Design and Fabrication by Example

Adriana Schulz[1] Ariel Shamir[2] David I. W. Levin[1] Pitchaya Sitthi-amorn[1] Wojciech Matusik[1]
[1]Massachusetts Institute of Technology [2]The Interdisciplinary Center Herzliya

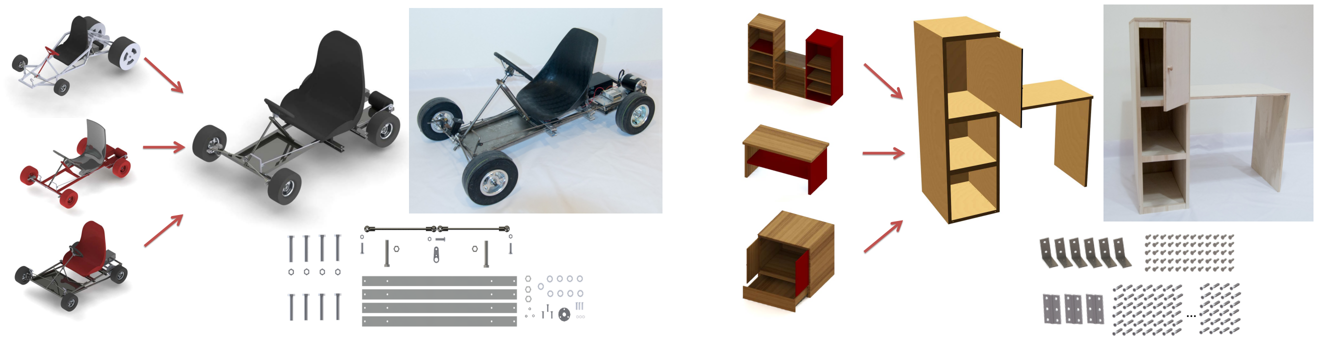

Figure 1: *The design and fabrication by example pipeline: casual users design new models by composing parts from a database of fabricable templates. The system assists the users in this task by automatically aligning parts and assigning appropriate connectors. The output of the system is a detailed model that includes all components necessary for fabrication.*

Abstract

We propose a data-driven method for designing 3D models that can be fabricated. First, our approach converts a collection of expert-created designs to a dataset of parameterized design templates that includes all information necessary for fabrication. The templates are then used in an interactive design system to create new fabricable models in a design-by-example manner. A simple interface allows novice users to choose template parts from the database, change their parameters, and combine them to create new models. Using the information in the template database, the system can automatically position, align, and connect parts: the system accomplishes this by adjusting parameters, adding appropriate constraints, and assigning connectors. This process ensures that the created models can be fabricated, saves the user from many tedious but necessary tasks, and makes it possible for non-experts to design and create actual physical objects. To demonstrate our data-driven method, we present several examples of complex functional objects that we designed and manufactured using our system.

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

Keywords: fabrication, data-driven methods, design

1 Introduction

We are approaching a time when ordinary people can fabricate their own 3D objects and products. However, to fabricate an object one first needs to design it. Currently, only experienced experts possess the knowledge and design skills to build complex, functional objects. This is because creating an object that can actually be fabricated involves more than just designing its shape. How should the parts be connected? Is the design stable and sturdy? Are the parts available for purchase or even affordable? In this work, we propose a data-driven method that addresses these and other design challenges, allowing non-experts to design and fabricate complex objects.

Data-driven methods have previously been used to make geometric design easier and therefore more accessible to non-experts. In the "modeling by example" approach, first presented by Funkhouser and colleagues [Funkhouser et al. 2004], new objects are constructed by assembling components of existing objects in a database. This allows even novice users to model complex 3D geometry. However, in creating *fabricable* designs, several challenges arise that have not been addressed in the previous research. First, all the components in the example database must be fabricable. Second, any manipulation applied to these components must preserve structure and manufacturability. Third, standard computer graphics techniques such as mesh blending cannot be applied to connect and assemble components. In order to combine parts, these parts must be accurately positioned, and real connectors (e.g., screws or hinges) must be used. The best choice of connectors will depend on the geometric, functional, and material properties of the object. Finally, the resulting model must be structurally sound, so that once it is built it will not topple or collapse.

In this work, we present an interactive design-by-example system for fabrication that addresses these challenges. To build the database of examples, we have worked with domain experts to create a representative set of example designs for several categories of objects. Each design is modeled using a commercial CAD software and is composed of a collection of standard parts that can be purchased from suppliers. It also includes all assembly details such as connectors and support structures, ensuring the object is fabricable. We automatically convert these specific designs to parameterized *templates* by extracting constraints and parameters from the models. This allows us to perform structure–preserving manipulations using both discrete and continuous parameters of the parts. The template representation is hierarchical and includes connectivity constraints between parts.

Using the dataset of templates and information extracted from them, we create an assembly-based modeling system for novice users. The user can pick and drag substructures from different designs and add them to a working model. The system guides the user through the *snapping* and *connecting* stages. *Snapping* involves automatically positioning the parts relative to each other, and selecting template parameters of the new parts to allow connectivity and alignment. *Connecting* involves automatically selecting the appropriate components and connectors that should be added to hold the parts together. Our system also includes a physics-based simulation component that can evaluate the stability of the composed model, highlighting unstable parts in the design. This relieves users from many tedious and complex tasks that are nevertheless necessary for feasible fabrication of the models, allowing them to concentrate on the creative process.

To the best of our knowledge, we are the first to propose a complete data-driven system for digital fabrication. In addition to the system, our work contains the following technical contributions:

- A hierarchical template representation for fabricable designs and a method for automatically converting fabricable designs to templates (Sections 3.2)
- An efficient method for *snapping* fabricable templates together (Section 4.4),
- An efficient method for *connecting* parts that guarantees manufacturability (Section 4.5).

To illustrate the generality of our data-driven method, we have used the system to design and fabricate a variety of different objects, from furniture to go-karts. We are also releasing a database of parameterized fabricable models, which we believe will be an invaluable resource for future work in this research area.

2 Related Work

Our work employs methods in data-driven modeling, shape manipulation, and fabrication-aware design.

Modeling by Example Shape collections have been widely used to allow data-driven geometric modeling. Modeling by example [Funkhouser et al. 2004] enables the design of new models by combining parts of different shapes from a large database. More recent work focuses on data-driven suggestions for adding details [Chaudhuri and Koltun 2010] and modeling [Chaudhuri et al. 2011]. Similarly, recombination of model parts has been used to expand databases [Kalogerakis et al. 2012; Jain et al. 2012] and repair low-quality 3D models [Shen et al. 2012]. Nevertheless, none of this research explores the fabrication aspect of data driven modeling. Creating models that can be physically realized adds a number of challenges to the modeling pipeline that are addressed in this work.

Template-based Shape Manipulations Many previously developed tools exist for the geometric editing of man-made shapes. Kraevoy et al.[2008] present a method for shape-aware resizing of man-made objects that can be non-uniformly scaled along three main axes. Our work is also related to the iWires system [Gal et al. 2009], which preserves structural relationships during editing using constrained non-linear optimization. Similarly, Zheng et al.[2011] segment a man-made shape and associate a controller with each component. The shapes of individual parts can be changed while preserving the structural constraints. The work by Xu et al. [2011] employs a similar strategy but uses an image as an input. Other shape manipulation methods that work with discrete variations (e.g., component repetitions) have been explored [Bokeloh et al. 2011; Lin et al. 2011; Bokeloh et al. 2012]. Finally, recent work [Ovsjanikov et al. 2011; Kim et al. 2013] explores the creation and use of parameterized templates in order to explore shape collections.

In our work, we create a parametrized template representation for manipulating shapes inspired by two classes of methods. The first class preserves global relationships [Gal et al. 2009; Zheng et al. 2011] but only considers continuous variations in the shape. The second class allows discrete variations [Bokeloh et al. 2012] but only accounts for local relationships. We combine these two ideas to construct models that both preserve global relationships such as symmetry *and* perform topological changes to preserve discrete regular patterns. Like Bokeloh et al. [2012], we construct a linear model that allows us to express the space spanned by all possible manipulations as a parameterized model. Our templates possess two additional qualities that are essential for our application. First, they follow a hierarchical tree structure that allows assembly of new models by composing substructures at different levels of the tree. Second, they encode information that guarantees fabricability. For example, we represent how parts connect to each other in the physical world, and we allow parts to be resized only if a corresponding physical process is possible. Our template representation is discussed in detail in Section 3.2.

