Computational Fabrication Guided by Function and Material Usage

Bongjin Koo

This thesis is submitted for the degree of
Doctor of Philosophy

Department of Computer Science
University College London

2016

Declaration

I, Bongjin Koo confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Bongjin Koo

To my parents.

Acknowledgements

I would like to thank my supervisor, Niloy J. Mitra, for providing great advices and ideas during my study. His approach on how to come up with new research ideas, tackle difficult problems and collaborate with other researchers is one of the most valuable things I learned from him, which will be a biggest asset for my future research (only if I can do them like him).

My gratitude to the colleagues in the Smart Geometry Processing Group at University College London: Melinos Averkiou, Aron Monszpart, Moos Hueting, James Hennessey, Tuanfeng Yang Wang, Paul Guerrero, Martin Kilian, Nicolas Mellado and Chi-Han Peng. It was a privilege to be able to work with these bright and hard-working people. Also, I owe much to the members of the VECG group for the seminars, discussions, jokes, and having coffee and lunch together.

Many thanks to my collaborators: Wilmot Li, JiaXian Yao, Maneesh Agrawala, Jean Hergel, Sylvain Lefebvre. Without their contributions and advices, this thesis would not have been completed.

I would like to thank my cousin, Jeongho, and my friends for supporting and encouraging me for the last three and a half years: Heesu, Inki, Jiho, Seung-ho, Kwang Hoon, Guillaume, Oscar, and especially, Aydan.

Lastly, I am very grateful to my parents for helping me get through the difficult times.

The work in this thesis was funded by the Marie Curie Career Integration Grant 303541, the ERC Starting Grant SmartGeometry (StG-2013-335373), and gifts from Adobe Research.

Abstract

This thesis introduces novel computational design paradigms for digital fabrication guided by *function* and *material usage*. With these approaches, the users can design prototypes of mechanical objects by specifying high-level functions of the objects, instead of manipulating low-level geometric details. These methods also provide the users with design suggestions which minimise material wastage during the design process.

The benefit of these approaches is that the users can focus on the exploration of the design space without worrying about the realisability of the design or efficient material usage. The shallow exploration of the design space due to the lack of guidance of the users in terms of function and material usage has been one of the most critical obstacles to achieving good designs using existing design tools. We verify this hypothesis by designing and fabricating a variety of objects using our computational tools.

The main contributions of the thesis are (i) clearly defined sets of constraints regarding function and material usage in the design and fabrication process, (ii) novel optimisation methods for generating designs subject to the constraints and (iii) computational tools which guide the users to design objects that satisfy the constraints.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

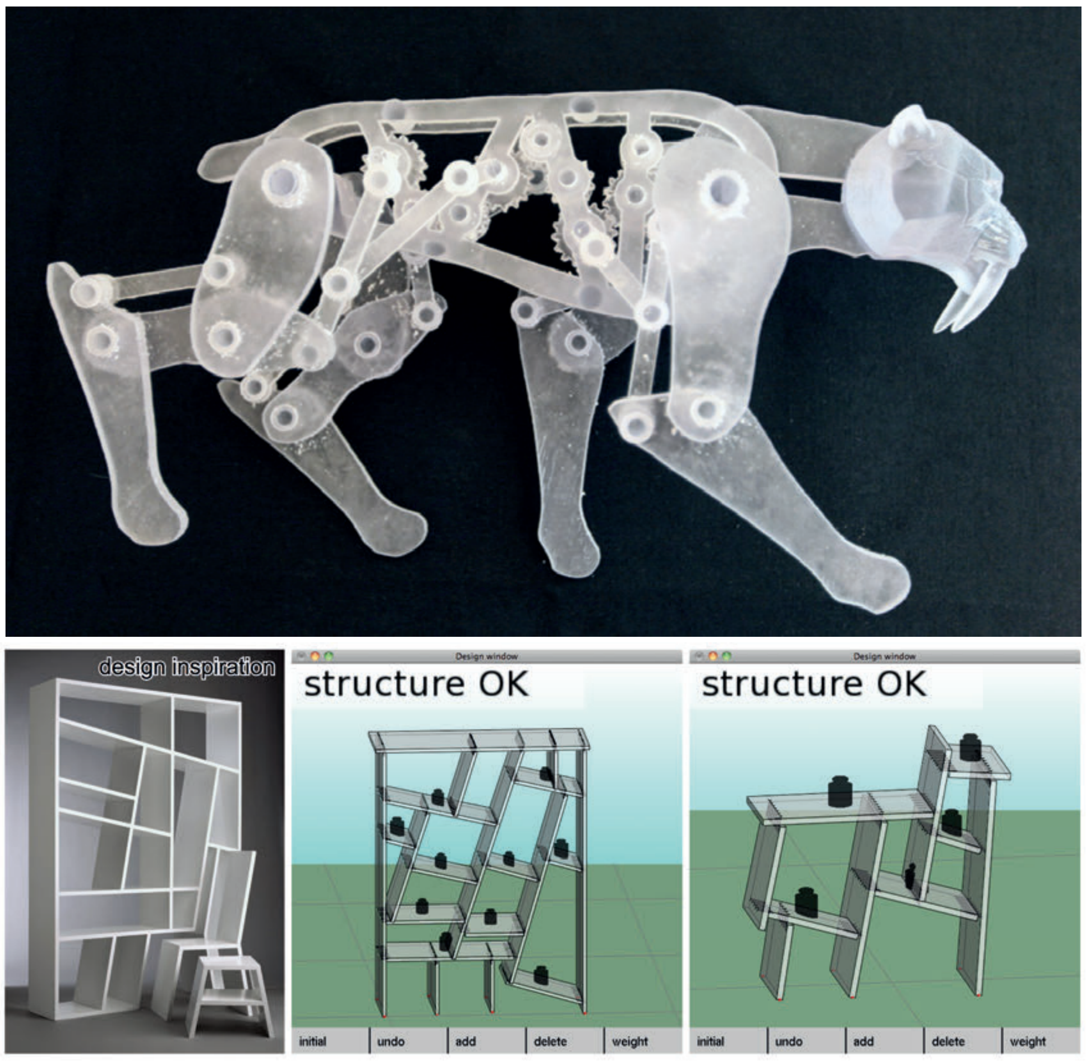


Figure 1.1: Computational fabrication. *Recently, computational fabrication methods such as 3D printing and laser cutting have been adopted widely for creating physical objects as they are becoming cheaper and easier to use. However, there are still challenges which prevent the users from fabricating objects from their own designs easily (top image by Coros et al. [CTN^{+}13] and bottom image by Umetani et al. [UIM12]).*

Mass production significantly lowered the cost of numerous products that are essential for our lives while improved their quality in general. It also helped to populate the market with many different products of various usage. There are many distinct products which we can choose in the market that one might think that we can always find the products that suit our purposes and tastes. However, this is not the case because there are large variations in our purposes and tastes to be accommodated in the standardised mass production process.

