{F}^B$, choosing one corner as the origin and using the two incident edges as the x and y axes. Note that changing the inequalities to equalities in the constraints above indicates that the faces in F^A should be large enough to "just cover" the corresponding faces in F^B.

To ensure that there are no gaps or overlaps between each group of covering or covered faces, we impose additional packing constraints that require adjacent faces to touch without overlapping, as in the example below.



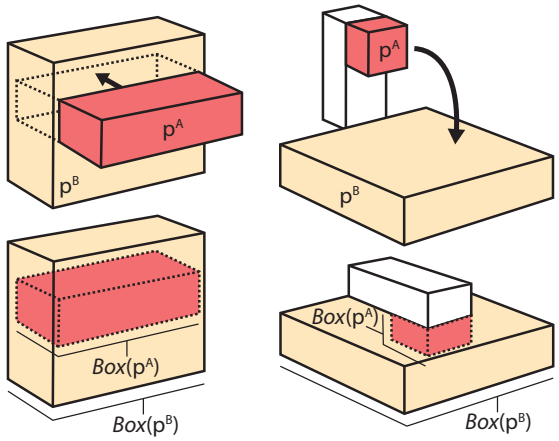
4.2 Fit Inside

Another common functional relationship involves one or more parts fitting inside another (Figure 4). For example, drawers must fit inside the body of a dresser, and a pocket door must fit inside its housing. Some designs include parts that fit partially inside other parts without being completely contained. In some cases, the inside part is designed to be just small enough to fit inside the container part in certain dimensions. We say that the parts are in their *fitting configuration* when the appropriate part fits inside the other.

We define a fit inside relationship as $Fit(P^A, p^B, J, \Theta)$, where the set of parts P^A fit inside part p^B, and $J(\Theta)$ are the set of joints and parameters that put the parts into their fitting configuration. In order for P^A to fit inside p^B, the following geometric constraints must hold:

$$\begin{aligned} Box(P^A)_l &\geq Box(p^B)_l & Box(P^A)_r &\leq Box(p^B)_r \\ Box(P^A)_b &\geq Box(p^B)_b & Box(P^A)_t &\leq Box(p^B)_t \\ Box(P^A)_n &\geq Box(p^B)_n & Box(P^A)_f &\leq Box(p^B)_f \end{aligned} \quad (2)$$

where $Box(P^A)$ is the 3D bounding box of the geometric union of parts P^A, $Box(p^B)$ is the 3D bounding box of p^B, and $Box(p)_l$, $Box(p)_r$, $Box(p)_b$, $Box(p)_t$, $Box(p)_n$, $Box(p)_f$ are the left, right, bottom, top, near and far coordinates of a bounding box. Both

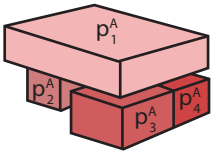


(a) Fit inside (sliding joint) (b) Fit partially inside (hinge)

Figure 4: Example fit inside relationships. *Each example shows the inner part p^A and enclosing part p^B in their base and fitting configurations. Setting p^A to be a portion of a part specifies a "fit partially inside" relationship (b).*

bounding boxes are defined with respect to the coordinate system of p^B and with all parts in their fitting configuration. To represent a relationship where a part $p^A \in P^A$ "fits partially inside" p^B, we set p^A to be the specific portion of the part that should fit inside p^B (Figure 4b). Note that changing any of the inequalities above into an equality constraint indicates that P^A should have enough room to "just fit inside" p^B in one or more dimensions.

Similar to the cover relationship, if there is more than one part in P^A, the parts cannot intersect each other in the fitting configuration. Thus, we apply non-overlapping constraints to adjacent parts, as in the example below.



$$Box(p^A_1)_b \geq Box(p^A_2)_t \quad Box(p^A_3)_f \geq Box(p^A_4)_n$$
$$Box(p^A_1)_b \geq Box(p^A_3)_t \quad Box(p^A_3)_l \geq Box(p^A_2)_r$$
$$Box(p^A_1)_b \geq Box(p^A_4)_t$$

4.3 Support

In some objects, certain parts are designed to support other parts in specific configurations. For example, in a folding table, the legs support the tabletop when the table is opened. In the simplest case, the relationship stipulates that one of the top faces of a supporting part must be in the same *supporting plane* and in contact with one of the bottom faces of the supported part. However, as with cover relationships, the general case involves a set of faces from multiple parts supporting a set of faces on another collection of parts, based on a corresponding faces graph (Figure 5). We say that the faces are in their *supported configuration* when all corresponding faces are in the same plane and in contact. Note that our definition of support does not consider whether the supported part maintains static equilibrium on top of the supporting parts.

