AutoConnect: computational design of 3D-printable connectors

AutoConnect: Computational Design of 3D-Printable Connectors

Yuki Koyama[1,2] Shinjiro Sueda[1,3] Emma Steinhardt[1] Takeo Igarashi[2] Ariel Shamir[1,4] Wojciech Matusik[5]

[1]Disney Research Boston [2]The University of Tokyo [3]Cal Poly [4]IDC Herzliya [5]MIT

Input: Two objects **Output:** Connector

(a) Mobile phone mounter for a car dashboard

Input: Two objects **Output:** Connector

(b) Mug holder for a chair arm

Figure 1: *AutoConnect creates 3D-printable customized connectors based on shapes of two given objects and a user-specified configuration. Custom holders are created for connecting a mobile phone to a car dashboard (a), and a mug to a chair (b).*

Abstract

We present AutoConnect, an automatic method that creates customized, 3D-printable connectors attaching two physical objects together. Users simply position and orient virtual models of the two objects that they want to connect and indicate some auxiliary information such as weight and dimensions. Then, AutoConnect creates several alternative designs that users can choose from for 3D printing. The design of the connector is created by combining two holders, one for each object. We categorize the holders into two types. The first type holds standard objects such as pipes and planes. We utilize a database of parameterized mechanical holders and optimize the holder shape based on the grip strength and material consumption. The second type holds free-form objects. These are procedurally generated shell-gripper designs created based on geometric analysis of the object. We illustrate the use of our method by demonstrating many examples of connectors and practical use cases.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.8 [Computer Graphics]: Applications

Keywords: 3D printing, fabrication, functional design

1 Introduction

Modern manufacturing processes such as 3D printing allow users to personally manufacture and print 3D objects. However, designing functional 3D objects is challenging, complex, and it demands domain-specific expertise. To address this, many end-users are limited to selecting 3D objects from design repositories (*e.g.,* `http://www.thingiverse.com`). One very common type of design in these repositories is a *connector* that attaches two mass-manufactured objects in a new way. For instance, there are many different designs for attaching smart phones to bikes. However, if users want custom connectors that attach less common objects, or attach two objects in a particular way, finding the right connector is typically impossible.

We address this problem by developing a method that automatically creates custom connectors for attaching two arbitrary objects in a desired configuration. Our AutoConnect method requires 3D models of both objects, such as an iPhone and a car dashboard or a mug and a chair-arm (see Fig. 1). Such models could either be downloaded from the Internet or created using scanning systems such as Kinect Fusion or Autodesk 123D Catch. Users virtually position both models relative to each other and provide some auxiliary information about them (*e.g.,* dimensions, weights, directions of free motion, and regions on the object that should remain uncovered). AutoConnect then provides a set of diverse suggestions of potential designs that are tailored specifically to connect the two input objects. These suggestions are computed by running geometric and force analyses in the background. Once users choose a design, AutoConnect finalizes and creates a customized connector that attaches the two objects in the desired configuration. The connector is functional and can be fabricated using a 3D printer. Fig. 2 summarizes this workflow. The user provides the "what and where," and our AutoConnect method determines the "how."

Our proposed problem is challenging because a complete geometry of a functional connector that can attach two input objects needs to be created automatically. Furthermore, there are many different possible design choices and objectives. The two objects can have completely different geometric structures. Some objects (*e.g.,* chairs, tables) contain common shapes such as pipes or planes, but others (*e.g.,* the Stanford bunny or shoes) have a free-form surface design. In addition, other considerations such as printing cost and aesthetics must also be addressed. Finally, connectors must be fabricated and satisfy the physical requirements (*e.g.,* holding objects without breaking or slipping).

We classify the objects being connected into two categories: structured objects that have a simple standard shape, such as a cylinder (bike frame) or a box edge (table top), and free-form objects that have a non-standard shape. We define two types of *holders* for

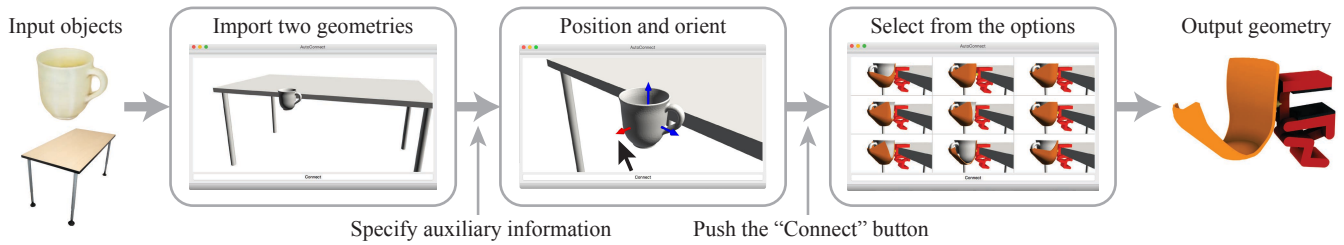


Figure 2: *User experience. AutoConnect requires a simple input to obtain a functional connector. Our method generates many connectors and allows users to choose the best one. Users can also specify some auxiliary parameters such as a free motion direction and a region on the object that should not be covered.*

these two categories, respectively. For structured objects, we use a database of parameterized mechanical holders. We optimize holder parameters based on the grip strength requirements and the material usage (§3). For free-form objects that cannot be held using structured mechanical holders, we create a custom shell-like holder based on geometric analysis of the connected object (§4). The final connector is created by combining two such holders. Hence, the final connector could be composed of two mechanical holders, a mechanical holder and a free-form holder, or two free-form holders. The holders themselves are fused using a structure that takes into account the physical requirements of the objects (§5). If one of the two objects is to be 3D-printed directly, we simply fuse it with a single holder (see Fig. 13 (i)).

Our primary contribution is to present the first method that automatically creates 3D-printable connectors and to validate this method on a number of practical use cases (Fig. 1). To enable this practical application, we offer the following two technical contributions for generating functional holders:

- We create a set of 3D-printable, parameterized, mechanical holders as well as an associated data-driven model that accurately captures their properties. All this is included as a supplementary material with the paper. We show how holder parameters can be optimized according to the desired grip strength.
- We develop a method for generating holders for free-form objects that ensures appropriate holdability and grip based on geometric analysis. We automatically create a diverse set of holders for the user to choose from.

2 Related Work

Commercial Sites and Systems Today, one of the most popular ways to obtain functional models for fabrication is to search for such models on the Internet. For example, Thingiverse is a web site for such a purpose where end-users can download and share 3D models. User can find many iPhone cases and bike connectors, but if they want a special kind of case or connector, chances are that this specific 3D model will not be found. Some 3D models allow customization by exposing some parameters, but customized connectors are rare and are usually limited to some trivial modifications—users can rarely change the type of objects being connected. In contrast, our method allows creating a completely new connector for a variety of objects and their relative arrangements. Another way of creating connectors is to design them using CAD tools such as Solidworks [Dassault Systèmes] or AutoCAD [Autodesk]. However, this requires domain-specific expertise and taking into account any functionality constraints. We develop an automated method that allows end-users to easily generate connectors between any two target shapes and some intuitive input parameters.