Fabrication-aware Design Digital, personalized fabrication has garnered a lot of interest in the computer graphics community. The Plushie system by Mori and Igarashi [2007] allows non-experts to convert 3D models to physical plush toys. Saul et al. [2011] propose an interactive system for sketching chair models that can easily be fabricated. More recently, Chen et al. [2013a] and Hildebrand et al. [2012] have proposed systems to convert 3D models to simplified, fabricable designs consisting of interlocking planar pieces. Schwartzburg and Pauly [2013] extend these ideas by developing an interactive design system that employs optimization and structure analysis to provide instant feedback to users. Similarly, Umetani et al.[2012] build an interactive system for furniture design that is tightly coupled with a physically-based simulation in order to correct invalid designs and provide users with design suggestions. Finally, Lau and colleagues [2011] suggest a method for generating the parts and connectors needed to convert a 3D model into a physical object. This work focuses on furniture models and defines a formal grammar for IKEA tables and cabinets.

There has also been work in the computer-aided design community related to creating fabricable designs from user input. Roy et al. [2001] provide a natural language specification and iterative design process that transforms a simple functional specification into a detailed design. Chiou et al. [1999] use a small set of primitives and an accompanying matrix decomposition to accomplish the same task. Finally Gui and Mäntylä [1994] provide a sketch-based system for mechanical design.

The main difference between our approach and the previous work is that we are the first to propose a *data-driven* method for fabrication. In data-driven methods, production rules are implied by the dataset and they do not have to be explicitly distilled. It is this feature that motivates our data-driven approach. In our work, we illustrate the expressive power of data-driven methods by using the same algorithm to build furniture and go karts.

3 A Database of Fabricable Templates

Our system relies on a database of fabricable templates. The user can pick templates from the database, modify their parameters, and assemble them to create new designs. To build this database, we first collect fabricable models and then convert these designs to a hierarchical template representation.

3.1 Collecting Fabricable Models

To the best of our knowledge, no available repository of 3D models contains the necessary information for fabrication. We have therefore gathered a collection of fabricable models with the help of design experts—in this case, a group of mechanical engineers. The data is divided into two sections: an items catalog containing a list of commercial items, and a set of *designs* constructed by domain experts. Each part used in a design references a corresponding item in the catalog. This imbues our data with the unique property that all the designs can actually be manufactured. The data is available at *http://fabbyexample.csail.mit.edu*.

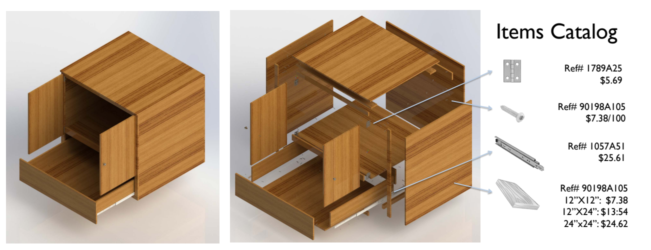

Figure 2: *An example of a fabricable object from our collection (left). Each design is detailed down to the level of individual screws, and each part maintains a reference to the items used from the items catalog (right).*

Items Catalog An *item* is a physical part that can be purchased from a variety of suppliers or manufacturers. The items catalog lists the available items along with all information required for their use during the design and fabrication process: their corresponding material type (e.g., wood, metal, glass), their price, their dimensions, and a 3D mesh representing their geometry.

The items within the catalog incorporate two additional pieces of information that are used in later stages of our method. First, each dimension of an item is labelled as either fixed or resizable. We only allow resizing of items if a corresponding physical process is possible. For example, we allow resizing of wooden components because they may be cut using available tools. This information is used during the design parameterization process. Second, each item maintains links to external suppliers (e.g., McMaster-Carr), allowing for easy sourcing during fabrication.

Set of Designs The design set contains a large number of manufacturable models (henceforth called *designs*) created using commercial CAD software (Solidworks). Each design is an assembly of parts, and the parts all contain links to the corresponding items in the items catalog (Figure 2). The parts are grouped into subassemblies in a hierarchical fashion, as is common in standard CAD tools. Once a design has been finalized, we build an associated connectivity graph, where nodes represent parts and edges indicate physical connectivity between them. We create this graph automatically based on the proximity of parts.

Designs often feature complex moving parts connected by mechanical joints. Such connections are used in various doors and drawers, and in complex moving objects like swings, wheels and steering assemblies. We rely on the experts to annotate their designs with this functionality if it exists. We store moving components in the standard way: as a hierarchy of joint transforms along with their respective joint types (prismatic, ball joint, hinge joint) and joint limits.

Domain experts also annotate parts that are purely structural (e.g., screws, hinges and brackets), henceforth called *connecting parts*. The *connecting parts* are separated from the *principal parts* of the design (e.g., shelves, legs, wheels) since the connecting parts will not be used explicitly in the design-by-example process. Instead, we relieve the user from the tedious task of specifying them by inferring and adding them automatically during the design process (see Section 4.5).

3.2 Parameterized Templates

Once we have the input collection of fabricable models, we create a template representation of the designs that allows structure-preserving manipulations and part recombination. First, we convert the designs into hierarchical parameterized representations which we call *templates*. Then, we augment this representation by incorporating information on how elements connect to each other in the physical space.

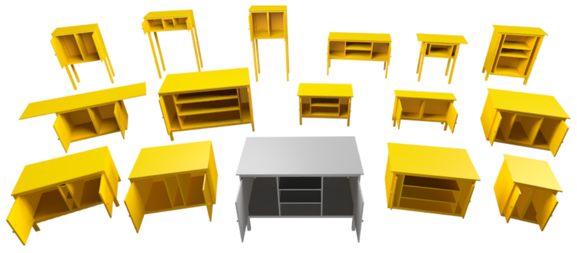

Figure 3: *An example of a template. The original design is shown in gray, and new designs generated by varying the template parameters are shown in yellow.*

Hierarchical Template Representation. We define a *template* as a part of a design that can be manipulated in a structure-preserving fashion. Templates provide a number of free parameters which can be used to manipulate associated geometry. An example of a single template is depicted in Figure 3. The figure illustrates the large amount of geometric variety that a single template can encode: in the figure, a cabinet becomes everything from a workbench to a nightstand.

More formally, a template at the i^{th} level of the template hierarchy T^i can be written as

$$T^i = \{\mathbf{q}^i, \mathcal{A}^i, F^i\} \tag{1}$$

where $\mathbf{q}^i$ are the degrees of freedom for the template; F^i is a deformation function that, given $\mathbf{q}^i$, computes new geometry; and $\mathcal{A}^i$ is the feasible space of $\mathbf{q}^i$, which is chosen to ensure that the geometry produced by a template remains fabricable and collision-free.

We convert each design to a template tree, following the hierarchical representation determined by the experts. For each leaf node, we explicitly define $\mathbf{q}^i$ and F^i, and we define $\mathcal{A}^i$ based on the set of constraints that act on $\mathbf{q}^i$. For the internal nodes, we specify $\mathbf{q}^i$ and F^i as the composition of the children nodes. The feasible set on an internal node can be defined as the intersection of the feasible sets of its children restricted by additional "coupling constraints" that bind multiple templates together.

Template Construction. Our method for automatically converting a design from the collection to a template comprises two steps. First, we select the leaf nodes and assign their degrees of freedom $\mathbf{q}^i$ and deformation functions F^i. Second, we analyze the semantic geometric (*geosemantic* [Shtof et al. 2013]) relationships between parts of our model in order to define structure-preserving constraints at each level of the hierarchy, thus determining $\mathcal{A}^i$ (see example in Figure 5).

We use the hierarchical structure of a design to guide the construction of the template tree. In most cases, we assign each part P^i in the design to be a leaf-node. We then choose $\mathbf{q}^i$ to be the 6-vector composed of the 3-dimensional position of the center of the part and the three axis-aligned scaling parameters. The deformation function F^i simply applies the prescribed scale and translation to P^i.