We define a support relationship as $Support(F^A, F^B, M, J, \Theta)$, where F^A and F^B are the sets of supporting and supported faces, M is the corresponding faces graph, and $J(\Theta)$ are the set of joints and parameters that put the faces into their supported configuration. This relationship implies the following geometric constraints for each pair of faces $f^A \subseteq F^A$ and $f^B \subseteq F^B$ connected by an edge in M:

$$\begin{array}{ll} f^A_l < f^B_r & f^A_t < f^B_b \\ f^A_r > f^B_l & f^A_b > f^B_t \end{array} \qquad c^A_b = c^B_t \tag{3}$$

where f_l, f_r, f_b,f_t are the left, right, bottom, top coordinates of each face with respect to the supporting plane, c^A, c^B are the parent cuboids of f^A, f^B, and c_b, c_t are the bottom and top coordinates of each cuboid. The four inequality constraints on the left ensure that f^A and f^B overlap in the supporting plane, and the equality constraint on the right ensures that f^A and f^B are at the same height. In addition, the support relationship specifies that the bottom coordinates of all parts with one or more supporting faces but no supported faces must be equal, which ensures that the entire set of supporting/supported parts can sit flat on the ground.

4.4 Flush

Finally, many designs include parts that fit together such that one or more of their faces are flush. For example, folding access panels on the side of a printer are usually flush with the printer body when closed. We say that faces are in their *flush configuration* when they lie in the same plane, and we define the relationship as $Flush(f^A, f^B, J, \Theta)$, where f^A and f^B are the two faces that are flush, and $J(\Theta)$ are the set of joints and parameters that put the faces into their flush configuration. The flush relationship constrains the coordinates of the parent cuboids of f^A and f^B such that the two faces are coplanar in the flush configuration.

5 Specifying Functional Relationships

To help users specify functional relationships for a given design, our system provides interactive tools that automatically infer the appropriate low-level geometric constraints given a small amount of user interaction.

5.1 Specifying Cover Relationships

Users specify a cover relationship by selecting the set of parts that contain the covering faces F^A and covered faces F^B, adjusting joint parameters to move the faces into their covered configuration, and indicating whether they want a regular cover relationship or a "just cover" relationship. The system infers the corresponding faces graph M with a simple greedy approach that considers every candidate covering face f^A and adds an edge to any candidate covered face f^B where f^B and f^A are parallel, separated by less than a small threshold distance, and overlap by more than half the area of the smaller face when both faces are projected onto f^A. The algorithm processes the candidate covering faces in order from largest to smallest and removes candidate covered faces from consideration once they are added to M. If the system infers any incorrect edges in M, the user can click on the appropriate corresponding faces.

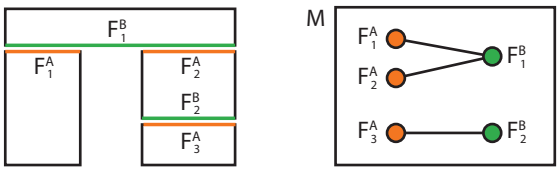


Figure 5: Example support relationship. *A set of supported faces (green) sit on top of their corresponding supporting faces (orange).*

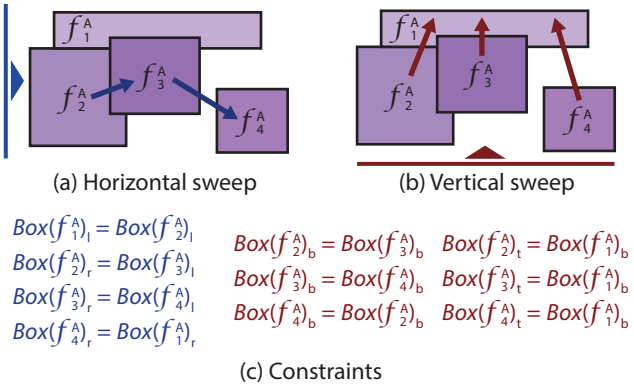


Figure 6: Plane-sweep. *We use a plane-sweep approach to order covering faces horizontally (a) and vertically (b) and then generate constraints between consecutive faces (c).*

Once M has been determined, our system generates the geometric constraints described in Section 4.1 for each set of corresponding faces. To compute all the relevant packing constraints (which eliminate gaps or overlaps between each set of covering or covered faces), we start by determining a spatial ordering over the faces using a plane-sweep approach [Nievergelt and Preparata 1982]. We sweep a vertical line from left to right and compute intersections between the line and face bounding boxes. We keep track of boxes that intersect overlapping segments of the line (ignoring overlaps that are less than a small threshold) to determine a partial ordering of the boxes in the x dimension (Figure 6a). We then sweep a horizontal line from bottom to top to compute a partial ordering in the y direction, ignoring any pair of faces that are already ordered in x (Figure 6b). Finally, for any remaining pair of faces that are not ordered in x or y, we sort them in one of the two dimensions based on their centroid coordinates. Given this ordering, we generate packing constraints between consecutive faces in each dimension, and we also constrain the coordinates of the leftmost, rightmost, bottommost and topmost bounding boxes to be equal, which results in a rectangular packing of all the face bounding boxes (Figure 6c).