Functional Design and Fabrication In order to allow non-expert users to create functional objects, researchers have investigated various computational design methods. Each of these tools produces objects that meet some specific functionality. For example, Prévost et al. [2013] proposed a method to evaluate and optimize the *standability* of 3D-printed objects, while the method proposed by Bächer et al. [2014] considers the *spinability* of objects. Umetani et al. [2014] proposed a method to deal with *flyability* of lasercut paper airplanes. General *structural strength* of 3D-printed objects has also been investigated recently by Stava et al. [2012] and Lu et al. [2014]. In our method, we focus on two specific functionalities for designing connectors: *grip strength* of mechanical holders, and *holdability* of free-form shapes. Other works follow an interactive approach to modeling, while still constraining some functionality of a design. These include works on durability and validity in interactive furniture design [Umetani et al. 2012], creation of chairs [Saul et al. 2011], stacking ability [Li et al. 2012], and fabricability by including connectors and fasteners [Schulz et al. 2014]. There is also a range of works that do not necessarily target fabrication but still address semantic-aware shape manipulations of man-made objects (*e.g.,* [Xu et al. 2009; Gal et al. 2009; Fish et al. 2014]). In our work, the human interaction part is kept to a minimum—only placing the two objects relative to each other and choosing some initial parameters. The system proposed by Schulz et al. [2014] also utilizes existing functional models to create new ones from parts. Such an example-based approach to modeling is promising, especially to help novice users with little knowledge to design new objects. Our method for designing mechanical holders is an implicit example-based method, since we use a database of parametric models of 3D-printable mechanical holders. However, the user does not need to explicitly pick or change the design—this is done automatically.

3D-Printing Mechanisms Modern commercial 3D printers allow printing not only static parts, but complex mechanical objects. By combining mechanism design with computational techniques, researchers have investigated several applications, for example, designing and printing articulated models [Calì et al. 2012; Bächer et al. 2012], toys [Zhu et al. 2012; Zhou et al. 2014], characters with designable motions [Coros et al. 2013; Thomaszewski et al. 2014], and figures that mimic human motions [Ceylan et al. 2013]. Others allow printing small working-prototypes of mechanical designs for testing [Koo et al. 2014]. Our method is the first to utilize mechanical *holders* for effectively gripping target objects. In addition, we combine the design with real-world measurement data to estimate functionality given different shape parameters. This measurement-based approach encapsulates the complex mechanism of holders, and allows handling many types of mechanisms in a consistent manner. Cost considerations also come into play when printing. Similar to previous work that examines the tradeoff between cost and strength [Wang et al. 2013], we consider the tradeoff between cost and grip strength while optimizing the shape parame-

ters of mechanical holders. Optimizing for the best orientation or best support structure is an important consideration [Vanek et al. 2014; Hu et al. 2014; Umetani and Schmidt 2013], but we do not address these issues in this work.

Analysis of Holding Objects Methods to hold objects could be broadly classified into two categories: holding objects by using frictional forces (*e.g.,* pinching an object with two fingers), and holding objects by contact regardless of the friction (*e.g.,* enveloping an object in your palm). In robotic grasping theory, these two alternatives are called *force closure* and *form closure*, respectively [Bicchi and Kumar 2000]. In our work we use both methods. We use force closure for mechanical holders using data-driven measurements, and form closure for free-form objects. A force-closure approach is also possible for free-form objects, by using physical analysis. For example, Chen et al. [2014] introduce a force-closure example of a phone holder. However, robotic hands can be actively actuated, while 3D-printed holders are passive. Our form-closure approach to free-form objects uses a *holdability* criterion. This criterion is related to methods in robotic grasping, but it is tailored to our application. In robotics, robotic-hands with a small number of contact points are used (*e.g.,* tips of robotic fingers). In our approach, we generate a continuous wrapper that fits the target free-form shape. We are also interested in holders that do not completely restrain the objects. These holders have *free* directions – the directions in which the held object is able to move without being blocked by contact. This allows, for instance, easy insertion or removal of the object into our holders. Such *blocking* relationships are investigated in the *assembly planning* domain [Wilson 1992; Hirukawa et al. 1994; Agrawala et al. 2003]. Our method uses a similar formulation to generate holders with free directions. However, as our target shapes is a free-form holder enclosing a free-form object, more contact points have to be dealt with than in assembly planning scenarios.

3 Holders for Structured Objects

We define a *structured* object as one in which its attachment area can be well approximated by a *standard* shape, which, in our current implementation, consists of: a *cylinder*, a *rectangular-prism*, a *box-edge*, and a *flat-plane* (Fig. 3). These standard shapes do not need to define the whole shape and can describe only a part of a given shape, as a single object can have multiple attachment areas. As an example, a table can be approximated by a flat-plane at the top, four cylindrical legs, and a box-edge at the table edges. The attachment area of a 3D object can be classified into one of these shapes by analyzing the object using geometry processing methods such as slippage analysis [Gelfand and Guibas 2004], or a manual labeling by the user. Hence, we assume that this information is part of the object description. Any object that does not fall under one of these categories is treated as a free-form shape (§4).

We create a database of mechanical holders annotated with their grip strength information (§3.1). We currently use six types of mechanical holders, shown in Fig. 4. Each holder type corresponds to one of the four standard shapes (*e.g.,* a cylinder, a flat-plane). The holders are designed using CSG-tree representations and are parameterized with a small number (≤ 4) of parameters, such as "width" and "thickness." These parameters are used to optimize the shape of the holder. Some of the parameters are used to fit the holder to the target object dimensions. Others are used to search for designs that minimize the volume and provide adequate grip strength (§3.2) according to a pre-defined table of stored measurements (§3.1). All of the resulting models of holders, including suction cups and multi-material toggle clamps, can be printed using a commercial printer.

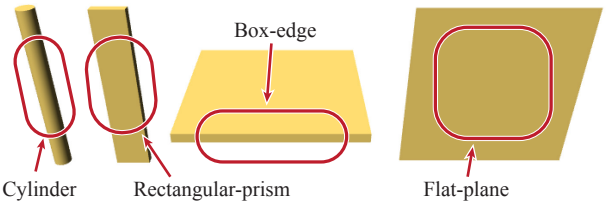


Figure 3: *Standard shapes in our current implementation. We assume that a structured object is composed of these shapes.*

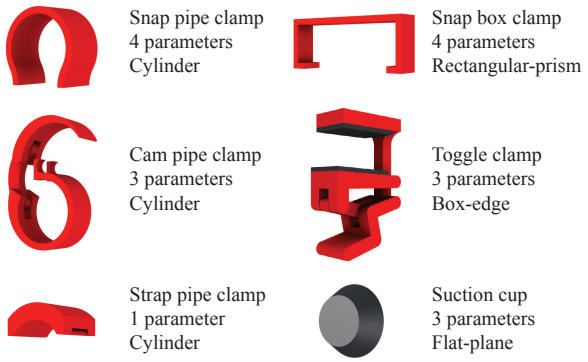


Figure 4: *Mechanical holders in our database. In each holder, the three lines of text indicates the name, the number of parameters, and the standard shape that the holder attaches to.*

To learn the relationship between the design parameters and the grip strength, we pick a set of sample points in the parameter space and print the respective holder designs. We then physically measure the grip strength and use interpolation to define an approximated grip-strength function for this holder. Note that the design and measurements of a given holder are performed once, stored in the database, and then used by our method to create many connectors. Because we provide this grip strength database as supplementary material, *other implementations of our method do not need to repeat the database creation step*. To add a new mechanical holder to the database, the following steps are necessary: designing a holder as a parametric model, printing holders with various sampled parameters, and measuring their grip strengths. The measurements take around two hours in our experiments for each new holder type. This process needs to be done only once performed by anyone. Once users have oriented and positioned the objects, the database is queried, and a set of candidate holder designs is created by optimizing for minimum material consumption while maintaining the requested grip strength (§3.2).