We also allow leaf nodes to represent repeating patterns of parts (Figure 4). We automatically search the design for repeating patterns and group them in a single leaf node. Although we can still represent $\mathbf{q}^i$ as the 6-vector that describes the scaling and translation of the pattern, the deformation function F^i is slightly more complex. Details of this process are given in Appendix A.

In our template model, a leaf-node in the template tree serves as a "least-fabricable-unit", the simplest single entity that can be constructed. Leaf-nodes play a crucial role in the remainder of our

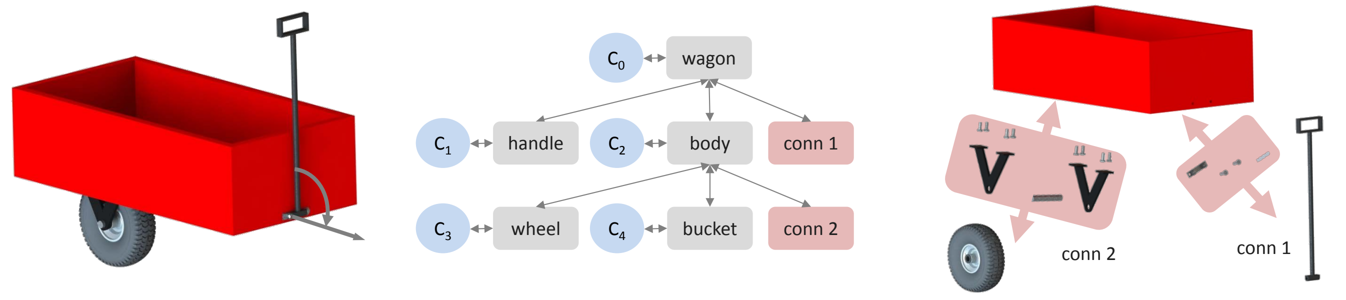

Figure 5: *From left to right: a design example of a toy wagon, the hierarchical template tree, and a visualization of the connections. The arrow on the handle indicates that this part has an articulation, namely that it can rotate along the depicted axis. The template tree includes the geosemantic relationships that are stored at each level of the hierarchy, C_0 to C_4 (shown in blue), as well as connections (depicted in red). The vizualization on the right illustrates the information contained in each connection node.*

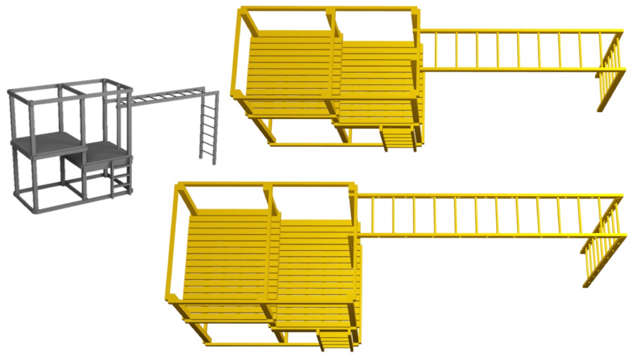

Figure 4: *A template with pattern elements. Upon resizing, both the number of floor planks and the number of rungs in the monkey bars change.*

algorithm, and are therefore referred to as *elements* to clearly distinguish them from the internal nodes of the template hierarchy. Elements that correspond to principal parts are called *principal elements*, while elements that correspond to connecting parts are called *connecting elements*.

To define $\mathcal{A}^i$, we constrain the space of template variations by extracting geosemantic relationships from the design and ensuring that they are preserved when the template parameters $\mathbf{q}^i$ are manipulated. Following the ideas described in [Gal et al. 2009; Zheng et al. 2011; Chen et al. 2013b], we take into account the following types of relationships between elements: concentricity, coplanarity, and symmetry. In addition, we consider relationships in the order of the elements. Preserving order relationships guarantees that elements do not penetrate each other or exchange position when the template parameters are modified.

We also extract geosemantic relationships guided by experts' annotations. First, we take into account the articulation information to ensure that all geosemantic relationships hold for all design configurations. Specifically, we consider the poses in which articulated parts reach their joint limits. Second, we take into account the physical properties of elements in the design using information from the corresponding items in the items catalog. For example, we constrain the scaling parameters of elements that are linked to items that cannot be resized in a certain dimension.

The geosemantic relations are stored in the hierarchy at the lowest internal node that is a parent of all the related elements. This allows the use of any sub-tree in the hierarchy as a template by itself since its defining relations are self-contained. Consider, for example, the toy wagon in Figure 5. The template leaf-nodes store scaling constraints on each element (C_1, C_3, and C_4). The body assembly stores the coplanarity and concentricity relationships between the wheel and the bucket (C_2). Finally, the root node stores the coplanarity relationships between the handle and the bucket (C_0). Notice that the handle has an articulation; therefore, we compute relationships in both the horizontal and vertical rest configurations.

We characterize geosemantic relationships as functions that act on the bounding box or *prominent planes* of each element. In most cases, we choose the six planes that define the bounding box to be the prominent planes. Since both the box and the planes are determined by $\mathbf{q}^i$, each geosemantic relationship can be expressed as a linear equality or inequality that constrains $\mathbf{q}$. For more details on how to find these relationships and express them as linear equations, please refer to Appendix B. Since we store the geosemantic relationships hierarchically, we can construct a linear system at each node T^i by aggregating the constraints of all its children. The feasible set $\mathcal{A}^i$ is then the set of all solutions to that linear system. Note that all of these relationships are computed automatically based on the geometry of the original design and the annotations on the input data.

Connections. Fabrication requires not only tracking abstract geometric constraints, but also understanding where and how elements connect to each other in the physical world. To accomplish this, we augment our template representation with nodes that keep track of the physical contact and connections between the principal elements. We represent these relationships as *connection* nodes. Connections include references to the data that will be manipulated and carried over when combining elements to compose a new model. These include

- the set of principal elements that are in contact (usually two but sometimes more),
- the set of connecting elements that are responsible for holding the principal elements together,
- the set of geosemantic relationships between the connecting elements and the principal elements, and
- a set of "soft" geosemantic relationships for placement of connecting elements (discussed below).

Grouping the elements into such structures is straightforward and can be achieved by directly analyzing the connectivity graph of the design described in Section 3.1. The set of "soft" relationships encode additional constraints on the connecting elements. At each connection, we compute the contact of the principal elements (the *contact patch*), and we add linear constraints to ensure that the dimensions and position of the connecting elements are preserved with respect to this contact patch . When the designer manipulates the template by changing the parameters of the principal elements, the parameters of the connecting elements are optimized using these additional relationships as soft constraints. This relieves the designer from the tedious task of manipulating each connecting element individually.

Like geosemantic relationships, connections are stored on the lowest internal node that is a parent of all the principal elements they reference. In the example of Figure 5, we have two connections. One connects the handle to the bucket and is stored in the root node of the toy wagon; the other connects the bucket to the wheel and is stored in the internal node that groups the body assembly.

Constructing the template hierarchy in this manner ensures that each node in the hierarchy is a complete representation that depends only on its children. Therefore, the database of templates that we construct includes not only full models of the original design (root nodes) but also all the other nodes in each hierarchy, representing parts and sub-structures. The result is a much richer database that supports the design-by-example mechanism of assembling new models by composition of templates representing parts at various levels.

4 Modeling

In this section we outline the design workflow of our system and describe in technical detail the system's main operational features.

4.1 Design Workflow

Our system is based on the design-by-example workflow [Funkhouser et al. 2004; Chaudhuri and Koltun 2010]. Figure 6 illustrates our user interface. Icons that link to components of the database are displayed on the left; and the canvas on the right is used to design a new model, henceforth called the *working model*. Users compose parts by dragging them onto the scene, and they can also remove selected parts at any level of the hierarchy.