Next, we compute the constraints on the 2D bounding boxes $Box(\mathcal{F}^A)$ and $Box(\mathcal{F}^B)$ defined in Equation 1. Since we now have a spatial ordering for each set of covering and covered faces, we can express $Box(\mathcal{F}^A)$ and $Box(\mathcal{F}^B)$ in terms of their constituent faces as follows:

$$Box(\mathcal{F}^A)_l = Box(f^A_l)_l \qquad Box(\mathcal{F}^B)_l = Box(f^B_l)_l$$
$$Box(\mathcal{F}^A)_r = Box(f^A_r)_r \qquad Box(\mathcal{F}^B)_r = Box(f^B_r)_r$$
$$Box(\mathcal{F}^A)_b = Box(f^A_b)_b \qquad Box(\mathcal{F}^B)_b = Box(f^B_b)_b$$
$$Box(\mathcal{F}^A)_t = Box(f^A_t)_t \qquad Box(\mathcal{F}^B)_t = Box(f^B_t)_t$$

where f_l, f_r, f_b, f_t represent the leftmost, rightmost, bottommost and topmost faces in the set of faces $\mathcal{F}$. This formulation results in a set of equality and inequality constraints that are linear with respect to the size and position of individual faces (and thus the size and position of the cuboids to which those faces belong).

5.2 Specifying Fit Inside Relationships

To specify a fit inside relationship, users select the set of inner parts P^A and the single enclosing part p^B, adjust joint parameters so that the parts are in their fitting configuration, and indicate whether they want a regular fit inside or "just fits inside" relationship. Users can also specify a "fit partially inside" relationship by selecting just a portion of an inner p^A to fit inside p^B.

Based on the specified parts and fitting configuration, our system generates the constraints defined in Section 4.2 to ensure that the inner parts P^A fit inside p^B. We use a similar strategy as with the cover relationship constraints. In particular, we use the same plane-sweep approach described earlier, but this time in 3 dimensions to determine a spatial ordering over all the parts in x, y and z. We then generate non-overlapping constraints between consecutive parts in each dimension and constrain the coordinates of the leftmost, rightmost, bottommost, topmost, nearest and farthest part bounding boxes to be equal. Finally, we rewrite the constraints from Equation 2 on the sizes and positions of $Box(P^A)$ and $Box(p^B)$ to obtain a set of linear constraints on the sizes and positions of the individual part cuboids in P^A and p^B.

5.3 Specifying Support Relationships

Users specify a support relationship by selecting the set of all parts that contain supporting faces F^A and/or supported faces F^B and adjusting joint parameters so that the faces are in their supported configuration. The system infers the corresponding faces graph M using a similar algorithm as described above for the cover relationship. However, in this case, we only consider horizontal faces, and we allow each pair of candidate supporting/supported faces to be separated by a larger threshold distance and to not overlap when projected onto the same plane. As with the cover relationship, the user can manually fix any incorrect face corresondences.

Given M, our system automatically generates the geometric constraints described in Section 4.3 for each pair of corresponding supporting/supported faces and their parent cuboids. The system also identifies all parts with supporting faces but no supported faces and constrains the bottom coordinates of those parts to be equal.

5.4 Specifying Flush Relationships

To specify a flush relationship, users select the two faces that must be coplanar and adjusts joint parameters to move the faces into their flush configuration. The system automatically adds an equality constraint to the appropriate coordinates of the parent cuboids of the selected faces.

5.5 Specifying Additional Geometric Constraints

Although the primary goal of works-like prototypes is to represent mechanical functionality, there may be some aesthetic properties of a design (e.g., symmetry) that the designer wants to enforce. To address such cases, we allow users to directly add low-level geometric inequality and equality constraints between the dimensions of part cuboids. For example, we constrain all the parts in the crate bed prototype (Figure 10) to have the same thickness.