3.1 Grip Strength Database

Under normal operating conditions, mechanical holders are expected to stay rigidly attached without any movement. To be precise, we use the term *grip strength* to mean the minimum force and torque required to perturb the holder away from the gripping configuration. When we search the database for the appropriate holders (§3.2), we require an estimate of the grip strength provided by the holder for various parameter values.

Building an accurate model to predict the strength of the mechanical holder is a difficult task. Such a model would need to consider not only the anisotropic elasticity of the printed material but also the frictional forces or air pressure between the holder and the object. Additionally, for some of the connector mechanisms currently in

our database, such as the toggle clamp or the cam clamp, we would also require an accurate contact model internal to the mechanism itself. If we were to add new types of mechanisms, such as Velcro, snap fasteners, or even screws, we would require a very complex, non-standard physics simulation. Finally, we require an approach that can analyze numerous parametric designs at interactive rates.

In order to make this problem tractable, we make some simplifying assumptions. First, we assume that we are dealing with light-weight objects, such as hand-held objects. Second, we assume that the weakest part of the holder is at the interface between the holder and the object—in other words, the holder itself does not break first. Third, we assume that the material properties do not depend on the number of times or the amount of total time the holder is used. Finally, we assume that the surface that the mechanical holder grips has a relatively low coefficient of friction (*e.g.,* we assume the worst case, low friction, scenario). See the supplemental document for a comparison of the gripping forces for two surfaces with different coefficients of friction.

A natural approach for estimating the grip strength is to use physical simulation each time a parameter is changed. For some of the holders in our database, simulation techniques would indeed be very effective. However, because typical simulation techniques would not suffice for some of the other holders, such as the suction cups or toggle clamps, we take a data-driven approach by building a lookup table of real data measurements. We capitalize on the recent success of data-driven physics in the context of digital fabrication [Bickel et al. 2010; Umetani et al. 2014]. Note, however, that it is possible to use simulation results in the lookup table along with measured results.

In our approach, we use scattered-data interpolation to learn the relationship between the design parameters $\mathbf{x}$ and the grip strength. We physically measure the grip strength for a discrete set of sample points in the parameter space. We use interpolation to define the grip strength functions $G_{\mathrm{force}}(\mathbf{x})$ and $G_{\mathrm{torque}}(\mathbf{x})$ on the whole parametric space. In our implementation, we use Radial Basis Function (RBF) interpolation [Anjyo et al. 2014], with the Gaussian kernel $K(x) = \exp(-\frac{x^2}{2})$. Other types of interpolation are also possible.

To compute the grip strength for each sample point, we print the specific holder model and physically measure the *minimum* force and torque that is required to move the holder by slowly pulling on the holder as shown in Fig. 5. We use a digital force meter to measure forces, and a video camera to record the measurement process, so that we can read the maximum force before the holder starts to move. We consider various possible directions for each printed holder and we use the minimum force to produce movement. When the movement occurs rotationally for a certain direction, we compute the torque around the center of rotation by dividing the measured force by the moment arm, and register the torque value to the database. For each mechanical holder, we obtain 15 to 25 sample points according to the number of design parameters. Details of the measurement setting and discussions of the quality of this approach can be found in the supplemental document. Although the accuracy of this approximation is moderate, in practice, using a small safety factor (§3.2) is enough to produce satisfactory results that also account for data interpolation.

3.2 Optimizing Shape Parameters

In most mechanical holder designs, there is a trade-off between the volume (material consumption) and the functionality (grip strength) of the holder. To balance such a trade-off, our method optimizes the parameters of mechanical holders so that the user can reduce the consumption of unnecessary materials while ensuring that the holder satisfies the functional requirement.

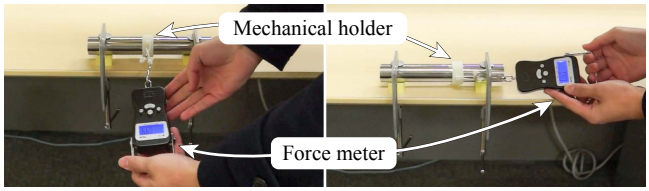


Figure 5: *Our setup for measuring the grip strength. We measure the minimum force and torque that are required to move the holder by pulling in various directions.*

The functional requirement is on the required grip strength: $G^{\mathrm{target}}_{\mathrm{force}} = s\|m\mathbf{g}\|$ and $G^{\mathrm{target}}_{\mathrm{torque}} = sl\|m\mathbf{g}\|$, where m is the mass of the object, l is the distance between the center of mass and the axis of the rotation[1], and s is the safety factor, set by the user. (We use the default value of $s = 5$ unless otherwise stated.) This assumes the worst-case scenario of the direction of the applied force being perpendicular to the moment arm direction, creating maximal torque. Hence, the optimization is formulated as minimizing the volume objective, with a constraint that the grip strength will be larger than $G^{\mathrm{target}}_{\mathrm{force}}$ and $G^{\mathrm{target}}_{\mathrm{torque}}$.

$$\begin{aligned} \underset{\mathbf{x}\in\mathcal{X}}{\text{minimize}} \quad & V(\mathbf{x}) \\ \text{subject to} \quad & G_{\mathrm{force}}(\mathbf{x}) \geq G^{\mathrm{target}}_{\mathrm{force}}, \\ & G_{\mathrm{torque}}(\mathbf{x}) \geq G^{\mathrm{target}}_{\mathrm{torque}}, \\ & C_{\mathrm{geom}}(\mathbf{x}) = 0, \\ & C_1(\mathbf{x}) \geq 0, \cdots, C_k(\mathbf{x}) \geq 0, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathcal{X}$ are the parameters that should be optimized, whose variable range $\mathcal{X}$ is manually designed for each holder type (see the supplemental document), $V(\mathbf{x})$ is a function that provides the volume of the design, $G(\mathbf{x})$ is the function that provides the estimated grip strength for a given parameter setting $\mathbf{x}$ described above, $C_{\mathrm{geom}}(\mathbf{x})$ is a geometric constraint that ensures the design fits the target standard shape (*e.g.,* the radius of a pipe clamp should be determined by the radius of the target pipe), and $C_1(\mathbf{x}), \ldots, C_k(\mathbf{x})$ are any additional constraints, such as minimum thickness.

Because we use CSG representations for the holder designs, exact computation of the volume $V(\mathbf{x})$ can be time consuming (*e.g.,* for the toggle clamp, it takes a few minutes to compute $V(\mathbf{x})$). To efficiently compute the volume during optimization, we approximate it by using an RBF data-interpolation approach; as preprocessing, we generate many sample designs with various parameters $\mathbf{x}$, compute their exact volumes, and define the function $V(\mathbf{x})$ for the whole parametric space using interpolation. Other approaches, including analytical approaches, are also possible.

The optimization itself is solved by COBYLA (gradient-free optimization) algorithm [Powell 1998] from the NLopt library. (Other algorithms, such as simulated annealing, can also be used.) To avoid getting stuck in local minima, we generate 20 random initial solutions for the optimization, and use the best results. This computation usually takes less than one second to complete. Fig. 6 shows example results of the snap pipe clamp for various target grip strength.

[1] We define the (approximate) axis of the rotation for each holder type based on the observation during measurement. This is described in the supplemental document.