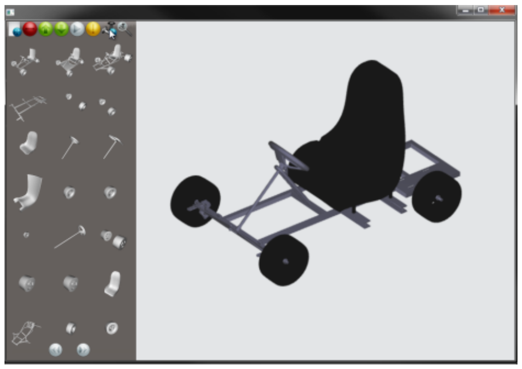

Figure 6: *The user interface. Icons that link to components of the database are displayed on the left, and the modeling canvas is on the right.*

As we explained in Section 3.2, the components of the database are hierarchical parametrized templates. This allows us to compose models from different designs at different levels, i.e., we can add and remove small components (e.g., a single shelf), medium components (e.g., a drawer), and large complex ones (e.g., an entire cabinet). When creating a new model, we can either start from scratch or work from an existing design.

The user can vary the shape of any component in the database by manipulating its template parameters (see Section 4.2). Our system handles composition to ensure that the working model, like the components of the database, is a hierarchical, parametrized template. Therefore, users can continue to manipulate parameters after components are assembled.

Composition in a fabrication-aware system is difficult because one cannot merely apply simple geometric operations to merge parts. To combine two substructures, the substructures must be perfectly fitted and aligned, and appropriate connecting elements (e.g., screws and hinges) need to be added between them. This process is not only tedious, but sometimes impossible for users who lack the necessary expertise.

Our system addresses these difficulties with two crucial operational features. First, when a user drags in a new component and drops it onto the scene, the system automatically uses information stored in the database to adjust the component's position and size so that it fits and aligns with the working model. We call this procedure *snapping*. The snapping operation optimizes the component's position and size based on the position in which the user dropped the component and the component's current dimensions. If the user is not satisfied with the snapped configuration, the user can edit the component (through template manipulations) and drag it around the scene, and the snapping procedure will then automatically compute the component's new optimal position and size.

Second, our algorithm automatically retrieves new connecting elements that attach the added component to the working model. This is achieved by searching the database for similar examples of connections. During this process, we compute new geosemantic relationships between the added template and the working model. Both the new connecting elements and the new geosemantic relationshops are added to the hierarchy of the working model together with the added component. We call this process *connecting*.

4.2 Template Manipulations

Allowing users to manipulate template parameters adds variety to the designs (see Figure 3) and allows fitting parts of different sizes. Adjusting template parameters modifies a design's shape and dimensions, but preserves its overall structure. This method of template manipulation guarantees that the composed models are fabricable.

We allow template parameters to be manipulated at all levels of our hierarchical structure. The user can select elements (leaf nodes) by clicking on them, and then traverse up the hierarchy to select internal template nodes. When a template node is selected, controls for scaling and translation are revealed (see Figure 7). At each level of the hierarchy, the controls act on the bounding box of the selected template. Therefore, the user can make higher-level changes by selecting internal nodes and make more detailed adjustments by selecting leaf nodes.

Template parameter manipulation is not an unconstrained procedure. A given template is restricted to the feasible space $\mathcal{A}$ stored at the root node of the hierarchy. As outlined in Section 3.2, we define the parameters of the root node $\mathbf{q}$ as the stacked vector of all the children $\mathbf{q}^i$. We can then represent all linear, geosemantic constraints (bilateral and unilateral) in the standard form: $\mathbf{A}\mathbf{q} = \mathbf{b}, \mathbf{G}\mathbf{q} \leq 0$. We augment $\mathbf{A}$ with constraints that fix the center of the model in order to prevent translation of the edited template.

To create the controls we use six functions $\mathbf{c}_j$, such that the $\mathbf{c}_j^T\mathbf{q}$ correspond to the center and dimensions of the axis-aligned bounding box. When manipulating leaf nodes, the $\mathbf{c}_j$ are standard basis vectors, since $\mathbf{q}^i$ defines the axis-aligned bounding box of the element. For larger substructures represented by internal nodes, we first compare the bounds of the children elements to determine which ones constrain the bounding box of the substructure in each dimension. We can then use these elements to explicitly determine the functions $\mathbf{c}_j$.

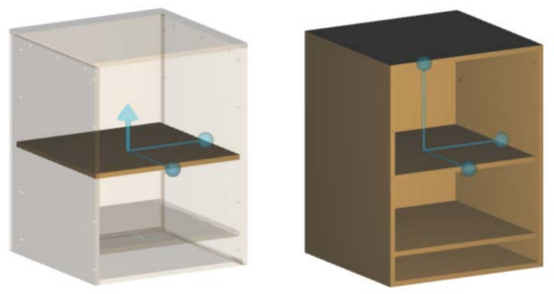

Figure 7: *An illustration of how template variations can be explored in our system. The arrows control translation, while spheres control scaling. On the left, we show the controls on a leaf node of the hierarchy; on the right, we show the controls on an internal node. During manipulation, elements on the selected node are represented in full color, while the others become semi-transparent. Notice that constrained degrees of freedom are hidden. For example, the user is unable to change the thickness of the shelf, since our items catalog states that planks of wood can be cut only in two directions.*

As the user drags a control j, we calculate the new template configuration by solving the simple quadratic program

$$\mathbf{q}^* = \underset{\mathbf{q}}{\operatorname{argmin}} \|\mathbf{c}_j^T\mathbf{q} - (\mathbf{c}_j^T\mathbf{q}_{\text{current}} + \delta)\|^2 + \alpha\|\mathbf{q} - \mathbf{q}_{\text{current}}\|^2 \quad \text{s. t.} \quad \mathbf{Aq} = \mathbf{b}, \mathbf{Gq} \leq 0 \tag{2}$$

where $\mathbf{q}_{\text{current}}$ are the template parameters in the current state and δ determines the amount of dragging. The second term penalizes large changes in parameter value. Accordingly, α is chosen to be less than one to give more importance to the first term. To reduce cluttering, we hide controls that manipulate a completely constrained scaling direction. We determine whether the j^{th} control is constrained by checking if $\mathbf{c}_j$ and $\mathbf{A}$ are linearly dependent.

4.3 Composition

We compose new designs by removing and adding components to the working model. To remove a part, the user explores the working model's hierarchy and selects a substructure of the model. The corresponding node is excised from the template tree. The hierarchical nature of our templates is leveraged to quickly remove all connections and geosemantic relationships incident on the deleted structure. This frees the working model of unnecessary constraints and connecting elements that serve no purpose in the new design.

As mentioned earlier, adding new parts is more difficult; therefore, our system assist the user in two ways. First, when a user chooses a template T^A, we adjust the dimensions and placement of T^A to *snap* it to its position. Second, we automatically compute new constraints and find the elements that *connect* T^A to T^W.

To define both snapping and connecting, we first examine the original design T^D from which the part T^A originated. We examine where and how T^A connected to T^D, and we try to use that information to align and connect T^A to T^W. If the information we have is not sufficient, we search the rest of the database for similar connections. In the following two subsections, we explain the snapping and connecting operations more fully.

4.4 Snapping

Snapping the additional template T^A to the working model involves computing a new template configuration $\mathbf{q}^A$, which is optimized based on the user's current positioning and dimensions of the part (i.e., the current state of the template $\mathbf{q}^A_{\text{current}}$). If the user is not satisfied with the solution, she can continue to change T^A's position or its parameters to bring them closer to the desired configuration, and the system re-optimizes $\mathbf{q}^A$ given the new current state.

To find an optimal template configuration, we try to identify constraints that $\mathbf{q}^A$ will have to satisfy when T^A is added to the working model. We do this in two steps. First, we analyze how T^A connects to the original design T^D and try to find constraints on $\mathbf{q}^A$ that would allow T^A to connect to T^W in a similar manner. Second, we search for prominent planes on T^A that are sufficiently close to planes in T^W and try to align them. In what follows we discuss each of these steps in detail.

Constraints Based on Original Design. To create constraints based on the original design, we look at the elements (leaf nodes) of T^D that are connected to T^A and extract the coplanarity relationships between these elements and T^A. As mentioned in Section 3.2 (and detailed in Appendix B), coplanarity relationships constrain prominent planes—in this case a plane of T^A with respect to a plane of T^D. To create an analogous constraint between T^A and T^W, we need to find a plane on T^W that has the same normal as the one in T^D. Since there might be many planes in T^W that satisfy this requirement, we take the K ones that are closest to T^A in the current configuration.