5.6 Double Pivot Joint Constraints

As described earlier, users add double pivot joints between parts by specifying the positions of the pivots on two coplanar faces. However, in most cases, these initial pivot placements will either cause the parts to interfere as they move between the relevant user-specified configurations and/or end up in the wrong positions/orientations (Figure 7a–c). To address these problems, our system automatically generates additional geometric constraints for each double pivot joint. In particular, for each double pivot joint j with pivots u^A and u^B attached to faces f^A and f^B of parts p^A and p^B, we constrain the two pivot positions to ensure that the parts can

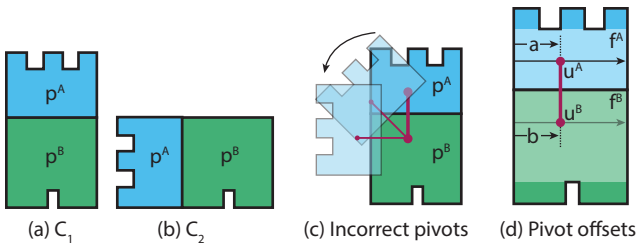
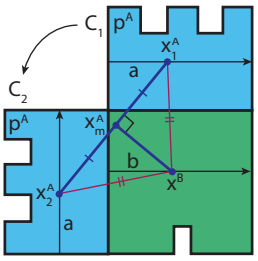


Figure 7: Double pivot joints. *Given user-specified part configurations C_1 (a) and C_2 (b), a naive placement of the pivots results in interference and an incorrect ending position for part p^A (c). We parameterize the pivot positions by pivot offsets a and b and impose constraints that enforce good pivot placements.*

successfully move between all the relevant configurations. To simplify our formulation, we restrict the possible position of each pivot to an offset vector that passes through the initial user-specified position and is parallel to the cuboid axis that corresponds to the largest cuboid dimension. Thus, we can parameterize the positions of u^A and u^B with scalar offsets a and b (Figure 7d). Here, we describe the constraints on the pivot offsets imposed by a single pair of part configurations, C_1 and C_2. If the user specifies additional part configurations, we add the corresponding constraints.



Position constraint. Given the positions and orientations of p^A and p^B in configurations C_1 and C_2, we can derive the following geometric constraint between the pivot offsets a and b. As illustrated in on the right, any value of a defines two positions x_1^A and x_2^A for pivot u^A that correspond to the two part configurations. Since the distance between u^A and u^B (which corresponds to the rigid link of the pivot) must remain fixed, it follows that u^B has to lie on the perpendicular bisector of the line segment that connects x_1^A and x_2^A. In addition, as explained above, we restrict u^B to lie on its offset vector. Thus, the position x^B of u^B is defined by the intersection of its offset vector and the perpendicular bisector. Based on this construction, we derive the following constraint:

$$(x_2^A - x_1^A) \cdot (x_m^A - x^B) = 0$$

Since x_1^A, x_2^A, x_m^A and x^B all depend linearly on a and b, the constraint is quadratic with respect to the pivot offsets.

Motion constraints. While the position constraint ensures that the pivots can, in theory, move p^A and p^B to the specified configurations, the distance between the pivots may not provide sufficient clearance to allow the two parts to rotate into the appropriate configurations without interfering (Figure 8 left). To enforce interference-free motion, we consider the bounding boxes of the two parts projected onto the pivot plane (i.e., the plane that contains f^A and f^B). Based on how the parts move between the two configurations, we determine what combination of bounding box corners the pivot link can potentially pass over, which allows us to derive a conservative lower bound on the length of the link (Figure 8 right). We formulate these minimum length requirements as linear constraints over a and b by representing the distance from the pivot positions to the bounding box corners (d^A and d^B in Figure 8b) with linear lower

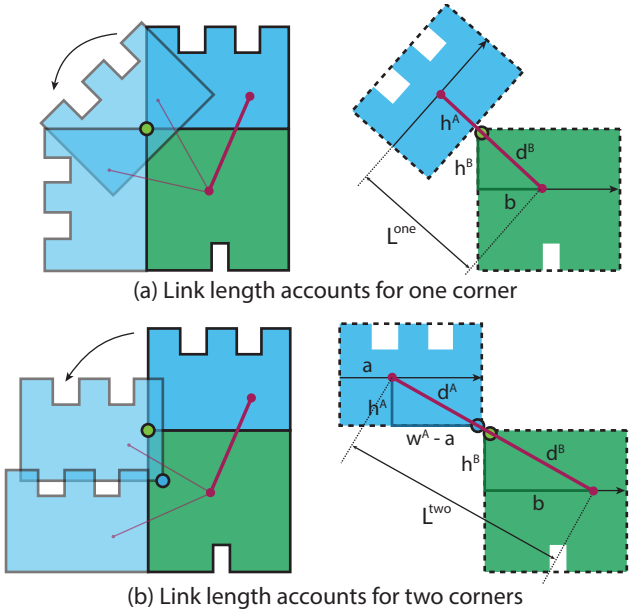


Figure 8: Motion constraint. *We automatically generate a motion constraint that ensures the pivot link is long enough to allow the parts to rotate into the appropriate configuration. We determine which bounding box corners to take into account based on the motion of the parts between the start and end configurations.*

bounds. For example, we write the pivot length constraint for the two examples in the figure as follows:

$$L^{one} > h^A + \frac{(h^B + b)}{\sqrt{2}}, \quad L^{two} > \frac{(h^A + (w^A - a) + h^B + b)}{\sqrt{2}}$$

Note that although the above constraints guarantee interference-free motion, they are conservative (i.e., the joint may be longer than necessary). Furthermore, we do not test for non-local interferences between parts.