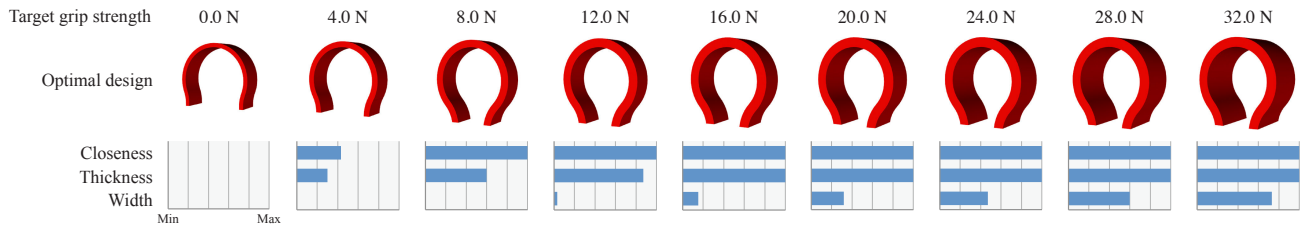


Figure 6: *Example results of our optimization while increasing the target grip strength $G_{\text{force}}^{\text{target}}$ (from left to right) for the snap pipe clamp. This clamp has 4 design parameters, one of which defines the target diameter and is fixed in the example. The other 3 parameters are optimized so that the volume is minimized while the target grip strength is achieved. For explanatory purpose, we specify $G_{\text{torque}}^{\text{target}} = 0.0\ \text{N} \cdot \text{m}$ here.*

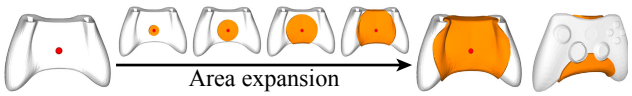


Figure 7: *Generating a free-form holder (orange) by computing the shell starting from a seed point (red) until the holdability criteria is achieved.*

4 Holders for Free-Form Objects

If the attachment area of an object does not have a standard shape, we categorize the object as a *free-form* object. For such an object, we use a geometry-based approach to generate a holder whose contact points prevent the object from moving. The user can optionally specify some auxiliary information, such as areas on the object that should remain uncovered and free motion directions. As a general principle, we aim to generate a small but diverse set of holders for a given object. Then, the user picks a holder from this set.

The overview of the approach is shown in Fig. 7. The input is the mesh of the free-form object. We choose a starting point on the mesh and apply *area expansion* using several priority biases (§4.1). As the holder area expands, the contact area increases. This makes the holder more and more capable of constraining the motion of the object. We terminate the expansion when the *holdability criterion* is satisfied (§4.2). The result is a set of triangles that provide contact points that hold the object rigidly. This set is converted into a 3D-printable mesh by thickening. In our approach, we do not consider elastic forces and friction but rather assume that everything is rigid. We rely on the geometric hold due to contact. Because there are many possible areas that can achieve holdability, and there is no clear *optimal* solution, we generate many candidate designs and allow the user to pick the most suitable one.

The generation of the free-form holder designs is composed of the following steps:

- Analyze the input mesh to find *intrinsic-free* motions (§4.3).
- Generate a shell that can hold the target object (§4.1 & §4.2).
- Optionally split the shell into multiple parts (§4.4).
- Select a diverse subset of designs to show to the user (§4.5).

4.1 Shell Computation

Computing a shell is governed by three principal characteristics, in addition to the stopping criteria §4.2: (1) what target shape to fit, (2) where to start (what seed triangle to use), and (3) what priority for area expansion to use.

By default, the target shape is the entire surface of the object. However, in order to create alternative designs, we also use the convex hull of the shape as an additional target shape for the creation of free-form holders. Other targets such as the lower envelope are also possible. In addition, users can specify certain regions on the mesh, such as the screen of a mobile phone, that should *not* be covered by the holder. The triangles in these regions are marked as *uncoverable* and they are not considered when expanding the holder area.

The starting point for the process has a large effect on the generated holder shape. For aesthetic reasons we use symmetry cues to generate these starting points. We detect the global reflective symmetry planes of the object ([Mitra et al. 2006; Podolak et al. 2006]), and choose the starting points by uniformly sampling from these planes. If no symmetry is found, we generate starting points for the expansion randomly.

The priority used in the queue for the area expansion also has a significant effect on the final shape of the holder. It biases the expansion direction by determining which triangles will be added next. The basic priority used is the geodesic distance of the triangle t in question to the seed triangle s. We use the weighted sum of the geodesic distance from the seed and two additional terms as our priority:

$$w_1 D_{\text{geod}}(t,s) + w_2 \min_k D_{\text{symm}}(t, P_k) + w_3 D_{\text{norm}}(n(t), N) \quad (2)$$

where $D_{\text{geod}}(t,s)$ is the geodesic distance between the barycenters of triangle t and triangle s, $D_{\text{symm}}(t, P_k)$ is the distance between the barycenter of t to the k^{th} symmetry plane P_k, and $D_{\text{norm}}(n(t), N)$ is the angle between the normal $n(t)$ of triangle t and a global direction vector N. The first term is used simply to expand the area at a constant speed on the target shape from the seed. The second term is used to encourage expansion along the symmetry planes (computed beforehand when finding seeds). Note that we can use a negative w_2 weight to expand in directions perpendicular to the symmetry planes. The third term is used to bias the expansion to cover certain mesh regions. For example, by setting N to the gravity direction, we can increase the coverage of the holder in the bottom part of the object (*e.g.,* for cup holders).

4.2 Holdability Criterion

To determine when to terminate the shell computation process, we use the *holdability* measure. Holdability indicates how well the contact area prevents the object from moving. Each time we add a triangle, we recompute the holdability measure and stop the process if it is above a certain threshold.

Our approach combines and extends two related concepts: *form closure*[2] from robotics [Bicchi and Kumar 2000] and *slippage analysis* from geometry processing [Gelfand et al. 2003; Gelfand and Guibas 2004]. Like slippage analysis, our approach deals with the geometry, or the triangle mesh, of the object, and thus handles a large

[2] "A condition of complete restraint in which the grasped body can resist any external disturbance wrench, irrespective of the magnitude of the contact forces."

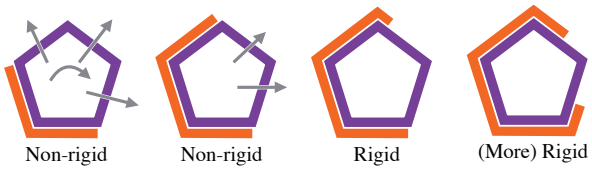
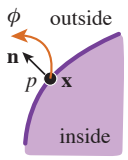


Figure 8: *2D schematic examples of our rigid and non-rigid cases. If there is no free-motion, we consider it is rigid; otherwise, it is non-rigid. Purple and orange parts correspond to the target and holder meshes respectively. Gray arrows correspond to free-motions.*

number of contact points compared to grasp analysis in robotics. However, unlike slippage analysis, our approach deals with unilateral (inequality) contact constraints, which are handled by grasp analysis. Proper handling of unilateral constraints is critical because we need to differentiate between penetration and separation. For example, slippage analysis would mark all of the cases in Fig. 8 as rigid, since any motion of the object causes the surface of the object to penetrate into or separate from the surface of the holder. Our approach correctly identifies the rigid and non-rigid cases because it can differentiate between penetration and separation.



To define holdability, we start with the 6-dimensional rigid motion (or twist), ϕ. Let p be a surface point with position $\mathbf{x}$ and normal $\mathbf{n}$. Given p and ϕ, we can compute how much the surface point p moves along its *outward* normal. This value measures the amount of blockage experienced by the point p when the object is moved infinitesimally by ϕ. We call this the *contact blockage*, denoted $b(p, \phi)$. This is a unilateral condition – if the point moves away from its outward normal, then there is no blockage, and so b always takes on a non-negative value. (See the supplemental material for the derivation of b.)