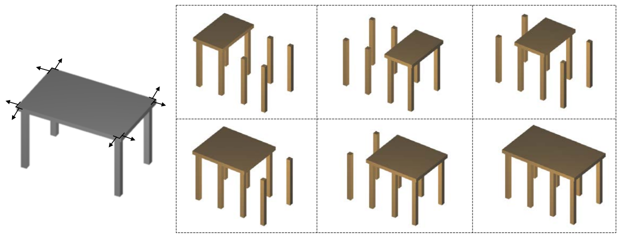

Figure 8: *An example of snapping to constraints. We add a tabletop T^A to the working model T^W containing eight legs (right). The coplanarity constraints on the original design T^D that contained T^A are represented by the normals of the corresponding planes (left; we show only the vertical ones). The feasible snapping configurations for $\mathbf{q}^A$ are shown on the right. The system will choose one of these configurations: its choice will depend on the scale parameters and the position on which the user places the tabletop.*

Using the template representation, we can write each connectivity relationship as a linear constraint $\mathbf{a}\mathbf{q}^A = b$, where b depends on the selected plane in T^W. We then extract a subset of the linearly independent constraints $\mathbf{a}$ that is also not restricted by the feasible set $\mathcal{A}^A$ of T^A. Notice that the number R of constraints in this subset cannot exceed the number of degrees of freedom of T^A, which tends to be small (usually six). Since b can be chosen in K different ways, we end up with a set of $N = K^R$ possible constraints $\mathbf{A}\mathbf{q}^\mathbf{A} = \mathbf{b}_n$ that restrict the template parameters of T^A. For each of these possible constraints, we compute the optimal $\mathbf{q}^A$ by solving the following least squares problem:

$$\min_{\mathbf{q}^A} \|\mathbf{q}^A - \mathbf{q}^A_{\text{current}}\|^2 \quad \text{s. t.} \quad \mathbf{A}\mathbf{q}^A = \mathbf{b}_n, \;\; \mathbf{q}^A \in \mathcal{A}^A \tag{3}$$

We then select the constraint matrix $\mathbf{A}\mathbf{q}^A = \mathbf{b}_{\tilde{n}}$ whose optimal solution has the smallest cost. The value of K is chosen depending on R to guarantee that $N = K^R$ does not become too large. Typically, we set $K = 4$.

Alignment of Prominent Planes In many cases, the constraints of the original design are not enough to position the part in the working model. For example, in the working model shown in Figure 8, the table legs were created by composing and snapping two sets of four legs, which are aligned even though they are not connecting and there is no resemblance to such a combination in the original design of either set. To find additional alignment constraints, we select the set of planes in $\mathbf{T}^A$ that are not restricted by $\mathbf{A}\mathbf{q}^A = \mathbf{b}_{\bar{n}}$. For each of these planes, we find the closest parallel plane in the working model, and we align them if the distance between them is smaller then a certain threshold. This gives us a new $\mathbf{q}^A$ that will be used to connect the additional template to the working model. For objects with functionality (i.e., several configurations), we construct prominent planes corresponding to all main rest configurations (see Appendix B). This guarantees that functional objects snap so that they align to the working model in all rest configurations (see Figure 9).

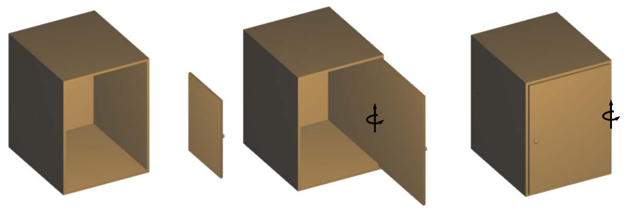
Figure 9: *An illustration of snapping for functional objects. When a door is added to the side of a cabinet, it automaticaly rescales so that, when shut, it will align with the oposite side. At left, a door is added to the working model. From left to right: the added door before snapping, the snapped configuration, and a visualization of the snapped configuration when the door is closed. The rotation axis of the articulations is depicted by the arrows.*

4.5 Connecting

Once the user is satisfied with the fitted part, they invoke the *connecting* method. Connecting automatically places T^A in the hierarchy of the working model and adds the appropriate connections and geosemantic relationships. Although the parameters $\mathbf{q}^A$ may still vary, the snapping result returns an approximate configuration of T^A. We use this information to find the elements in the working model T^W that should be connected to elements T^A based on proximity. We call these *linked elements*. We then search the database for a connection that can be used to connect each pair of linked elements. After these connections are selected, a final composition step is performed to create a new working model that preserves the hierarchical structure and is correctly parametrized. We discuss each of these steps in the following paragraphs.

Searching for Connections As in our snapping algorithm, we first search for connections in the original design T^D. We consider all the connections in T^D that connect T^A to elements in $T^D \setminus T^A$, and we try to transfer these connections to the working model. Transferring involves matching principal elements in $T^D \setminus T^A$ to elements in T^W. Since we have the $\mathbf{q}^A$ that resulted from the snapping algorithm, we can first fit T^D by finding $\mathbf{q}^D$ that minimizes $\|\mathbf{S}\mathbf{q}^D - \mathbf{q}^A\|$, where $\mathbf{S}$ is a matrix whose columns are the standard basis vectors that correspond to the indices of $\mathbf{q}^D$ that refer to $\mathbf{q}^A$. Once the fitting is done, we can use a standard distance function on the bounding boxes of each element to retrieve the closest matches. Finding the matches creates a candidate connection that acts on $T^W \cup T^A$.

Once we have candidate connections, we need to determine if they can be used in the composed design. A connection contains the set of relationships between the elements it references (see Section 3.2), which we can represent by the feasible set $\mathcal{A}^C$. We cannot add a connection if the feasible set $\mathcal{A}^C \cap \mathcal{A}^A \cap \mathcal{A}^W$ is empty. If the feasible set is nonempty, we find the configuration of the template $T^W \cup T^A$ in this set that is as close as possible to the snapped configuration. We do this by solving a quadratic program:

$$\begin{aligned} \min_{\mathbf{q}^A,\mathbf{q}^W} & \|\mathbf{q}^A - \mathbf{q}^A_{\text{current}}\|^2 + \|\mathbf{q}^W - \mathbf{q}^W_{\text{current}}\|^2 \\ \text{s. t.} \quad & \mathbf{q}^A \in \mathcal{A}^C \cap \mathcal{A}^A, \quad \mathbf{q}^W \in \mathcal{A}^C \cap \mathcal{A}^W \end{aligned} \tag{4}$$

We allow the error to be larger than zero because, in many cases, principal elements need to be slightly shifted or scaled in order to insert connecting elements (see Figure 10). Nevertheless, in order to guarantee that we do not drift too much from the user's design, we use a connection only if the error of this minimization is smaller then a fixed threshold.

When not all connections are found in T^D, we extend the search to all templates in the database. We use a priority based on a similarity metric that compares connections by evaluating: 1) similarity of principal components, and 2) the relative distance between them. We compare principal components by first making sure that the materials match and then measuring the distance between the sizes of the bounding boxes. Since the dimensions of the components can vary according to template manipulations, we would like our similarity metric to encode a weighted average of the amount of template manipulation necessary and the final distance between the two components. We accomplish this by giving extra weight to dimensions that are constrained, which naturally encode the distance between the models after fitting.

We compare relative distances between elements using the distance between the bounding planes of the axis-aligned bounding boxes of each element. For each dimension, we compare both bounding planes against each other—a total of four evaluations. By doing so, we encode not only coplanarity relationships but also order. This is important because parts often need to have an intersecting area in order to be connected.

We pre-compute the descriptors for all the connections in the database, so that a simple weighted distance function efficiently retrieves the closest candidate connections at runtime. After the candidate connections are retrieved, we evaluate them using the method described above. We try only the K closest connections: if none of them pass the evaluation, we refrain from adding a connection and warn the user that no connection was found. Typically, we set $K = 5$.