6 Computing Part and Joint Parameters

Once the user has specified all the necessary functional relationships, we solve the constraint system to update the model. The computation proceeds in two stages. First, we consider the cuboid parameters, which we denote as $B = \{B_i\}$, where B_i represents the left, right, bottom, top, near and far coordinates of the i-th cuboid in the model. We solve for B under all the constraints related to part geometry (Sections 5.1–5.5) to determine the appropriate size and position of each part for all user-specified configurations. Then, we fix the cuboid parameters and solve for the double pivot parameters, which we denote as $L = \{L_i\}$, where L_i represents the two pivot positions of the i-th double pivot joint. We solve for L under the constraints described in Section 5.6. Since each stage updates different sets of parameters, we do not need to iterate.

For most designs, the user-specified functional relationships do not fully constrain the part and joint geometry. To find unique solutions for both cuboid and double pivot parameters, we introduce an energy function that minimizes deviations from the base configuration in a least squares sense:

$$E(B, L) = \sum_i \|B_i - \bar{B}_i\|^2 + \sum_j \|L_j - \bar{L}_j\|^2 \qquad (4)$$

where $\bar{B}$ and $\bar{L}$ are the values of the cuboid and double pivot parameters in the base configuration. With this energy function, we can

formulate both the cuboid and double pivot parameter optimizations as quadratic programming problems, which we solve using Matlab's $fmincon$ function. If there is no valid solution, the system tells the user that there are conflicting constraints.

Given that most of our constraints are linear (except for the quadratic position constraint on double pivot joints) and the fact that we typically have a relatively small number of variables, our system solves for both part and joint parameters at interactive rates. This allows users to explore design variations either by modifying the dimensions of any part cuboid, changing the desired configurations of parts, or performing discrete edits such as adding/removing parts or changing their joint types. Once the user performs an edit, the system automatically updates the rest of the model based on the geometric dependencies imposed by the specified functional relationships. Figure 9 shows some design variations that we generated with our system, and the submission video shows additional editing sequences.

7 Generating Fabricatable Geometry

Once we have determined all the cuboid and double pivot parameters, our system converts the design into fabricatable form. This involves three steps: adding cavities to ensure that parts can move into their fitting configurations, generating working joint geometry, and creating gaps where necessary to prevent interferences.

Cavities. We assume that only parts with fit inside relationships may require cavities to be generated. We move each inner part p^A from its base configuration to its fitting configuration with respect to the outer part p^B and sweep out a volume along this motion path. We then use CSG to subtract the swept volume from p^B.

Joint geometry. We take a procedural approach to generate joint geometry that takes into account the geometry of the connected parts. For a hinge attached to edge e, we attach a cylindrical *pin* to one part and two *yokes* that surround the pin to the other part. We set the length of the pin to the length of e and position the yokes symmetrically near the ends of the pin. We set the radii of the pin and yoke loops to half the thickness of the thinnest attached part. For a sliding joint, we check if the two connected parts fit together tightly (e.g., one part slides into a cavity). If so, we do not add any extra joint geometry. Otherwise, we add mated rails that allow the parts to move along the sliding vector with respect to one another. For a sliding hinge, we make the hinge pin slightly longer than the shared edge e and then add rails that allow the entire pin to move along the sliding vector. For double pivot joints, we generate a pin for each pivot that allows it to rotate and then connect the pivots with a rigid link.

Gaps. Since our joints are designed to be printed in fully assembled form, we add a small gap around all touching faces to ensure that parts do not fuse together and that the support material can be removed after printing. We also add gaps around cavities and all the moving components of the synthesized joints. We export the final printable geometry as an STL file that can be sent directly to the printer.

8 Results

We used our system to create works-like prototypes for 8 different designs (Figure 10). Our submission video shows how the various parts move and fit together. Creating each working model in our system took between 1–10 minutes of user interaction. Modeling the parts and adding joints took about three quarters of the time, and the remaining time was spent specifying functional relationships.