1.1 Motivation

It is very difficult, if not impossible, to make changes or customise some features of a product that is manufactured by mass production. This is due to the nature of mass production, which lowers the cost and increases the quality of products by limiting the tasks performed in one production line. As the machines in a production line are built for certain tasks, as well as the human workers are trained to do only particular jobs in the line, all products from the production line are identical or very similar in terms of appearance and function. This means that we might not be able to find products that fit our purposes and tastes though there are a vast number of products available in the market. For example, it could be impossible to find a bookcase whose dimension fits with the empty space in your living room. Or, you may be able to find a bookcase that has the correct dimensions but you do not like its style. Then, the only option is to make a bookcase from your own design which perfectly fits your purpose and taste. However, until recently, making your own physical objects was not an easy task.

You could ask a factory to manufacture objects from your designs but it would be significantly expensive because the factory should create a production line, with necessary machines and workers, just for one or a few custom objects. Also, this process will take much time to build the production line and train the workers if necessary. Therefore, this is not a viable option for a hobbyist to take. You may do woodworking to make the objects yourself but it requires significant time and effort to master the skills needed.

Computational fabrication methods such as 3D printing and laser cutting, however, have potential to address this problem. Computational fabrication refers to a set of technologies that can be used to create physical objects instructed by computer systems (see Section 2.3 for more detail). They have recently gained popularity among novices in design and fabrication, who are interested in building objects from their own designs as well as among professional designers or companies making products. This popularity stems mainly from the fact that these fabrication technologies have become more accessible thanks to the decrease in cost and to advances in design tools for fabrication, which make the technologies easier to use. Therefore, some might think that it is now easy to create physical objects from their own ideas.

However, this is not very true. To create an object using the computational fabrication methods, we need a design of the object that can be fed into the computer systems. In general, this design is created using modelling tools such as SketchUp, AutoCAD, SoildWorks, etc. These tools have become easier and more intuitive to use but still *creating a design* is very challenging. A design does not mean just the shape or geometry of an object that is pleasing to look at. A design is a solution to the problems we have or a goal that we would like to achieve with the realisation of the design (see Section 2.1). Therefore, a design should satisfy all the constraints related to the problems. The problems or goals can be creating an object that functions in a certain way, e.g., drawers or doors. Also, they can be fabricating an object while minimising the amount of wasted material during the fabrication process. Creating such a design is not trivial. Even though there are many available design tools for fabrication and they are becoming easier to use, it is not a simple task for novices to design and fabricate objects with the existing tools. This is mainly because the tools usually require prior experience and domain knowledge to create designs.

The main reasons why prior experience and domain knowledge play an important role in creating a design are:

- We need to manipulate low-level geometric features such as vertices, edges or faces in order to achieve high-level design goals related to function or material usage. And, it takes experience/knowledge to model by manipulating geometric features.

- It is not trivial to imagine the effect of the low-level changes on the high-level goals before actually fabricating and testing the design and the available design tools do not inform the user of the effect of the changes during the design process.

Even for professional designers with prior experience and domain knowledge, making a design that satisfies the constraints with respect to the problems to be solved by the design is a challenging and time-consuming task. This is because we cannot verify if all the constraints are satisfied until the actual object is fabricated and tested. This difficulty can severely slow down the product development or fabrication process because a typical process for making products or customised objects requires many iterations of *analysis*, *ideation*, *prototyping* and *evaluation* until a satisfactory design is achieved (see Section 2.2 for more detail). However, there are few tools available that directly address this problem by taking the relationship between the function and the design (or between the material usage and the design) into account so that we do not have to fabricate and test the design to verify that the constraints are satisfied.

One recent development to mitigate this difficulty in creating designs is a group

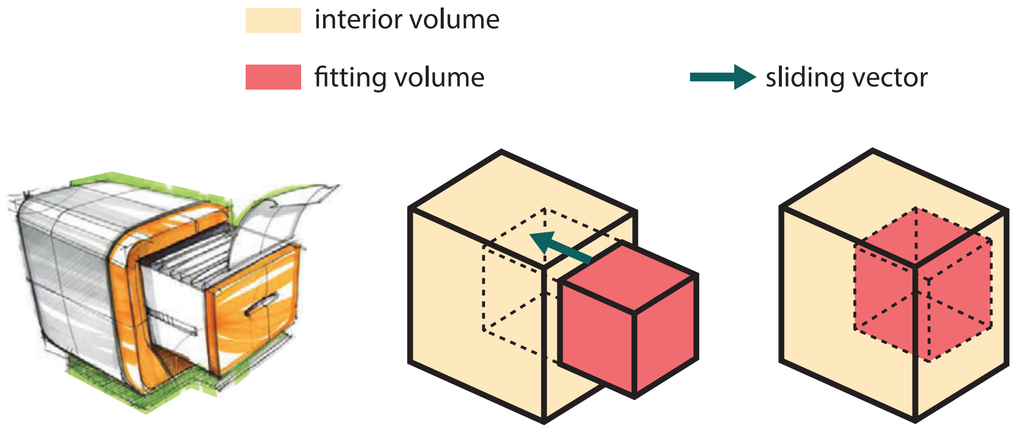


Figure 1.2: Functional relationship between parts in an object. *There are functional relationships to be satisfied for an object to function properly. For instance, the fitting volume must be smaller than the interior volume for the drawer to be closed completely as well as the cavity in the interior volume should not be too larger than the fitting volume. Low-level geometric features should be manipulated to achieve this high-level goal, i.e., a properly working drawer.*

of online design communities such as Thingiverse, where people can upload and share their designs for fabrication. Designs in these communities are fabricable and functional. As more people join the communities, the number of designs available and the quality of them are becoming higher. Therefore, we can just download a design, and 3D print or laser cut it. This is, however, only if there is a design that exactly matches our needs, i.e., with exact dimension, appearance and functions. If the design does not satisfy all of these, and usually it does not, we need to modify the design or create our own design from scratch. Thus, the difficulty still remains.