Final Composition Once we retrieve the set of edges, we are ready to generate the new working model $T^{\bar{W}}$ that incorporates T^A into T^W. We place T^A into the hierarchy of T^W by adding it as a sibling to the lowest node that groups the elements that connect to T^A. We add all the connections found by transferring the connecting elements to the working model and adding the relationships between them and the elements of $T^{\bar{W}}$. These relations constrain $\mathbf{q}^A$. Once these constraints are built, we optimize $\mathbf{q}^{\bar{W}}$ so that it is as close as possible to the current snapped configuration (solving a least squares problem). This may effect minor changes in the parameters of T^A. For instance, the parameters may change to allow connecting elements to fit between parts, as in Figure 10. Finally, we find and add geosemantic relationships between elements of T^A and T^W automatically in the same manner in which we built the original templates (see Section 3.2).

4.6 Physical Tests

The template manipulation and composition procedures guarantee that the objects modeled by our system can actually be fabricated by assembling parts purchased from suppliers. They assure that the manufactured objects will have the same *appearance* as the virtual design, but nothing can be said about the way the object will *behave* in the physical world. Therefore, a necessary step in modeling for fabrication is to examine the real-world physical behavior of the working model.

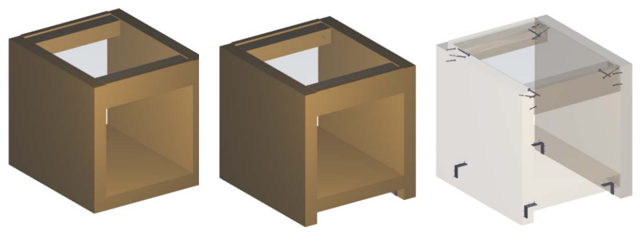

Figure 10: *An example of changing parameters to fit connectors. From left to right: the bottom shelf snapped to the bottom of the table, the resulting configuration of the model after the connecting step, and the vizualization of the connectors (principal elements are made semi-transparent). Notice that, in order to connect the bottom shelf to the table legs, the system raises the shelf above the ground to leave room for l-brackets.*

Many possible "product testing" tools can be used to verify a design's safety and feasibility. In this work we illustrate this by performing rigid-body stability analysis on the model. We assume that the working model is in static equilibrium inside a (potentially moving) rigid body frame. We model each principal element in the model as a rigid body. We exclude connecting elements from the analysis due to their sheer number and their negligible mass. Whenever the user checks the stability, we perform a single time-step dynamic simulation using the staggered projections algorithm [Kaufman et al. 2008] for resolving frictional contact, and then examine the forces acting on each component. We label elements on which the acting forces are balanced as *stable* and color them green. All other parts are considered unstable and colored red. The user can then take appropriate action to correct stability problems in the design.

5 Results and Discussion

In this section we discuss the main steps in our method and show a variety of designed and fabricated results.

5.1 A Database of Fabricable Templates

With the help of domain experts, we have built what we believe to be the first open collection of fabricable designs. It typically takes experts many hours to design a complete model using a commercial CAD software package. The simplest model in our database took approximately one hour to design, and the most complex (the go-kart) took three months. In this context, the time that is required to add the few annotations described in Section 3.1 (between 10 to 20 minutes) is almost negligible. More time is needed to create the items catalog: most of that time is spent finding suppliers for each of the items. But designers *always* need to find suppliers if they want to manufacture their models. Also, creating the items catalog can be viewed as a preprocessing step because, once completed, the items catalog can be reused for subsequent designs.

One of the key features of our method is the use of hierarchical parametrized templates that are fabrication-aware. By allowing only shape manipulations that correspond to template parameter variations, we are able to generate a bill of materials for every model designed in our system. This allows us to directly manufacture models that are created by our tool, as shown later in this section. Another important aspect of our template representation is that it handles the connecting elements differently from principal parts. While principal parts have degrees of freedom that have to be specified by the user, the parameters for the positioning and scaling of connecting parts are completely determined by the hard and soft constraints on the connections. This relieves the users from the tedious task of dealing with connecting parts.

5.2 Modeling

Figure 11 depicts a few results built using our tool. We indicate in the figure the number of individual parts in each design. Observe that even models that appear to be simple are composed by over one hundred parts. It would take an expert from one to four hours to build each of these models with commercial CAD software. However, using our system, users with no expertise in mechanical engineering were able to create these designs in less than twenty minutes.

In Figure 11, we highlight the different templates that were added to the model using different colors. Notice how the number of elements in each color-coded template varies. This illustrates how the users can explore the hierarchy by composing parts using smaller or larger substructures.

While the snapping and connecting steps of our system are responsible for the speed in which users can create such complex objects, the template manipulation feature helps to add diversity to the models. By adding new geosemantic relationships each time parts are added to the working model, we guarantee that the working model maintains the template representation which allows structure–preserving manipulations. Figure 12 shows an example of how the user can continue to explore the space of template variations of a composed model.

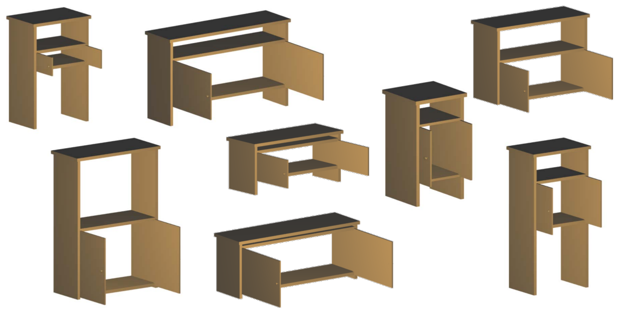

Figure 12: *An example of different manipulations of a working model after it has been composed from multiple templates.*

5.3 Fabrication

We tested the full data-driven fabrication pipeline to build four designs, illustrated in Figures 1 and 13. These models were created by combining parts from multiple designs, also shown in the figures. The output of the system is a comprehensive bill of materials that is generated by looking up the items for each part in the items catalog. Then, using the information provided by the external supplier, we could easily order the items and then assemble them. We can also minimize the total cost of materials by grouping together and combining items – for instance, by cutting many wood elements from a few pieces of stock material.

5.4 Limitations

As with all data-driven methods, the main limitation of our system it that we are restricted to the designs in the database. If a connection is not in an example, we are not able to retrieve it. For example, we cannot connect a go kart wheel to a table because go kart wheels attach to metal axles that do not resemble any part of a table. The only way to solve this problem would be to add to the database a design that has a similar type of wheel connected to some kind of wooden frame.

Another limitation is that we have so far only dealt with simple mechanical models. Although we can handle some functionality like drawers, cabinet doors, and wheels, our system cannot handle

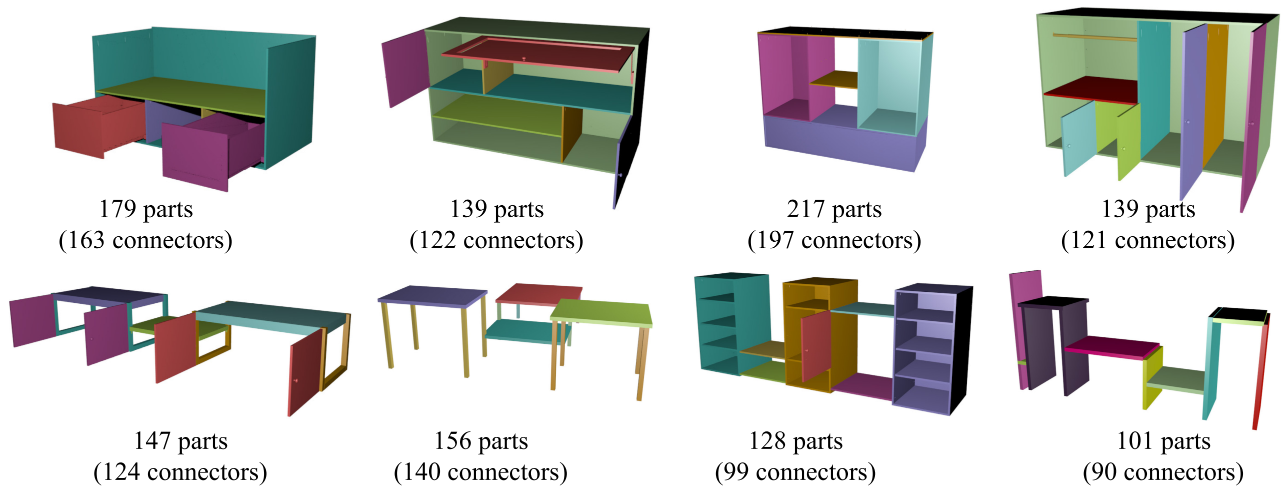

Figure 11: *Examples of models designed using our system and the number of individual parts they comprise. Different colors indicate the different parts that were added to the model.*