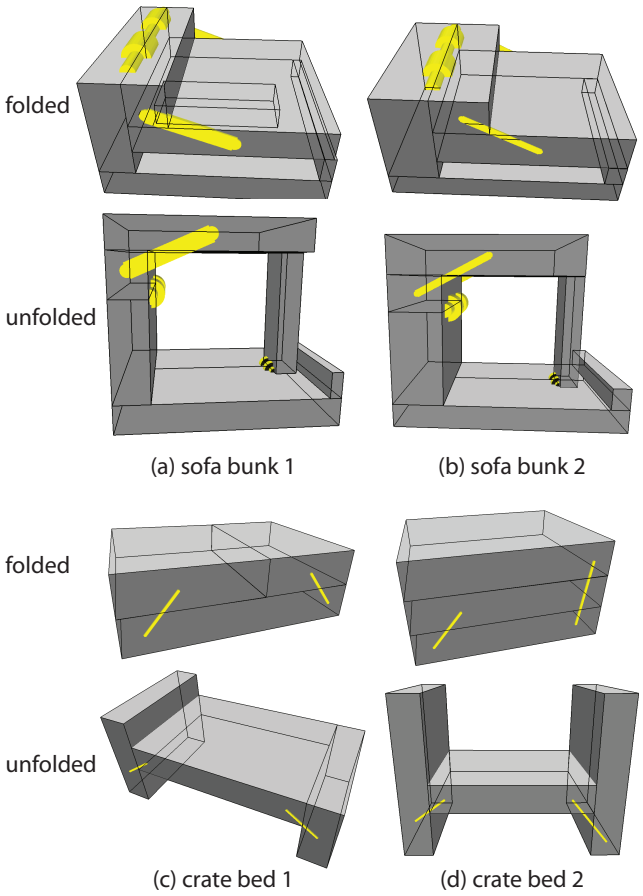


Figure 9: Variations of sofa bunk and crate bed.

In addition to cover, fit-inside, support and flush relationships, we added 4 geometric constraints: 2 symmetry constraints for the top two cabinet doors and the two folding sections of the crate bar; and 2 to make the folding parts of the tablet and crate bed the same thickness. Table 1 summarizes various statistics for all the results.

While some of our prototypes may appear simple, mechanisms with even a small number of moving parts often involve non-obvious geometric dependencies. By automatically maintaining the specified functional relationships, our system helps users create an initial working design from a rough input model and modify that design to generate variations. For example, the crate bed only has three parts, but creating variations of the design with different arrangements of the head and footboards requires non-trivial adjustments to the double pivot joint parameters (Figures 9c–d). The folding sofa also has interesting dependencies due to the fact that the top bunk must be supported when the model is unfolded into the bed configuration and then fit together with the headrest and base when folded into the sofa configuration. Figures 9a–b) shows two variations that we created by changing the height of the headrest. The system automatically updated the other part and joint parameters appropriately to maintain the functional relationships. Our system also supports discrete edits. For example, our submission video shows an editing sequence with a jewelry box design where removing various compartments causes the remaining parts to update in different ways.

While the most common use case for our system is to help designers generate working models that can be 3D printed, the optimized part and joint parameters can also be used as instructions for hand-built