This is where our research intervenes. This thesis is an attempt to find ways for helping users, including professionals as well as novices, easily design and fabricate what they have in their minds via computational guidance with respect to *function* and *material usage*. By computational guidance, we mean the suggestions provided by computational tools which optimise and update the design and other parameters algorithmically, so that all the necessary constraints regarding function and material usage are automatically satisfied. We seek to minimise the laborious manipulation and specification of low-level geometric features by the user through allowing her to specify and achieve the high-level goals, i.e., functions and minimal material usage, directly.

1.1.1 Function

In this thesis, function refers to the functional relationships between parts in an object to achieve certain goals and to operate properly (Chapter 4). The relationships include cover, fit inside, support and flush. Joints such as hinges are usually needed between the parts that have a dynamic relationship. Depending on the function, there can be various constraints with respect to the shape of the parts and parameters of the joints. For example, the depth of a drawer should be smaller than that of the chest of drawers in which the drawer can fit as shown in Figure 1.2. Also, the cavity on the chest must be large enough for the drawer to fit in but not too big because there might be an unnecessary gap between the chest and drawer. The drawer cannot slide in, on the other hand, if the cavity is too small. This is a very simple example with a trivial mechanism but there are more complicated types of functional relationships between parts as discussed in Chapter 4. Considering these constraints on functional relationships during the design process and changing the design accordingly for the fabricated object to function properly, require time and expertise. This is because the user must manually specify all necessary low-level geometric details satisfying the high-level constraints. Moreover, the design process loop should be repeated if the fabricated object does not function as intended. Given that there are few modelling tools that address this difficulty, users are left to the trial-and-error approach, which requires much time and cost.

1.1.2 Material Usage

Material usage refers to the efficiency of the material that is used to fabricate an object. The material is *wasted* when part of the material is left after fabrication and it cannot be reused for another fabrication process. Though additive manufacturing methods such as 3D printing generally do not suffer from the material being wasted during fabrication, subtractive manufacturing methods, e.g., laser cutting and milling, can produce wasted material. This is due to the difference in their fabrication methods. Additive manufacturing methods add material only where needed and the materials used for the methods are usually in a powder, liquid or filament form, which can be easily separated or combined to obtain a desirable amount. Therefore, additive methods produce zero or minimal material waste in general if the user plans carefully. Subtractive manufacturing methods, however, remove material where necessary. The methods cut out necessary pieces from material, which leaves part of the material unused. Also, the materials used are usually in a sheet form (of wood, plastics, etc.), which is not useful when it is segmented into many pieces. For instance, we would not be able to cut a tabletop if we only had a set of small pieces of wood. Therefore, these small pieces of material should probably be thrown away.

Wasted material during the fabrication process might seem unimportant but it has a negative effect economically as well as environmentally. The leftover materials that cannot be reused for another fabrication process will be thrown away, which incurs waste and the material cost. This wasted cost can add up drastically if we consider fabricating a large amount of objects. Also, the materials that are disposed generally end up being landfilled, which can produce noxious gases and liquid [BFM03]. All or part of the remaining material after fabrication should be thrown away if the geometry

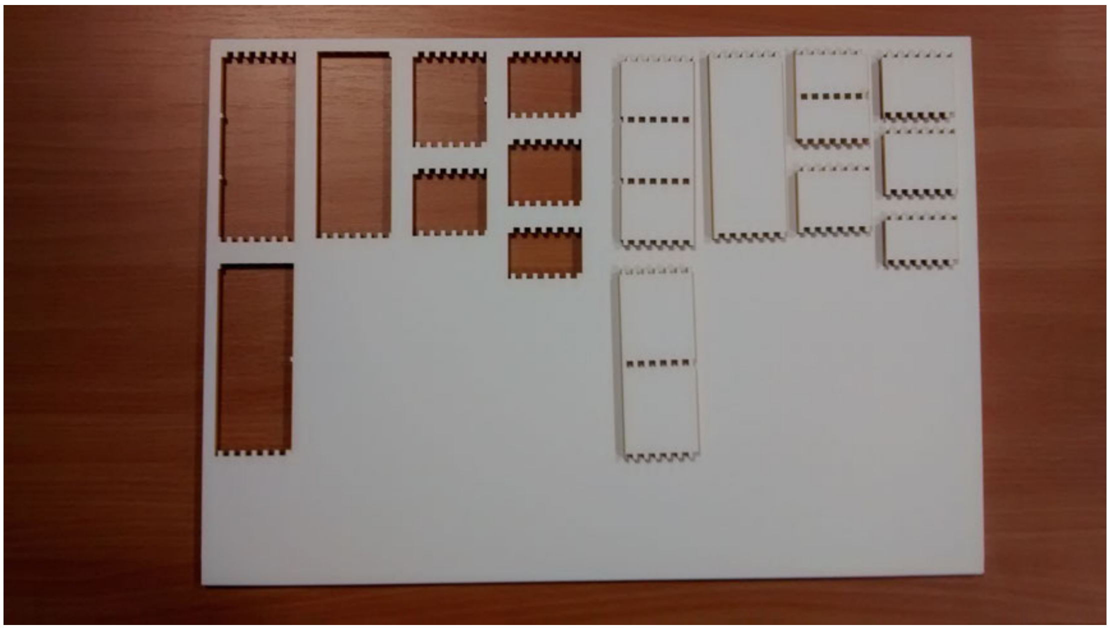


Figure 1.3: Hard-to-reuse material. *This figure shows the material after laser cutting. The space between the cut-out regions is either too small or narrow to use for another cutting. It seems easy to reduce the material wastage by changing the layout of the parts in the design in this case. However, it is not very trivial to reduce the wastage due to constraints related to the design when the shape of each part has to be changed as well as the layout (Figure 5.16). Leftover materials are disposed, which leads to economical and environmental cost. The cut-out pieces are taken out and placed on the right.*

of the remaining material makes it hard to reuse as shown in Figure 1.3. It might seem easy to minimise this wastage by changing the layout, in this case by moving the cut-out parts towards any corner of the material space (e.g., to the left-top corner of the material sheet in the figure). However, this is not trivial when there are many parts and various constraints in the design as discussed in Chapter 5. In many cases with complex designs and many constraints, the shapes of the parts should also change along with their layouts (Figure 5.16). Also, there are few computational tools available which consider material usage directly during the design process. Therefore, designers usually model with 3D modelling tools such as SketchUp or SolidWorks, and create a layout of the parts in the design using Illustrator, which means that they have to switch between different tools for 3D modelling and 2D layout. This can severely slow down the design process and result in the shallow exploration of the design space.