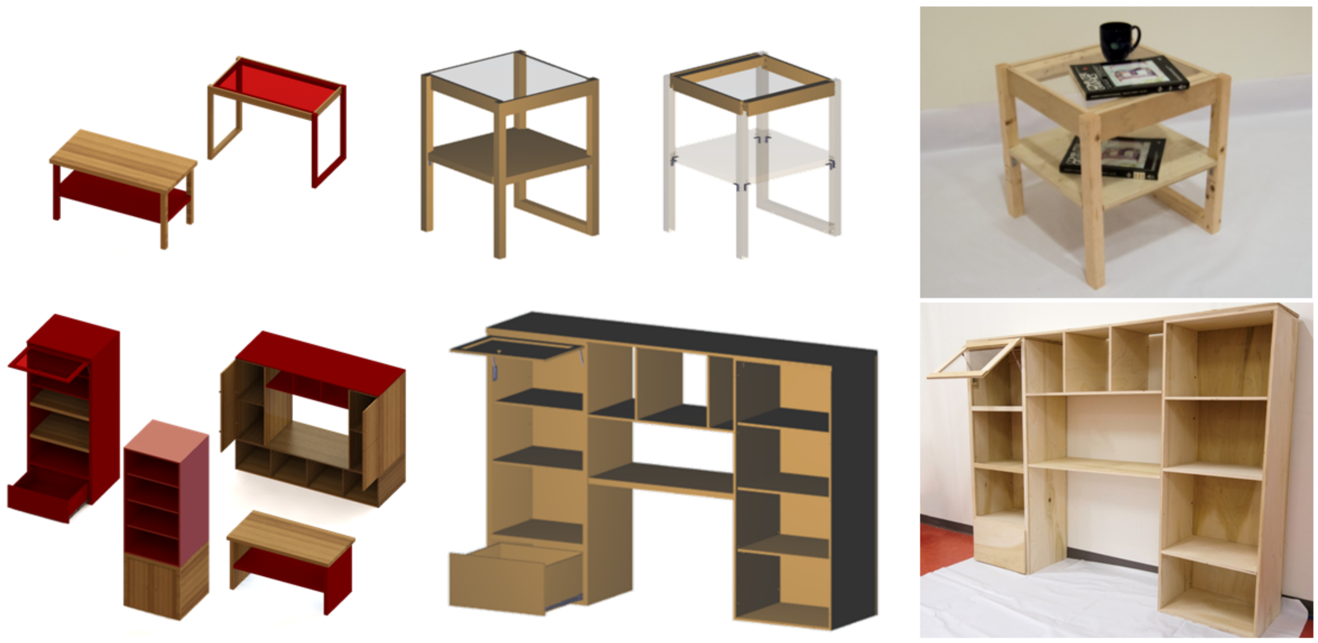

Figure 13: *From left to right: input designs, models created using the system, and fabricated results. We highlight the connecting elements on the first model by making all principal elements semi-transparent.*

more complex dynamic components such as electronics. In the go kart examples shown in the teaser, we were able to compose wheels (adding bearings), the seat (adding connecting planks with bolts) and the steering (adding in tierods). However, we could not interchange motors and controls, and so we considered them as part of the frame assembly of the go kart.

The templates themselves are limited in two ways. First, we choose to use a linear representation of our templates, because this allows for solving for parameter constraints using simple linear and quadradic programs, which can be efficiently computed. Though this computational efficiency is essential in an interactive system, the linearity condition constrains our manipulations to scaling and translation of parts, since rotations are non-linear. Second, we base most of our computations on coplanarity relationships between prominent planes. This works well for many man-made models, but our system would have difficulties with models that have a more curved or complex geometry.

6 Conclusion and Future Work

In this paper we have presented what we believe to be the first complete data-driven system for digital fabrication. Our algorithm successfully leverages a database of parameterized fabricable templates, allowing casual users to design models that can be physically realized. The output of our algorithm is comprehensive in that it provides a list of all parts necessary for construction, as well as a detailed bill of materials that lists where parts can be purchased and the total cost of construction. We have demonstrated the power of our method by fabricating different models. We have shown the scope of the data-driven method by applying the same algorithm to fabricate furniture and go karts.

We believe that this work, together with the database we are releasing, will spur interesting future work. For example, it would be interesting to present relevant components to the user during the modeling session. Chaudhuri et al. [2011] propose a probabilistic model that suggests components based on style and geometrical semantics. In the context of fabrication, such suggestions should also include metrics such as construction time, required tools, and currently available materials. We could also use this database and composition method to automatically synthesize new fabricable models by finding plausible combinations of components, as accomplished for virtual shapes by Kalogerakis et al. [2012]. Also, it would be interesting to output assembly instructions along with the bill of materials. Though the output of our system could be used as input to the automatic algorithm proposed by Agrawala et al. [2003], it would be interesting in the future to add assembly information to the catalog and use this data to automatically generate instructions.

One of the most under-explored elements of our pipeline is the use of physical simulation. Though we perform rigid-body stability analysis, there are many other properties that could be investigated. Future work could test alternate failure modes (such as strength of joints and stress distributions in objects) and check for fracture and potential future areas of material fatigue. Furthermore, while our database contains fabricable objects, much more information could be stored to make them truly physical entities. Extending the database to include usage data such as the typical forces an object encounters in the world would allow us to more rigorously validate a design's durability. Finally, because the items in our database are continually reused, there are opportunities to leverage preprocessing to dramatically accelerate our physics computations.

Finally, it would be interesting to test our algorithm for other categories of objects such as architecture and clothing, or for more complex mechanical systems. Since the annotations are simple and the template generation method is automatic, adding new models to the database is straightforward. In the future, we would like to create a web-based environment with a verification system to allow for the crowd-sourcing of database creation. We feel that a data-driven approach is the most general method for design and fabrication, and this paper represents an initial step in the exploration of such systems. We believe that, in the future, data-driven fabrication will allow us to design and manufacture a plethora of user-designed physical objects, from furniture to vehicles to robots, and beyond.

A Defining the mapping function F

In this appendix we discuss how we define our mapping function F^i (Equation 1) for elements which contain multiple parts that form a discrete regular pattern (e.g., an array of screws or a row of wooden planks).

We select regular patterns by searching for identical parts that have uniform inter-component spacing in one or two dimensions. Parts are said to be identical if they have the same corresponding item and the same dimensions. We perform the search by first grouping all identical parts and then extracting all the non-intersecting subsets of that group which form a regular pattern. We select the subsets in order of number of parts (highest first) to guarantee that the non-intersecting rule does not prevent us from extracting the larger sets.

Following the notation of Bokeloh *et al.* [2012], we consider only translational patterns. One-dimensional patterns are parameterized by $(\mathbf{o}, l, \mathbf{p})$, where $\mathbf{o}$ is the center of the first part (with respect to the bounding box of the element), l is the number of parts, and $\mathbf{p}$ is the generator translation. Two-dimensional rectangular patterns can be represented in the same manner.

Given $\mathbf{q}_i$, the deformation function F_i calculates the new optimal values of l and $\mathbf{p}$, which we call $\bar{l}$ and $\bar{\mathbf{p}}$, respectively. We choose $\bar{\mathbf{p}}$ to be as close to $\mathbf{p}$ as possible. In the case of connecting elements, we are conservative when computing $\bar{l}$ in order to guarantee manufacturability. However, we ensure that $\bar{\mathbf{p}}$ does not shrink to the point that parts intersect. Finally, if the parts have resizable dimensions in the directions of $\mathbf{p}$, we scale them according to $\bar{\mathbf{p}}$.