As the holder area expands, the number of contact points increases. For each triangle, we use the barycenter as the contact point, assuming that the input mesh is well-conditioned. Let $\mathcal{T}$ be the set of triangles of the current shell. Given $\mathcal{T}$ and ϕ, we define a non-negative valued *subregion blockage* as a sum of contact blockages:

$$B(\mathcal{T}, \phi) = \sum_i b(p_i, \phi), \tag{3}$$

where p_i is the barycenter of the i^{th} triangle. Intuitively, B represents the amount of blockage the holder applies to the object when the object tries to move in the direction ϕ. If $B(\mathcal{T}, \phi) = 0$, then none of the triangles in the holder blocks ϕ, and thus ϕ is a *free-motion*. In this case, the object can locally move away from the holder defined by $\mathcal{T}$ without any collisions. If $B(\mathcal{T}, \phi) > 0$, then there is at least one triangle in $\mathcal{T}$ that blocks the object from moving in the ϕ direction.

We define the holdability of the holder $\mathcal{T}$ as the minimum blockage with regard to all the possible motions.

$$H(\mathcal{T}) = \begin{cases} \underset{\phi}{\text{minimize}} & B(\mathcal{T}, \phi), \\ \text{subject to} & \|\phi\| = 1. \end{cases} \tag{4}$$

We treat ϕ as a 6D direction, and so we restrict the search to the unit hypersphere $\|\phi\| = 1$. (We can also use any small positive number instead of 1.) We use hyper-spherical coordinates to parameterize ϕ, giving us an unconstrained minimization problem with 5 degrees of freedom that allows us to deal with the constraint $\|\phi\| = 1$ implicitly. When $H(\mathcal{T}) > 0$, *i.e.,* $B(\mathcal{T}, \phi) > 0$ for any ϕ, the target object is considered to be rigidly held by at least one triangle and cannot move without any collision.

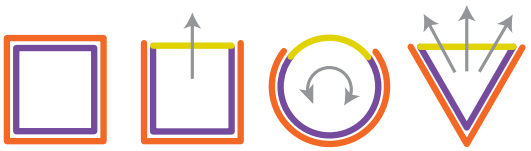


Figure 9: *Concept of intrinsic free motions. These 2D schematic examples show areas for holder expansion (purple), constrained regions (yellow), "maximum" holders (orange), and intrinsic free motions (gray).*

We define the *normalized* holdability measure as

$$\tilde{H}(\mathcal{T}) = \frac{H(\mathcal{T})}{H(\mathcal{T}_{\text{whole mesh}})}, \tag{5}$$

so that $0 \leq \tilde{H} \leq 1$. This normalization allows us to use the same threshold to get adequate results for all our examples. If $\tilde{H} \approx 0$ the area holds the object very lightly, and if $\tilde{H} \approx 1$, the area holds the object very rigidly. To balance the scaling effect of the rotations vs. translations, we regularize (scale and translate) the mesh into the unit bounding box before evaluating this function. This has the effect of roughly equalizing the maximum torque to the unit translational force [Gelfand et al. 2003]. To solve this minimization problem (Eqs. 4–5), we use COBYLA, a gradient-free method [Powell 1998]. We run this optimization every time we add a new triangle and terminate the process when $\tilde{H}(\mathcal{T})$ becomes greater than a threshold. In our implementation, we set this threshold to 0.1. The shell computation process finishes in a few seconds for a smaller mesh and around 10 seconds for a large mesh.

4.3 Free Motions

In some cases, the holdability measure is zero even if all triangles of the mesh are included in the triangle set $\mathcal{T}$. For example, for objects with primitive shapes such as a sphere or a cylinder, there remains a rotational motion that cannot be blocked by the holder composed of all the triangles of the object mesh. We call these unblockable motions the *intrinsic free motions* of the object. Before starting the shell computation process, we analyze the input mesh to find these intrinsic free motions. We ignore these motions when computing the holdability measure for the stopping criteria. The algorithm for computing the set $\mathcal{F} = \{\phi_i^{\text{free}}\}$ of intrinsic free motions of an object is given in the supplemental material. We run this algorithm on the whole input mesh as a preprocessing step before we start the shell computation process.

Some examples of intrinsic free motions are given in Fig. 9. (The yellow regions are marked as *uncoverable* by the user.) Let us first consider the box example with one free intrinsic motion (the second one from the left). Let ϕ^{free} be this free motion. When we solve for ϕ in Eq. 5, we want ϕ to not point in the same direction as ϕ^{free}. The can be expressed as

$$\phi \cdot \phi^{\text{free}} \leq \alpha. \tag{6}$$

This constraint forces the solution, ϕ, not to point within a cone of directions centered around ϕ^{free}. The parameter α controls the size of this cone. For the box example in Fig. 9, since there is a single free direction, $\alpha = 1$ would work well, preventing ϕ from pointing exactly in the direction of ϕ^{free}. However, for the triangular object in Fig. 9, there is a cone of intrinsic free motions, indicated by the three gray arrows. If we know exactly what this cone is, then we

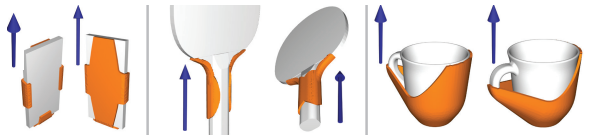


Figure 10: *Leave-one-motion examples, where the up direction is specified as an extrinsic free motion. Each example is shown from two different views. For the mug example, the convex hull is used for shell computation.*

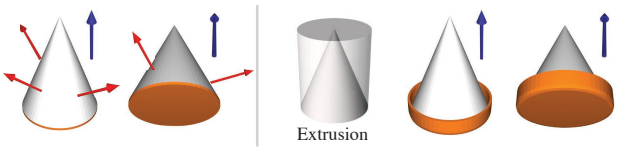


Figure 11: *A failure case in leaving one free direction for shell computation (left). In this case the up direction was specified, but the shell computation process stops before achieving the stopping criteria, while leaving more than one free direction. The process stops because no triangle can be added without blocking the up motion. A solution to this can be found by expanding the holder on an extrusion of the object along the free motion direction (right).*

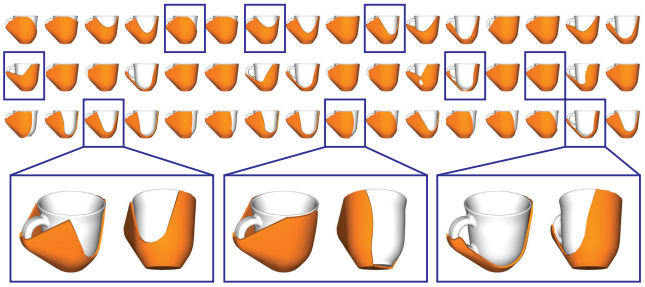


Figure 12: *(Top) 48 mug holder candidates are generated, from which the 9 holders annotated with boxes are selected by the clustering-based selection algorithm. (Bottom) Close up views of some selected mug holders.*

can set the correct value of α, and a single ϕ^{free} would be sufficient to fully cover the free motion directions. However, since we do not know the extent of these cones for an arbitrary mesh, we set $\alpha < 1$ and sample the cone of free directions. If α becomes closer to 1 then the approximation becomes more accurate and we will obtain more solutions at the expense of requiring more ϕ_i^{free}. For the triangular shape in Fig. 9, we have $i = 3$ with each ϕ_i^{free} covering roughly a third of the total cone. In our implementation, we use $\alpha = 0.5$ for all the examples.