1.1.3 Computational Guidance

As mentioned above, the difficulty that novices, even professionals, face when they design and fabricate objects often comes from the functions or material usage not being taken into account or integrated well during the design process in currently available design tools. This leads to numerous iterations of the design process loop (i.e., refining the design, fabricating prototypes and evaluating them). This can deter novices from enjoying fabricating objects from their own designs. Also, it will prevent professionals from focusing on the aesthetics of an object itself while they are trying to optimise the design with respect to the constraints based on their experience and knowledge, which leads to the shallow exploration of the design space.

We hence hypothesise that computational guidance, i.e., informing the user of the quality of her design in terms of function and material usage, and suggesting optimal designs with respect to these constraints during the design process will help her create good designs more easily and quickly.

1.2 Contributions

Our goal, therefore, is to equip anyone who would like to design and fabricate objects from her own imagination with adequate computational tools that can guide her to achieve it. We believe that this thesis is timely and relevant as the customised fabrication is becoming more prevalent among hobbyists and professionals. To the best of our knowledge, guiding the user to have optimal design with respect to function and material usage directly during the design process has not been explored yet. We propose novel methods addressing the challenges and computational tools implementing the methods were built. Finally, various objects designed using the tools have been fabricated to verify the hypothesis.

The main outcomes of this thesis are:

- Clearly defined sets of constraints regarding function and material usage in the design and fabrication process.
- Novel algorithms for generating designs subject to the defined constraints.
- Computational tools which guide the users to design objects that satisfy the constraints.

There has been one publication [KLY$^+$14] from the project discussed in Chapter 4 and a paper from the project in Chapter 5 is under review for publication in a journal.

1.3 This Thesis

The rest of the thesis proceeds as follows:

- Chapter 2 introduces background information useful to understand the later chapters.
- Chapter 3 reviews previous research efforts related to this thesis and shows how our approaches differ from them to address the challenges better.

- Chapter 4 proposes an interactive system with which the user can design and create works-like mechanical prototypes by annotating high-level functional relationships between moving parts in the design instead of manipulating low-level geometric features laboriously.
- Chapter 5 introduces a computational design tool taking material usage into account directly in the form finding process. The tool provides users with design suggestions by analysing and optimising material space design layout while satisfying various constraints set by the users.
- Chapter 6 discusses the conclusions drawn from this research and proposes some possible future research directions based on the limitations of our methods.

Chapter 2

Background

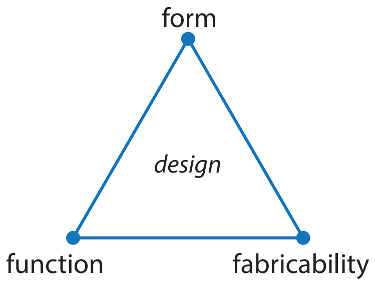


Figure 2.1: Elements of design. *A design is an interplay between form, function and fabricability. These three elements are highly correlated, thus need to be carefully adjusted together for a good design.*

This chapter is to provide prerequisite information that will be helpful to understand the later chapters in the thesis. It is important to know what a design is and which steps should be taken when we design and fabricate a physical object, in order to identify and appreciate the challenges existing in computational fabrication. Also, information on the computational fabrication technologies that are widely used recently, and also used for the fabrication of the examples appearing in the thesis, is provided. Finally, recent developments in a new research direction, fabrication-aware design, which shares the same goal with this thesis, i.e., helping the user create designs that satisfy the constraints related to fabrication, will be discussed.

2.1 Design

A design is a solution to a certain problem. The problem can be a need for a table at which five people can dine, for a chest of drawers that fits an empty space in the bedroom or for predicting tomorrow's weather. A product or object built from a design is a physical realisation of the solution to the problem (N.B. in this thesis, we only consider designs for physical objects, not abstract ones such as software). Therefore, a design must satisfy various constraints regarding *form*, *function* and *fabricability* on which the solution is based.

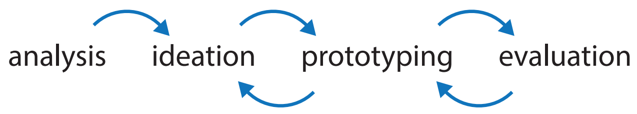


Figure 2.2: Typical design process. *In most cases, many iterations of the subset of the process may be necessary until a good solution is found. Therefore, there is a need for faster prototyping, which is a main bottleneck of the process.*

Form is the shape or geometry representing the appearance of an object. Usually, form only refers to the aesthetic aspect of an object such as styles (modern, Victorian, etc.). Function refers to the tasks that the object is supposed to perform in order to solve the problem. For example, a chair should be stable enough for a person to sit on or the lid of a clamshell mobile phone must be large enough to cover the main body of the phone. Fabricability stipulates requirements for the object to be realised in a physical form. It includes physical stability (a fabricated bookcase should stand stably), durability (the bookcase should endure the weights of the books), and gaps between moving parts.

Though these three elements seem unrelated, they are so intertwined that changes in one aspect of them lead to changes in others. For instance, functions of an object become different or undesirable if the form changes, and vice versa. Moreover, fabricability can also be affected in those cases. This dependency among form, function and fabricability makes the design process very challenging. Therefore, much practice, experiences and experiments are necessary to master the art of designing. To help with this difficult process, systematic approaches have been devised.

2.2 Design Process

As defined earlier, a design is a practice to find a fabricable form that functions in an intended way to solve a certain problem. Due to the entanglement among form, function and fabricability, it is not trivial to create a good, i.e., satisfying all relevant constraints, design. Therefore, systematic approaches are necessary to facilitate the process of design.

A design process will typically look as follows (modified from Koberg and Bagnall [KB74]):

- Analysis
- Ideation
- Prototyping
- Evaluation

Please note that this is an iterative process. It may be the case that the built prototype does not solve the problem (e.g., it does not function as intended or its shape is different from the one that was originally designated) when the prototype is tested in the evaluation stage. Then, a new shape is ideated and another prototype must be created and evaluated, which means that we have to go back to one of the previous stages and restart the process. Sometimes it is necessary to go back to the ideation stage if there are flaws in the solution itself. Or, only the prototyping can be repeated if something went wrong during the fabrication of the prototype. Therefore, these stages can, or often must, be repeated many times until an optimal solution or design to the problem is found (see Figure 2.2). Let us look at these stages in more detail.

2.2.1 Analysis

In this first phase of the design process, the problem to be solved are analysed and all necessary information helpful to solve the problem is gathered and organised. A thorough analysis is crucial to a successful design as the design, i.e., solution, is based solely on the information at hand. Imperfect or partial information on what the problem is, what constraints the problem has, etc., can lead to an imperfect or partially successful design. For example, if we do not know the bookcase to be created must hold at least 100 books, the design will be a failure when we decide to make it hold only 50 books for some reasons.