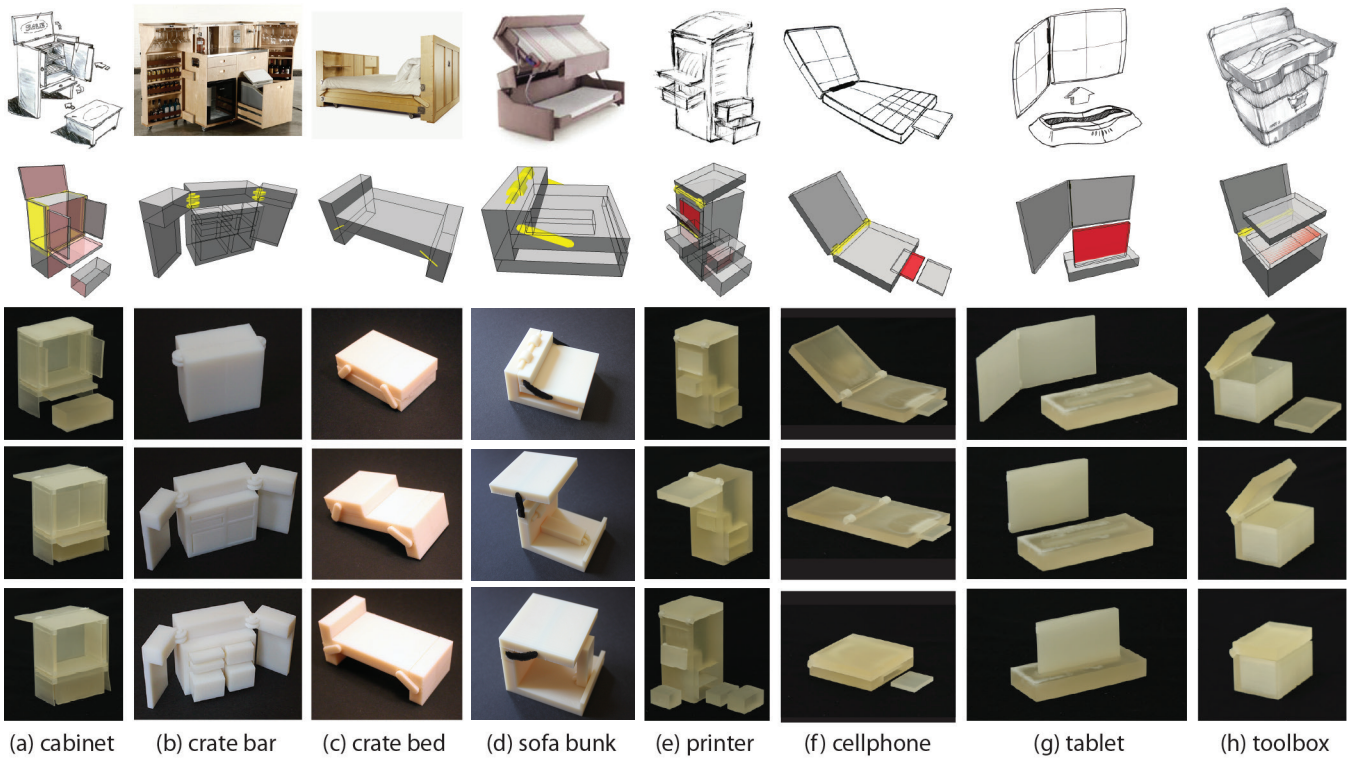


Figure 10: Various works-like prototypes created using our system.

prototypes. For example, Figure 11 shows a variation of the crate bed design that we built out of paper using measurements computed by the system.

Designer feedback. To get informal feedback on our approach, we showed our system to the three professional designers that we consulted as part of our formative work. We asked them to comment on the general usefulness of the system, whether they could imagine using it as part of their workflows, and what aspects of the system should be improved. All of the designers felt that our system would be very useful during the early stages of design when works-like prototypes provide important feedback on potential mechanical architectures for a product. They said that our tool was better suited to this kind of early prototyping than existing CAD software; one of the Anvil designers, who has been using SolidWorks for over 15 years, said that our system significantly streamlines the process of designing working mechanisms by abstracting away the low-level details of joint and part geometry. There was also a positive reaction to our use of functional relationships as the main authoring paradigm, which allows users to understand and engage with moving parts in an intuitive way.

Models	# Joints				# Constraints		Int. Time (s)	
	Hinge	Slide	S-Hinge	D-Pivot	Fxn	Geom	Proxy	Fxn
Cabinet	1	1	3	0	7	1	150	90
Crate Bar	2	5	0	0	2	1	300	60
Crate Bed	0	0	0	4	3	1	60	40
Sofa Bunk	2	0	0	2	5	1	450	60
Printer	2	3	0	0	2	0	90	90
Cellphone	1	1	0	0	2	1	30	40
Tablet	1	1	0	0	2	1	40	30
Toolbox	1	1	0	0	2	0	40	40

Table 1: Statistics for fabricated prototypes.

Limitations. The main limitation of our approach is the restricted set of part primitives and joint types that we support. While compositions of cuboids are sufficient for many works-like prototypes, designers sometimes want higher fidelity geometry to understand the relationships between the form and function of a design. Similarly, other joint types and part interactions could be useful for prototyping certain classes of products, including ball joints, snapping features, threads, and simple gears. Our current system can certainly be extended to handle more complex geometry and a wider range of joints, but this would require modifications to the constraints imposed by our functional relationships.

9 Conclusions and Future Work

In this work, we present a new approach to authoring works-like prototypes with functional mechanisms. By providing a 3D modeling interface where geometry is determined by functional relationships between parts, we allow users to work top-down and focus on the functional goals of the design rather than working bottom-up from low-level geometric details. Our fabricated results demonstrate that our system can generate functional models with a small

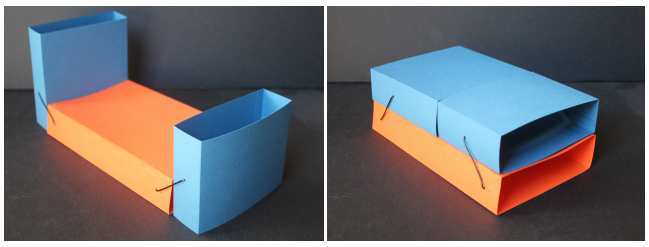


Figure 11: Hand-built prototype.

amount of high-level user interaction, and the informal feedback from professional designers suggest that our approach could significantly improve existing prototyping workflows.

Given the recent advances in 3D printing, we see a huge potential for modeling tools that target the physical prototypes that product designers create. Our work takes a small step in this direction, but we see many opportunities for future work in this vein:

Other functional properties. While we focused on four common functional relationships, it would be interesting to consider other functional requirements like whether an object can stand or roll or fold flat. Supporting such requirements would entail further analysis of the geometric and physical properties of the design.