In some cases, users may manually specify additional free motion directions, which we call *extrinsic free motions*. For example, users may specify that the holder should not block the object from moving vertically up, so that they can insert and remove the object freely in that direction. (In this case, $\phi^{\text{free}} = (0\,0\,0\,0\,0\,1)^T$, assuming gravity points in the $-z$ direction.) We add these additional motion directions to the set $\mathcal{F}$ of intrinsic free motions.

To incorporate $\mathcal{F}$ into our pipeline, we slightly modify both the area expansion process and the holdability criterion. First, when expanding the holder area, we do not add a triangle that blocks any of the specified free motions in $\mathcal{F}$. In other words, we only add triangles that satisfy the following criteria:

$$b(p, \phi_i^{\text{free}}) = 0, \quad \forall \phi_i^{\text{free}} \in \mathcal{F}, \tag{7}$$

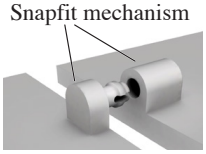
where c is the contact penetration function from §4.2, and p is the contact point on a triangle. (Note that b represents a unilateral contact constraint, and thus is never negative.) Second, we modify the computation of $H(\mathcal{T})$ in Eq. 4 by adding additional constraints:

$$\phi \cdot \phi_i^{\text{free}} < \alpha, \quad \forall \phi_i^{\text{free}} \in \mathcal{F}. \tag{8}$$

These constraints keep ϕ away from all the intrinsic and extrinsic free motions.

Discussion Fig. 10 shows some examples of free-form holders, where the *up* direction is specified as an extrinsic free-motion. Fig. 11 (left) shows a failure case, where shell computation terminates before achieving the stopping criterion because no more triangles can be added due to the condition Eq. 7. Thus, there are many free motions left in the resulting holder. One possible solution is to use the extrusion towards the specified extrinsic free direction as the area expansion target, as shown in Fig. 11 (right).

4.4 Splitting and Attaching Snapfits



When users do not specify any (extrinsic) free-motion, we split the shell into multiple parts so that we can physically attach/detach the shell to/from the target object without any intersection. We use the symmetry planes of the target objects if available or ask users to specify these cutting planes. We then add snapfit mechanisms (see the inset image) for each split part to ensure that the parts can snap to each other. In our implementation we use only translational snapfits, but it is possible to attach rotational snapfits (*e.g.,* locking hinge) when a rotational motion path is available. Please see the supplemental material for more details.

4.5 Providing a Diverse Subset

The generated holders are sometimes similar to each other even when different area expansion weights (Eq. 2) are used. Here, we show a simple algorithm to select a small number (~ 9) of diverse, distinguishable holders from the randomly generated outputs (~ 40).

Let n be the target number of designs to display to the user. First, we run the shell computation process m ($> n$) times with different weights for the area expansion (Eq. 2) and seed positions (§4.1). Then, we build an $m \times m$ similarity matrix of these shells, and then apply the spectral clustering algorithm [Shi and Malik 2000; von Luxburg 2007], using n as the number of clusters. The result of the spectral clustering is n clusters each of which contains similar shells. From each cluster, we select the representative sample that is closest to the center of the cluster. Finally, we compute smoothing and thickening for the selected shells and we display them to the user.

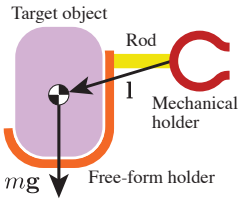
In our current implementation, we define the similarity metric for shells as follows. For the target triangle mesh that has k triangles, we consider a k-dimensional binary vector $\{0, 1\}^k$. We compute this vector for each computed shell by setting the i^{th} element as 1 if the shell contains the i^{th} triangle, and 0 otherwise. We measure the dissimilarity between shells by using the Hamming distance between the corresponding vectors. Finally, we define the similarity between shells as the dissimilarity subtracted from k. Fig. 12 shows some examples outputs of this algorithm.

5 Connecting the Two Parts

After generating two holders by the algorithms described in §3 and §4, we connect these parts by adding a cylindrical rod between them. The attachment point of the rod for the mechanical holder is set to be where the forces were applied during measurement. The attachment point for the free-form holder side is computed such that the rod length is minimized. To ensure the structural strength of the rod, we consider the *bending moment* at the end of the rod that is the furthest away from the object being held. Under the Euler-Bernoulli assumption [Umetani and Schmidt 2013], the minimum required rod radius to hold the object can be computed as

$$r = \left(\frac{4}{\pi} \cdot \frac{\|\mathbf{l} \times m\mathbf{g}\|}{\sigma_{\max}} \right)^{\frac{1}{3}}, \tag{9}$$

where $\mathbf{l}$ is the displacement vector (the vector between the rod end and the center of mass), $m\mathbf{g}$ is the gravity force on the object, and $\sigma_{\max}$ is the *maximum tensile stress* of the rod material. In our printing setting, $\sigma_{\max}$ is between 20 – 30 MPa, obtained from the material specifications. Considering the safety factor s, which can be the same value in the mechanical holder optimization (§3.2), and the *worst case* of the applied force direction, we modify Eq. 9 by replacing $\|\mathbf{l} \times m\mathbf{g}\|$ with $s\|\mathbf{l}\|\|m\mathbf{g}\|$. We also allow the user to choose a larger radius if desired.



6 Results

We use AutoConnect to create and fabricate many connectors for many practical applications. We fabricate all our examples using Stratasys Objet Connex 500 with Endur material. Standard shapes are manually labeled in our examples. In Fig. 1 and Fig. 13 (a)–(f), we connect a structured object and a free-form object, and specify a free-motion when generating connectors. For many use cases, it is desirable to specify a free-motion, since it allows users to attach and detach objects easily.

We demonstrate two examples that use off-the-shelf scanning tools to produce a connector: the car dashboard in Fig. 1 (a) and the speaker in Fig. 13 (c). For these, we use Kinect Fusion and 123D Catch iPhone App, respectively with only a few manual operations (*e.g.,* removing unnecessary parts, and smoothing noisy regions). For 123D Catch, we use a set of photographs (~ 40) of the object as the input.

Sometimes, the user may want to connect two objects very firmly. Fig. 13 (g) is an example where strong external forces can be applied to the target objects when using this connector. For this scenario, we do not specify any free-motions and we specify a large safety factor. Our method generates design options by using the strap pipe clamp, which has a very strong grip strength compared to the other mechanical holders for cylinders.

Our method can also be applied when one side of the connected objects is directly 3D-printed. Fig. 13 (i) shows such cases; first, we attach a 3D-printed dragon's head to a high-heel shoe by using the free-form holder. Second, we attach a 3D-printable object to a structured object by using one of the mechanical holders. We can also connect structured-structured combinations; we show such scenarios in Fig. 13 (h).

Performance Table 1 shows the size of the meshes, various input parameters, and the computation times for the examples. We use the safety factor of 5 for most examples. In some cases, we used a higher number to be conservative. In particular, we use 50 for the phone–bike example to force the optimization to choose the strap pipe clamp. We also use 25 for the bunny–wall so that we can hang objects from it (please see the video). Our optimization (Eq. 1) takes less than 2 seconds in all the examples using a 3 GHz Intel Core i7. The sizes of the beverage can and the belt fall outside the range specified in the database. In these cases, we simply clamp the corresponding parameters within the range in the database when performing the optimization. The number of triangles used for shell computation differs from the original input mesh if the convex hull, uncoverable regions, or a free motion direction is specified (Eq. 7). In all the examples, shell computation takes less than 2 minutes with our multi-threaded implementation, and clustering the results (§4.5) takes a fraction of a second. Before the diverse subset is visualized, we smooth and thicken the mesh by offsetting the triangles along their normals. This process, which we also multi-thread, takes a few seconds to a few minutes depending on the resolution. After the user makes the final selection, generating the print-ready, watertight mesh can take up to a couple of hours. The print itself takes up to 5 hours depending on the size of the connector.