The information necessary for the design can be obtained from experiments, research or any other methods. Experiments can be used to get the physical properties of materials to be used for fabrication. Market research might be needed to understand the need for the product to be created and to predict the revenue by selling the product. Other methods such as literature review or consulting experts can also be used to gather the information on various factors of the design.

The constraints related to the problem are then extracted from the information. Constraints can be anything from colour, dimension, function of an object to price, material usage and so on (they are the "function" and "material usage" in this thesis). These constraints become the design specifications, which the design to be created in the next stage will be based on.

2.2.2 Ideation

Possible solutions, based on the design specifications, are created in this stage. Brainstorming and drawing concept sketches are usually involved.

Brainstorming [Osb79] is a process of gathering ideas from a group of people to find creative solutions to a problem. The members of the group freely suggest ideas. The ideas can be radical or difficult to realise because the power of brainstorming stems from increasing the number of ideas. Brainstorming, therefore, is premised on the

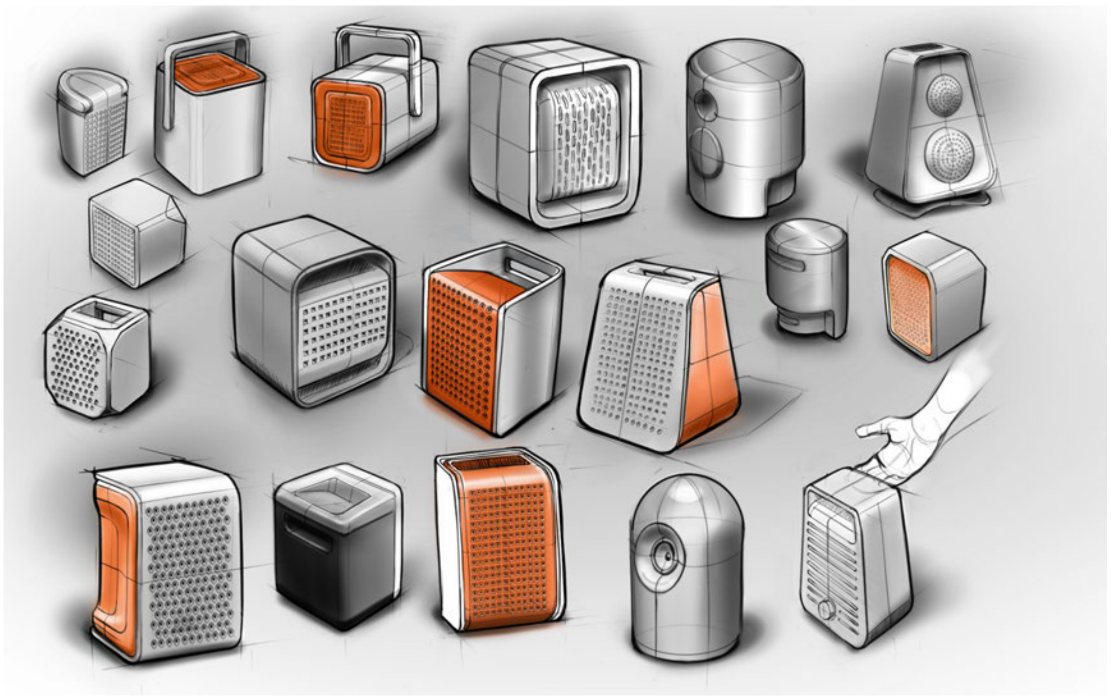


Figure 2.3: Concept sketches. *Concept sketches are a powerful tool that allows designers to explore the design space effectively. Many sketches with various specifications of the design are produced and compared for the effective exploration of the design space (image by Derick Schweppe).*

belief that the quantity of ideas increases the quality of the ideas in the group.

Concept sketches are brief illustrations of a design that focus on representing the essence of the design goals. Designers draw many different concept sketches which will potentially solve the problem, as shown in Figure 2.3, so concept sketches can help exploring various possible designs.

Using brainstorming and concept sketches, candidate designs are generated for prototyping.

2.2.3 Prototyping

In the prototyping phase, actual objects or products are built out of the designs selected among the candidates from the previous phase (see Figure 2.4). Usually, 3D models are created from the designs using modelling tools such as AutoCAD, SolidWorks, SketchUp, etc. As discussed later in more detail, this transition from 2D concept sketches to 3D models is a time-consuming process which many research efforts have been made to facilitate (see Section 3.1).

A prototype is an approximation, in terms of form and function, to the final object. The purpose of building prototypes is to accelerate the design process by simulating the final object with a low fidelity model using less expensive materials and fabrication methods. This is because it will take much time and resources if we were to build the final object with exact specifications. Hence, it will significantly slow down the design

Figure 2.4: Prototyping. *Various designs are fabricated in prototyping phase (image by Flypcap4u). As it usually takes considerable time to create prototypes, designers started to turn to 3D printing, which can accelerate the fabrication of prototypes.*

process loop given that we would need many iterations to get a satisfactory design. This bottleneck leads to a bad design due to the shallow exploration of the design space because there are always constraints on time and resources.

Still, prototyping can be slow and expensive compared to the other stages in the design process, which will again lead to the shallow exploration of the design space. However, recent advancements in computational fabrication such as 3D printing and laser cutting can help us reduce the time and resources for prototyping. These technologies that enable quick fabrication of prototypes are called *rapid prototyping*.

2.2.4 Evaluation

Finally, the fabricated objects are tested and evaluated to measure if they work properly as intended and how well they solve the problem. Tests can be just to operate the object to check if it functions, e.g., a drawer opens and closes properly. Or, they can be about physical stability such as a chair that stands stably when a person is sitting on it. If they do not solve the problem, the previous stages should be revisited and repeated, from an appropriate earlier stage to the last one, until an optimal solution is found.

2.3 Computational Fabrication

Computational fabrication refers to making physical objects using methods such as 3D printing and laser cutting, which are instructed to fabricate objects by computer systems. A file which has a design of a physical object is created generally with a design tool. Then, the file is sent to a 3D printer or laser cutter for the fabrication of the object. Due to their ease of use and low cost, these methods are used for fabricating prototypes to accelerate the product design loop. Also, hobbyists who are passionate about creating objects from their own designs have adopted these technologies because of the advantages. One thing to note is that the usage in industrial or commercial environment used to be rare though 3D printing has been widely adopted by hobbyists. This is not only because the quality of fabricated objects using 3D printing was not good enough to be used for final products but also because only a limited number of material types could be used for fabrication. However, companies are starting to use 3D printing for making final products or parts as the quality and number of usable materials increase. For example, Boeing, an aerospace company, uses 3D printed parts in their aircraft.