B Geosemantic Relationships

In this appendix we show how to extract geosemantic relationships and represent them as linear constraints. Consider the table in Figure 14 (in 2D for simplification). It consists of three elements, namely, the tabletop and two legs. The parameters of each element are $\mathbf{q}^i = [p_x^i, p_y^i, \Delta_x^i, \Delta_y^i]$, which correspond to positions and sizes in each dimension. Rules for coplanarity will stipulate that the bottom of the tabletop must coincide with the top of the legs, i.e., $p_y^{Top} - \frac{\Delta_y^{Top}}{2} = p_y^{Leg1} + \frac{\Delta_y^{Leg1}}{2} = p_y^{Leg2} + \frac{\Delta_y^{Leg2}}{2}$. Rules for concentricity constrain the center of the bounding boxes in each dimension. In rules for order, we simply replace the equality by an inequality.

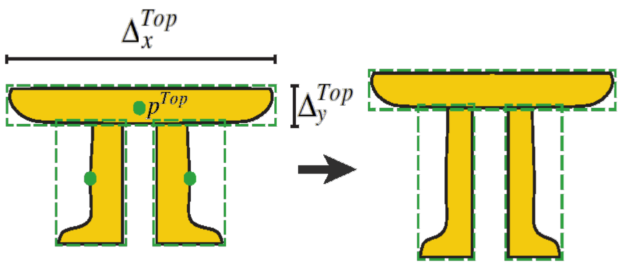

Figure 14: *We show a simple 2D table consisting of three parts, a top and two legs (Leg1 and Leg2). Each part is contained within a single element for which the $\mathbf{q}^i$ are the positions (x, y) and sizes $(\Delta x, \Delta y)$ of the bounding box of the part.*

Notice that the legs have a reflective symmetry. To write this relationship as a function of $\mathbf{q}$, we need to find a third element that has a center on the symmetry plane. In this case, we observe that this is true for the tabletop. Hence, if $\mathbf{n}$ is the normal of the symmetry plane (in the example, $n = [1\ 0]^T$), we can write down symmetry relationships as $\frac{(p^{Leg1}+p^{Leg2})^T\mathbf{n}}{2} = (p^{Top})^T\mathbf{n}$. (In the example, $p_x^{Leg1} - p_x^{Top} = p_x^{Top} - p_x^{Leg2}$.)

While concentricity and symmetry constraints are written directly as functions on the bounding box, we define coplanarity relationships on the prominent planes of the model. In most cases, we use the six bouding planes as the prominent planes. However, in elements with more complex geometry, such as the go kart frames, we extract additional planes to describe relationships (for example, the plane on the axle that connects to the wheel). In this implementation we have manually annotated such prominent planes, but one could use simple geometry processing tools to infer them automatically.

We also create additional prominent planes in the case in which we have functional elements that assume multiple rest configurations. In this case, we create planes for every rest pose. We consider the centers of the joints $\mathbf{q}_c$. For every resting pose j, we can write the variables of the transformed element as linear combinations of $\mathbf{q}_i$ and $\mathbf{q}_c$. By adding $\mathbf{q}_c$ to the $\mathbf{q}$ vector, we can write $\mathbf{q}_i^j = \mathbf{V}\mathbf{q}$ and proceed to add constraints in the standard linear notation described above.

Acknowledgements

The authors would like to thank Ilya Baran for his helpful discussions, Katie Bartel for implementation of user interface, and Baker Logan, Keneth Pinera, Helena Wang, Nilu Zhao, Kendall Helbert, Molly Donalson, Saul Lopez, Alexxis Isaac, and Jackson Wirekoh for designing the models in the database. This research was funded by NSF grant 1138967. Ariel Shamir is partly supported by the Israel Science Foundation (grant no. 324/11).

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 Transactions on Graphics 22*, 3, 828–837.

BOKELOH, M., WAND, M., KOLTUN, V., AND SEIDEL, H.-P. 2011. Pattern-aware shape deformation using sliding dockers. *ACM Transactions on Graphics 30*, 6, 123:1–123:10.

BOKELOH, M., WAND, M., SEIDEL, H.-P., , AND KOLTUN, V. 2012. An algebraic model for parameterized shape editing. *ACM Transactions on Graphics 31*, 4.

CHAUDHURI, S., AND KOLTUN, V. 2010. Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics 29*, 6, 183:1–183:10.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L. J., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3d modeling. *ACM Transactions on Graphics 30*, 4, 35.

CHEN, D., SITTHI-AMORN, P., LAN, J., AND MATUSIK, W. 2013. Computing and fabricating multiplanar models. *Computer Graphics Forum (Proceedings of Eurographics 2013) 32*, 2.

CHEN, T., ZHU, Z., SHAMIR, A., HU, S.-M., AND COHEN-OR, D. 2013. 3sweep: Extracting editable objects from a single photo. *ACM Trans. Graph. 32*, 6, 195:1–195:10.

CHIOU, S.-J., AND SRIDHAR, K. 1999. Automated conceptual design of mechanisms. *Mechanism and Machine Theory 34*, 3, 467 – 495.

FUNKHOUSER, T. A., KAZHDAN, M. M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. P. 2004. Modeling by example. *ACM Transactions on Graphics 23*, 3, 652–663.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. Iwires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics 28*, 3.

GUI, J.-K., AND MÄNTYLÄ, M. 1994. Functional understanding of assembly modelling. *Computer-Aided Design 26*, 6, 435 – 451.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. crdbrd: Shape fabrication by sliding planar slices. *Computer Graphics Forum (Proceedings of Eurographics 2012) 31*, 2.

JAIN, A., THORMÄHLEN, T., RITSCHEL, T., AND SEIDEL, H.-P. 2012. Exploring shape variations by 3d-model decomposition and part-based recombination. *Comp. Graph. Forum (Proc. Eurographics 2012) 31*, 2.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics 31*, 4.

KAUFMAN, D. M., SUEDA, S., JAMES, D. L., AND PAI, D. K. 2008. Staggered projections for frictional contact in multibody systems. *ACM Trans. Graph. 27*, 5 (Dec.).

KIM, V. G., LI, W., MITRA, N. J., CHAUDHURI, S., DIVERDI, S., AND FUNKHOUSER, T. 2013. Learning part-based templates from large collections of 3d shapes. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2013)*.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics 27*, 5, 111:1–111:9.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3d furniture models to fabricatable parts and connectors. *ACM Transactions on Graphics 30*, 4, 85.

LIN, J., COHEN-OR, D., ZHANG, H., LIANG, C., SHARF, A., DEUSSEN, O., AND CHEN, B. 2011. Structure-preserving retargeting of irregular 3d architecture. *ACM Transactions on Graphics 30*, 6, 183:1–183:10.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. *ACM Transactions on Graphics 26*, 3, 45:1–45:8.

OVSJANIKOV, M., LI, W., GUIBAS, L. J., AND MITRA, N. J. 2011. Exploration of continuous variability in collections of 3d shapes. *ACM Transactions on Graphics 30*, 4, 33.

ROY, U., PRAMANIK, N., SUDARSAN, R., SRIRAM, R., AND LYONS, K. 2001. Function-to-form mapping: model, representation and applications in design synthesis. *Computer-Aided Design 33*, 10, 699 – 719.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. Sketchchair: an all-in-one chair design system for end users. In *Proceedings of the fifth international conference on tangible, embedded, and embodied interaction*, TEI '11, 73–80.

SCHWARTZBURG, Y., AND PAULY, M. 2013. Fabrication-aware design with intersecting planar pieces. *Computer Graphics Forum (Proceedings of Eurographics 2013) 32*, 2.

SHEN, C.-H., FU, H., CHEN, K., AND HU, S.-M. 2012. Structure recovery by part assembly. *ACM Transactions on Graphics 31*, 6.

SHTOF, A., AGATHOS, A., GINGOLD, Y., SHAMIR, A., AND COHEN-OR, D. 2013. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum 32*, 2, 245–253. Proceedings of Eurographics 2013.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics 31*, 4.

XU, K., ZHENG, H., ZHANG, H., COHEN-OR, D., LIU, L., AND XIONG, Y. 2011. Photo-inspired model-driven 3d object modeling. *ACM Transactions on Graphics 30*, 4, 80.

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C., AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum 30*, 2, 563–572.