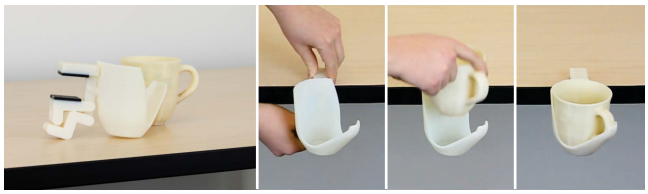
7 Conclusion

In this paper, we presented AutoConnect, a method to automatically design 3D-printable connectors that are tailored to the two input geometries and user's specifications. Our method classifies the target geometries into two categories, structured objects and free-form objects, and applies different strategies to generate holders for each of them. For structured objects, we perform force-based optimization to predefined mechanical holders. We use a data-driven approach to estimate the grip-strength based on real measurements. This database, which can be augmented with simulation as well, is made freely available as supplementary material so that *other implementations of our method do not need to recreate the database.* For free-form objects, we compute the shell of the holder with geometry-based criterion of holdability. Finally, we show various possible scenarios for the usage of AutoConnect, which include mounting various objects to the desk or chair (phone, ping-pong paddle, mug), creating phones mounts to the bike or the dashboard of the car, and creating decorative objects such as wall mounted bunny and a shoe dragon.

7.1 Future Work

Our method is the first solution to the novel and practical problem: "given two geometries of physical objects, how can we generate a *functional* and *aesthetic* connector automatically?" Since this problem is application-driven, in contrast to many geometric topics such as mesh fairing, it is difficult to mathematically formulate a simple objective. Functionality is validated both geometrically (the objects are held) and physically (the connectors do not slip or break) and is demonstrated by showing various practical examples. Aesthetics are addressed by incorporating stylistic considerations into the algorithm (such as symmetry-based priority). These are inspired by observing real-world connector designs. Aesthetics are also addressed by presenting several design variations from which the user can choose. Considering not only these but also broader aspects of human centered design in the solution is an important future work.

Currently we use only six models in our parametric mechanical holders database (Fig. 4). This limits variation of possible designs, but in the future it is easy to add more parametric holders to build a larger database, or even use a web-based service, where dedicated users can upload their holder designs. In order to add such a design to the



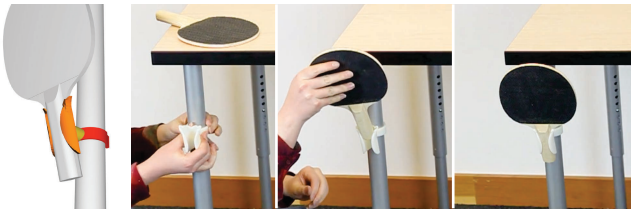
(a) Mug holder at desk edges

(b) Mobile phone holder on the wall

(c) Speaker hanger for a shelf

(d) Mobile phone attacher for guitar performance

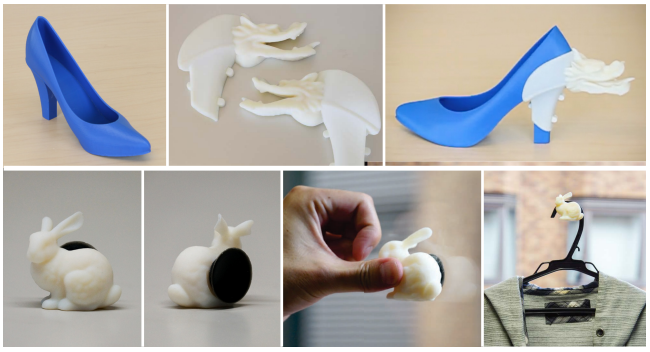
(e) Game controller holder on a chair arm



(f) Table-tennis paddle holder for table legs

(g) Mobile phone mounter for a bike handle

(h) Examples of using mechanical holders for both sides



(i) Examples of attaching printed objects to physical objects

Figure 13: *Applications of AutoConnect. In examples (a)–(g), we connect a structured object and a free-form object. Our method can also be used for other cases such as (h) the case that both objects are structured, and (i) the case that one side is directly printed.*

database, the design needs to be parametric, and the grip-strength needs to be measured (or simulated) and stored for various parameter values. The lookup and interpolation-based approach during optimization of the mechanical holders sidesteps the complexity of real-time simulation with real data and measurement. On the other hand, using explicit simulation does not require pre-processing. In the future, it would be interesting to combine the two approaches as they need not be mutually exclusive.

When the mass of the object is too large or when the user positions the object too far away, creating a large torque, the parameter optimization (Eq. 1) fails because there is no feasible solution. Also, some of our assumptions (*e.g.,* both free-form and mechanical holders are infinitely strong and never break) may become invalid. Which of these becomes invalid first depends largely on the example, and automatically determining this ordering is an interesting direction for future research. Other parts of the pipeline can also fail in some situations, such as when the user places the objects on top of each other so that the objects are intersecting. A graceful way to handle these errors would be an important future work.

As most mechanical holders are easy to detach (*e.g.,* a toggle clamp has a handle for detaching), we do not consider the difficulty of detaching in our optimization. However, since some holders, such as the snap pipe clamp, must be difficult to move but easy to detach, it would be interesting to find a way to include such conflicting requirements in our optimization. We also assume currently that both the target objects and the printed connectors do not break. In the future, we would like to consider both soft deformation and structural strength of the connectors.

For free-form holders, when the target shape is highly complex or includes many concave parts, such as in the Stanford dragon's mouth or legs or the armadillo model, our method can generate a complex holder that is difficult to attach/detach because of blocking. In this case, our split-and-verify algorithm (§4.4) often fails, or needs many splitting planes. In most practical scenarios, this is not a problem, as many artificial artifacts have nearly convex shapes, and can easily achieve attachability/detachability using a single splitting. In addition, the ability to use the convex hull of the target shape can provide some solution to this problem.

Table 1: *Performance numbers for the examples. "Mass" is the mass of the handheld object. "Grip opt." is the time in seconds to compute Eq. 1. The number of triangles used for shell computation differs from the original input mesh if the convex hull, uncoverable regions, or a free motion direction is specified (Eq. 7). "#cand." is the number of candidate holders generated (Fig. 12), which takes "shell comp." seconds to complete. Some examples contain only a mechanical holder or only a free-form holder.*

	mass (kg)	safety factor	grip opt. (s)	#triangles (orig.)	#cand.	shell comp. (s)	clust. (s)	conv. hull	uncov. regions	free dir.
phone–car	0.112	10	0.117	1836 (2600)	32	21.8	0.097	no	yes	yes
mug–chair-arm	0.320	5	0.365	1646 (10000)	48	34.8	0.143	yes	no	yes
mug–desk-edge	0.160	5	0.185	2985 (10000)	48	63.4	0.153	yes	no	yes
phone–wall	0.112	5	0.176	1921 (2600)	32	20.3	0.102	no	yes	yes
speaker–shelf	0.300	5	0.096	2837 (16000)	40	78.1	0.126	yes	no	yes
phone–guitar	0.112	5	0.178	1836 (2600)	32	23.8	0.088	no	yes	yes
phone–bike	0.112	50	0.793	2000 (2600)	40	25.7	0.122	no	yes	no
xbox–chair-arm	0.300	5	0.366	1407 (9377)	24	22.4	0.073	yes	no	yes
paddle–deskleg	0.180	5	0.691	5471 (7184)	40	58.9	0.130	no	yes	yes
shoe–dragon	-	-	-	9998 (9998)	32	85.6	0.109	no	no	no
bunny–glass	0.050	25	0.561	-	-	-	-	-	-	-
light-bike	0.150	10	1.677	-	-	-	-	-	-	-
can–deskleg	0.300	5	1.278	-	-	-	-	-	-	-
can–belt	0.300	5	1.542	-	-	-	-	-	-	-

Our work lies in the relatively unexplored research domain of automatic shape design for end-users using high-level specifications. In this context, we believe that many follow-up works would be possible. For example, it would be interesting to allow more user input either in the form of more declarative constraints such as where to position the cutting planes, where to place the connecting rod, or how to divide the connector.