2.3.1 3D Printing

3D printing is a set of processes for creating physical 3-dimensional (3D) objects using materials in various forms. It adds material (which can be in a powder, filament or liquid form) layer by layer where specified by a computer programme to fabricate an object. This is the reason why it is also known as additive manufacturing (AM). 3D printing was first developed in the 1980s but it is only recent, i.e., since 2000s, that the awareness and use of 3D printing have grown significantly.

A typical process of 3D printing is as follows: 3D models to be fabricated are designed using 3D modelling tools such as AutoCAD or SolidWorks. Also, 3D scanning can be used to acquire and reconstruct 3D models from physical objects though they would need an additional step that fixes errors in the data, e.g., being non-manifold, for being able to be 3D printed. This is because the data obtained using 3D scanning tend to be noisy. Then, a 3D model designed is usually exported to STL (STereoLithography) file format, which is widely used for 3D printers. After that, the 3D model is sliced into thin layers and the information on the layers and instructions for a 3D printer are saved in a G-code file. G-code (or RS-274) is a programming language for controlling automated fabrication tools such as 3D printers or laser cutters. Finally, the G-code file is sent to the 3D printer to fabricate the object. After printing, if applicable, the support material used for holding overhanging parts is removed.

2.3.1.1 Types of 3D Printing

There are many different types of 3D printers using various printing methods. Some of widely used technologies include fused deposition modelling (FDM), stereolithography apparatus (SLA) and selective laser sintering (SLS) [PG98].

FDM printers deposit a string of molten material from a nozzle attached to the

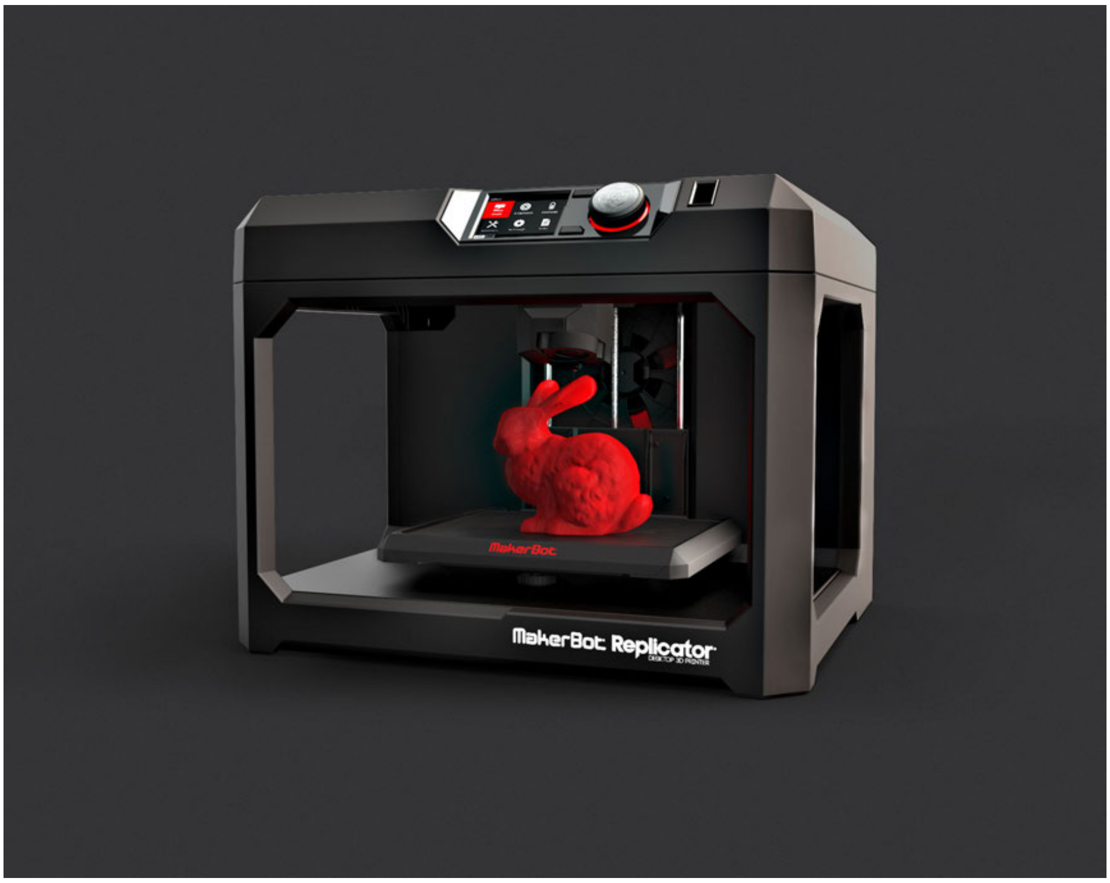


Figure 2.5: 3D printer. *The price of 3D printers has decreased significantly over the last few years, which made the technology available to wider audience. Also, the size of the printers has been reduced to enable desktop 3D printers (image by MakerBot).*

moving head, on a building platform. The material, which includes PLA (polylactic acid), ABS (acrylonitrile butadiene styrene), etc., is provided in the form of filament and melt by the heated nozzle. A separate material used for supporting overhanging parts can be used along with the material for building the object. This supporting material can be easily removed after the fabrication. FDM has the advantages that the materials used are generally cheap and a variety of materials and colours are available while it has a disadvantage that the quality of the fabricated object surface is bad compared to other 3D printing methods due to the low precision.

SLA technologies use an ultraviolet (UV) light to solidify a photopolymer resin. The UV laser draws a layer of the input 3D model on a vat of the photopolymer resin and a layer of the resin is solidified. A building platform placed in the resin vat is then lowered to the bottom of the vat to coat the solidified part completely with the resin. The platform moves upwards until the top of the part is level with the surface of the resin and a blade sweeps across the top of the part so that it leaves just one layer of the resin. Finally, the building platform is lowered again exactly by the thickness of one layer of the resin and this process is repeated until the object is built. After that, the object is taken out of the vat and drained to remove any excessive resin. Then, it is placed in a UV oven to make sure that there remains no resin that is not cured. The advantages of SLA are the quality of the fabricated object surface and fabrication speed. However, the materials used for SLA are generally expensive.