Interacting with existing geometry. Some products are designed to interact with existing reference objects (e.g., phone cases and bicycle mounts). Thus, it could be useful to extend our approach to handle interactions between parts and this reference geometry.

Alternative printing technologies. High-resolution 3D printers can produce accurate geometry with assembly-free mechanisms, but these printers are slow and expensive. One area for future work would be to develop techniques that produce working mechanisms with faster and cheaper printing technologies to reduce the time and cost of mechanical prototyping.

Acknowledgements

We thank the reviewers for their helpful feedback. We also thank Arna Ionescu from Proteus Digital Health, and Greg Janky and Treasure Hinds from Anvil Studios for their insightful feedback. This work was supported in part by the Marie Curie Career Integration Grant 303541, the ERC Starting Grant SmartGeometry (StG-2013-335373), and gifts from Adobe Research.

References

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4.

BAE, S.-H., BALAKRISHNAN, R., AND SINGH, K. 2008. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *ACM UIST*, 151–160.

BOKELOH, M., WAND, M., SEIDEL, H.-P., AND KOLTUN, V. 2012. An algebraic model for parameterized shape editing. *ACM SIGGRAPH 31*, 4, 78:1–78:10.

CALÌ, J., CALIAN, D., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3d-printing of non-assembly, articulated models. 130:1–130:8.

CEYLAN, D., LI, W., MITRA, N. J., AGRAWALA, M., AND PAULY, M. 2013. Designing and fabricating mechanical automata from mocap sequences. *ACM SIGGRAPH Asia 32*, 6.

COROS, S., THOMASZEWSKI, B., NORIS, G., SUEDA, S., FORBERG, M., SUMNER, R. W., MATUSIK, W., AND BICKEL, B. 2013. Computational design of mechanical characters. *ACM SIGGRAPH 32*, 4, 83:1–83:12.

DANIEL, M., AND LUCAS, M. 1997. Towards declarative geometric modelling in mechanics. In *Integrated Design and Manufacturing in Mechanical Engineering*, P. Chedmail, J.-C. Bocquet, and D. Dornfeld, Eds. Springer Netherlands, 427–436.

EISSEN, K., AND STEUR, R. 2009. *Sketching: Drawing Techniques for Product Designers*. BIS Publishers.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iwires: An analyze-and-edit approach to shape manipulation. *ACM SIGGRAPH 28*, 3, #33, 1–10.

HALLGRIMSSON, B. 2012. *Prototyping and modelmaking for product design*. Laurence King.

KELLEY, T. 2001. *The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm*. Crown Business.

NIEVERGELT, J., AND PREPARATA, F. P. 1982. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM 25*, 10 (Oct.), 739–747.

PRÉVOST, R., WHITING, E., LEFEBVRE, S., AND SORKINE-HORNUNG, O. 2013. Make It Stand: Balancing shapes for 3D fabrication. *ACM SIGGRAPH 32*, 4, 81:1–81:10.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Transactions on Graphics 28*, 5. ACM SIGGRAPH Asia.

SHAO, C., BOUSSEAU, A., SHEFFER, A., AND SINGH, K. 2012. Crossshade: Shading concept sketches using cross-section curves. *ACM SIGGRAPH 31*, 4.

SHAO, T., LI, W., ZHOU, K., XU, W., GUO, B., AND MITRA, N. J. 2013. Interpreting concept sketches. *ACM SIGGRAPH 32*, 4.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: Improving structural strength of 3d printable objects. *ACM SIGGRAPH 31*, 4, 48:1–48:11.

ULRICH, K., AND EPPINGER, S. 2007. *Product Design and Development*. McGraw-Hill.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM SIGGRAPH 31*, 4, 86:1–86:11.

XU, W., WANG, J., YIN, K., ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. In *ACM SIGGRAPH*, 35:1–35:9.

XU, B., CHANG, W., SHEFFER, A., BOUSSEAU, A., MCCRAE, J., AND SINGH, K. 2014. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM SIGGRAPH 33*, 4.

YVARS, P.-A. 2008. Using constraint satisfaction for designing mechanical systems. *International Journal on Interactive Design and Manufacturing (IJIDeM) 2*, 3, 161–167.

ZHENG, Y., CHEN, X., CHENG, M.-M., ZHOU, K., HU, S.-M., AND MITRA, N. J. 2012. Interactive images: Cuboid proxies for smart image manipulation. *ACM SIGGRAPH 31*, 4, 99:1–99:11.

ZHU, L., XU, W., SNYDER, J., LIU, Y., WANG, G., AND GUO, B. 2012. Motion-guided mechanical toy modeling. *ACM SIGGRAPH Asia 31*, 6, 127:1–127:10.