Acknowledgments

We would like to thank David Levin for the insightful discussions and for helping us with the prints. We would also like to thank the anonymous reviewers for their helpful comments. Yuki Koyama is funded by JSPS research fellowship. This work was partially supported by JSPS KAKENHI Grant Number 26·8574. The bunny and dragon models are provided by The Stanford 3D Scanning Repository. The mug model used in Fig. 2, Fig. 10, Fig. 12, and Fig. 13 is copyrighted by barspin (a Thingiverse user) and licensed under CC BY 3.0.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *ACM Trans. Graph. 22*, 3 (July), 828–837.

ANJYO, K., LEWIS, J. P., AND PIGHIN, F. 2014. Scattered data interpolation for computer graphics. In *ACM SIGGRAPH 2014 Courses*, 27:1–27:69.

AUTODESK. AutoCAD. http://www.autodesk.com.

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4 (July), 47:1–47:9.

BÄCHER, M., WHITING, E., BICKEL, B., AND SORKINE-HORNUNG, O. 2014. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph. 33*, 4 (July), 96:1–96:10.

BICCHI, A., AND KUMAR, V. 2000. Robotic grasping and contact: a review. In *Proc. IEEE International Conference on Robotics and Automation*, 348–353.

BICKEL, B., BÄCHER, M., OTADUY, M. A., LEE, H. R., PFISTER, H., GROSS, M., AND MATUSIK, W. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph. 29*, 4 (July), 63:1–63:10.

CALÌ, J., CALIAN, D. A., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3D-printing of non-assembly, articulated models. *ACM Trans. Graph. 31*, 6 (Nov.), 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 Trans. Graph. 32*, 6 (Nov.), 186:1–186:11.

CHEN, X., ZHENG, C., XU, W., AND ZHOU, K. 2014. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph. 33*, 4 (July), 95:1–95:11.

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 Trans. Graph. 32*, 4 (July), 83:1–83:12.

DASSAULT SYSTÈMES. Solidworks. http://www.solidworks.com/.

FISH, N., AVERKIOU, M., VAN KAICK, O., SORKINE-HORNUNG, O., COHEN-OR, D., AND MITRA, N. J. 2014. Meta-representation of shape families. *ACM Trans. Graph. 33*, 4 (July), 34:1–34:11.

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

GELFAND, N., AND GUIBAS, L. J. 2004. Shape segmentation using local slippage analysis. In *Proc. 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 214–223.

GELFAND, N., IKEMOTO, L., RUSINKIEWICZ, S., AND LEVOY, M. 2003. Geometrically stable sampling for the icp algorithm. In *Int. Conf. 3-D Digital Imaging and Modeling*, 260–267.

HIRUKAWA, H., MATSUI, T., AND TAKASE, K. 1994. Automatic determination of possible velocity and applicable force of frictionless objects in contact from a geometric model. *IEEE Transactions on Robotics and Automation 10*, 3 (Jun), 309–322.

HU, R., LI, H., ZHANG, H., AND COHEN-OR, D. 2014. Approximate pyramidal shape decomposition. *ACM Trans. Graph. 33*, 6 (Nov.), 213:1–213:12.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph. 33*, 6 (Nov.), 217:1–217:9.

LI, H., ALHASHIM, I., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2012. Stackabilization. *ACM Trans. Graph. 31*, 6 (Nov.), 158:1–158:9.

LU, L., SHARF, A., ZHAO, H., WEI, Y., FAN, Q., CHEN, X., SAVOYE, Y., TU, C., COHEN-OR, D., AND CHEN, B. 2014. Build-to-last: Strength to weight 3D printed objects. *ACM Trans. Graph. 33*, 4 (July), 97:1–97:10.

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2006. Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph. 25*, 3 (July), 560–568.

PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S., AND FUNKHOUSER, T. 2006. A planar-reflective symmetry transform for 3D shapes. *ACM Trans. Graph. 25*, 3 (July), 549–559.

POWELL, M. J. D. 1998. Direct search algorithms for optimization calculations. *Acta Numerica 7* (1), 287–336.

PRÉVOST, R., WHITING, E., LEFEBVRE, S., AND SORKINE-HORNUNG, O. 2013. Make it stand: Balancing shapes for 3D fabrication. *ACM Trans. Graph. 32*, 4 (July), 81:1–81:10.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. Sketchchair: An all-in-one chair design system for end users. In *Proc. 5th International Conference on Tangible, Embedded, and Embodied Interaction*, ACM, 73–80.

SCHULZ, A., SHAMIR, A., LEVIN, D. I. W., SITTHI-AMORN, P., AND MATUSIK, W. 2014. Design and fabrication by example. *ACM Trans. Graph. 33*, 4 (July), 62:1–62:11.

SHI, J., AND MALIK, J. 2000. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 22*, 8 (Aug), 888–905.

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

THOMASZEWSKI, B., COROS, S., GAUGE, D., MEGARO, V., GRINSPUN, E., AND GROSS, M. 2014. Computational design of linkage-based characters. *ACM Trans. Graph. 33*, 4 (July), 64:1–64:9.

UMETANI, N., AND SCHMIDT, R. 2013. Cross-sectional structural analysis for 3D printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, ACM, New York, NY, USA, SA '13, 5:1–5:4.

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

UMETANI, N., KOYAMA, Y., SCHMIDT, R., AND IGARASHI, T. 2014. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph. 33*, 4 (July), 65:1–65:10.

VANEK, J., GALICIA, J. A. G., AND BENES, B. 2014. Clever support: Efficient support structure generation for digital fabrication. *Computer Graphics Forum 33*, 5, 117–125.

VON LUXBURG, U. 2007. A tutorial on spectral clustering. *Statistics and Computing 17*, 4, 395–416.

WANG, W., WANG, T. Y., YANG, Z., LIU, L., TONG, X., TONG, W., DENG, J., CHEN, F., AND LIU, X. 2013. Cost-effective printing of 3D objects with skin-frame structures. *ACM Trans. Graph. 32*, 6 (Nov.), 177:1–177:10.

WILSON, R. H. 1992. *On Geometric Assembly Planning*. PhD thesis, Stanford, CA, USA. UMI Order No. GAX92-21686.

XU, W., WANG, J., YIN, K., ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. *ACM Trans. Graph. 28*, 3 (July), 35:1–35:9.

ZHOU, Y., SUEDA, S., MATUSIK, W., AND SHAMIR, A. 2014. Boxelization: Folding 3D objects into boxes. *ACM Trans. Graph. 33*, 4 (July), 71:1–71:8.

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