SLS methods use a carbon dioxide laser to sinter a fine powder of materials, such as metals, nylon, polycarbonates, etc. Before starting the actual printing, the whole bed containing the material powder is heated up to the point just before the powder melts. This is to minimise the distortion due to large temperature changes and to help the fusion between layers. Similar to SLA, one layer of the input object design is projected on the powder bed by the laser to sinter the material powder. Then, the power bed descends exactly by the thickness of one layer and a new layer of the powder is applied over the bed. This process is repeated until the whole object is built. One advantage of SLS over other technologies is that it does not need separate supporting structures for overhanging parts. This is thanks to the part during fabrication being surrounded by the unsintered powder. Another advantage is that it does not affect the fabrication cost much even if more parts are fabricated as long as all of the parts can be fitted in the powder bed. This is because the powder bed is filled with the powder all the time during printing even if there is only one part to be printed. However, SLS cannot fabricate objects which are hollow but watertight because unsintered powder cannot be drained in this case. Also, it can be slow for the fabricated part to be cooled before being able to be removed from the printer.

2.3.2 Laser Cutting

Laser cutting is a method to cut materials using a laser, e.g., carbon dioxide (CO_2) laser. A sheet of material is placed on the work table, and a laser beam follows the pattern given from a G-code and cuts the material as shown in Figure 2.6. The pattern represents a 2D layout of pieces of a design and the final object is built by assembling the cut pieces.

Engraving, which does not cut through the material completely, is also possible. Other types of lasers than one using CO_2 use neodymium (Nd) or neodymium yttrium-aluminium-garnet (Nd-YAG). Also, there are three common types of laser cutters based on which part is moving during fabrication, i.e., only a head that projects a laser beam moves, only the work table on which the sheet of material is placed moves, or both the head and work table move.

Laser cutters were developed in 1960s and had been used mainly in industry but the use among hobbyists has largely increased recently due to the price drop. Various types of materials, e.g., wood, plastics, glasses, metals, etc., can be used depending on the cutting method adopted by the laser cutter.

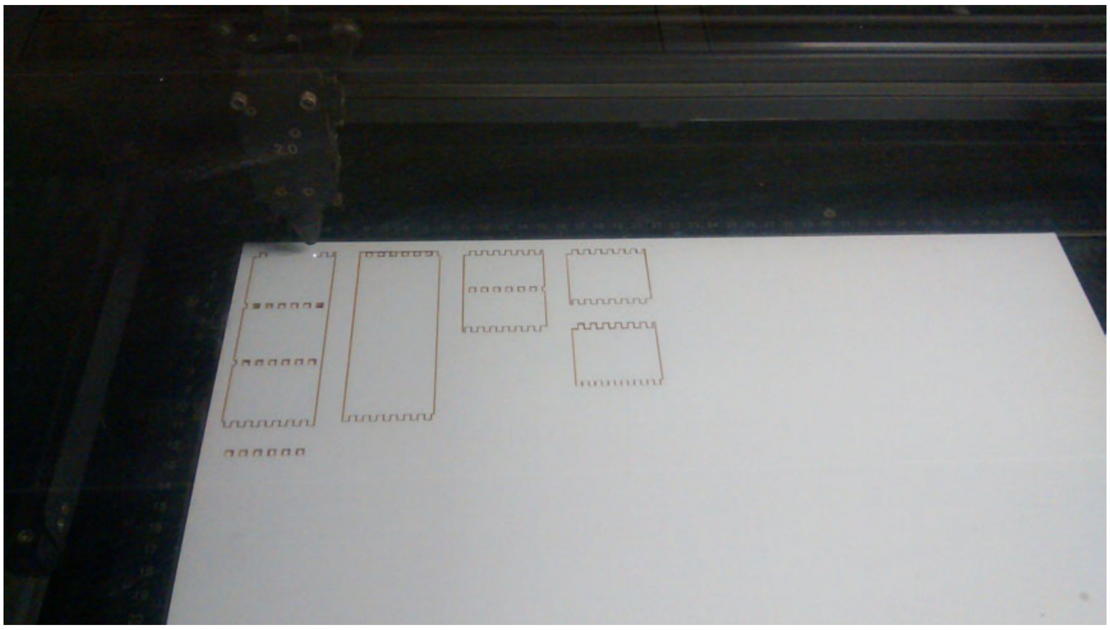


Figure 2.6: Laser cutter in action. *Laser cutters use a laser beam to cut pieces from a sheet of material placed on the work table. As with 3D printing, laser cutting is a popular option among fabrication enthusiasts.*

2.4 Fabrication-aware Design

Building a physical object from one's own design can still be very time-consuming and difficult though the computational fabrication methods became easier to use, i.e., you only need to send a design file to a 3D printer or laser cutter via computer systems. As discussed earlier, the main challenge for creating an object using computational fabrication methods lies in how to create a functioning and fabricable design. This is chiefly because there are many constraints that must be satisfied for a design to be fabricated and to function properly. Moreover, it is not easy to satisfy these constraints using most of available design tools because the tools do not consider or integrate the constraints into the design process well. Therefore, a new research branch, *fabrication-aware design*, emerged to address this problem. This discipline deals with various challenges regarding constraints imposed by the fabrication process and seeks to find ways to provide better tools for the users to design fabricable objects.

Some researches try to devise ways to automatically convert the input design, which is usually not fabricable as it is, into a fabricable one. For example, an input 3D model is approximated with a set of planes so that it can be fabricated [MSM11, HBA12, SP13, CPMS14]. Furthermore, some methods aim at fabricating objects with mechanisms such as hinges [BBJP12, CCA$^+$12, ZXS$^+$12, CLM$^+$13, CTN$^+$13].

Others concern how to reduce the material wastage and/or building time when fabricating with 3D printers. Some achieves them by hollowing the interior of an object [SVB$^+$12, PWLSH13, LSZ$^+$14] or using frames or scaffolds [WWY$^+$13, DHL14].

Regarding the efficient material usage when fabricating with laser cutters, an interactive system that allows the user to move the cut-out parts for compact 2D layouts was proposed [SCMI13]. More generally, achieving efficient material use for laser cutting is a 2D bin packing problem, which has been widely investigated [Jyl10].

These related researches to the thesis will be reviewed in the Chapter 3 in more detail. As we will see, however, creating a design and changing the design for satisfying the constraints are separate in most of the existing methods, which often makes the user switch between different tools several times and slows down the whole fabrication process. This problem is what this thesis tries to address.

Chapter 3

Literature Review

There has been an increasing amount of work on computational fabrication recently as the popularity of the fabrication methods grows. However, design and fabrication are considered separate and sequential steps in most of the previous work, as opposed to our approach where constraints related to fabrication are automatically satisfied when the design changes. In particular, considering functions that the design can exhibit and material usage for fabrication directly during the design process, to the best of our knowledge, has not been explored.

3.1 Analysing Designs

In the design process, a 3D model is created usually based on concept sketches. This means that we need to construct a 3D representation from 2D representations of the design. This is often a time-consuming and difficult task, which can slow down the design process. Much research effort, therefore, has been devoted for generating 3D representations automatically from 2D concept sketches. We employ one of the methods [SLZ$^+$13] to generate input 3D models for the system introduced in Chapter 4.

Bae et al. [BBS08] introduce a system for creating 3D curves with sketch-like workflow with better user experience than similar previous approaches. Schmidt et al. [SKSK09] propose a method for inferring 3D curves from single-view drawings. They utilise analytic drawing techniques, which draw 2D lines or scaffolds to constraint a 3D shape. Then, 3D lines and curves satisfying the constraints are inferred from a sketch drawn upon the scaffolds. This method is very effective because one can create 3D drawings by doing the operations that would be needed to create the drawings on a paper. Shao et al. [SBSS12] create 3D-like shadings from 2D concept sketches by leveraging a convention used by designers in the concept sketches, i.e., cross-sections. Similar to the scaffolds explained above, cross-sections provide information about 3D shapes. They formulate the relationships between the cross-sections and 3D geometries mathematically, and infer normal fields using the formulation. Xu et al. [XCS$^+$14] also introduce a framework to generate 3D curve networks from 2D drawings by extracting information about 3D shapes. In this case, the authors propose a selective regularisation method based on insights that designers often use viewpoints that are most representative of the 3D shapes and curves which convey intrinsic information on the shapes, e.g., symmetry, curvature, etc.

However, these techniques focus on creating static 3D geometry, whereas we aim to produce models with moving parts. In this respect, the most relevant previous work is by Shao et al. [SLZ$^+$13], who develop a system for creating interactive 3D models from a set of concept sketches. These sketches depict an object from different viewpoints and with different configurations of moving parts. Each part in the sketch is represented by a proxy and the system automatically infers the correspondences between the proxies across the different viewpoints. However, their approach does not produce a single consistent 3D model, which is necessary for creating a physical realisation of the design.

3.2 Guided Design

The hypothesis of this thesis is that design tools can be more helpful if they guide the user to achieve optimal designs with respect to constraints regarding high-level goals. Several methods have been proposed with a similar motivation.

Talton et al. [TGY$^+$09] developed a system that the user can explore a design space to choose a 3D model from and customise the selected model. The system uses a parameterised design space for 3D models and approximates the distribution of models in a design space to suggest desirable model candidates. This distribution is based on the models created by the users of the system. This system is helpful for novices, as well as professional designers, because it suggests models that satisfy the parameters and constraints set by the user. Xu et al. [XZCOC12] propose a fit-and-diverse framework which generates novel 3D models from an initial set of models using set evolution. Set evolution evolves all 3D models in an object class, e.g., tables or bookcases, not just individual models. This framework allows the user to interactively guide model suggestions to fit their preferences but to be diverse. These efforts, however, focus on aspects of digital content creation without fabrication and material considerations.

Among the methods which are more closely related to our motivation, Umetani et al. [UIM12] introduce an interactive tool that guides the user in terms of physical validity (e.g., stability or joint bending forces) throughout the modelling process by suggesting valid designs when the design is physically invalid. More recently, Shugrina et al. [SSM15] developed a system that allows novices to easily customise parametric models while maintaining 3D printability of the models. The system precomputes the design validity and 3D printability by sampling the design space and creates only valid designs via the parameters set by the user at runtime. We share a similar goal by taking into account the functional relationships of parts in objects or material usage in order to guide the user to design fabricable objects during the design process. However, we are unaware of any previous methods that have investigated how material usage can be analysed to guide the design of shapes.

3.3 Constraint-based Modelling

In the computer graphics and CAD community, constrained-based modelling has long been demonstrated as a powerful parametric way to design shapes and interact with them (see [BR98]). For the work in Chapter 4, we adopt the high-level approach as previous graphics research that tries to automatically determine the relevant geometric relationships between parts to enable editing and synthesis of 3D models. iWIRES [GSMCO09] analyses 3D models to extract 1D wires which have information on the characteristics and global structure of the model shapes. Then, the wires are used for editing the models while maintaining the characteristic properties and global structure. Xu et al. [XWY+09] perform slippable motion analysis to detect joint constraints in models. The components connected by joints in the models can be rigid or deformable. The joint constraints, which represent high-level semantics between components, are then used to deform the models while maintaining the user's design intent and the natural behaviour of the joints.

Zheng et al. [ZFCO+11] propose component-wise controllers which capture the degrees of freedom of components in 3D shapes so that the user can edit the shapes intuitively. These controllers are inferred by a hierarchical analysis on the shapes and are therefore adapted to the characteristics of the components. Thus, high-level characteristics of the shapes, e.g., symmetry and parallelism are maintained during the editing of the shapes using the controllers. Also, a new algebraic model of shape structure for high-level shape editing which respects global characteristics was introduced [BWSK12]. In this model, regular translational patterns existing in a 3D shape are extracted and are used to explain the shape by identifying useful degrees of freedom in the shape. Then, these degrees of freedom are presented to the user for intuitive shape editing. Zheng et al. [ZCC+12] partially reconstruct the scene in images using cuboid-based proxies by the help of the user and extract non-local relations between the proxies such as coplanarity and parallelism. The user can then edit the scene in a plausible way, i.e., as if the objects in the scene are manipulated in real world, by manipulating the proxies.

Our approach is also similar in spirit to previous mechanical engineering research that proposes a declarative modelling scheme which enables the user to create an object by providing a set of abstract specifications of the object regarding geometric or topological properties [DL97]. Yvars [Yva08] uses a state graph to model the design space and solves the model as a constraint satisfaction problem (CSP) for the exploration of the design space.

However, to our knowledge, we are the first to focus specifically on the design of articulated, works-like prototypes, and a key part of our contribution is defining a set of functional relationships that are useful for this design task. We also note that some professional CAD tools include constraint-based modelling features, but they require users to manually specify low-level geometric relationships between part and joint parameters. In contrast, our approach automatically converts high-level user-specified functional relationships into the relevant low-level constraints (see Chapter 4).

3.4 Fabrication-aware Design

The recent advances in computational fabrication made the fabrication technologies cheaper and easier to use. However, for novices, or even for professional designers, it takes time and effort to create a design that can function as intended when fabricated. Therefore, researchers have started to explore methods for helping the user create designs that satisfy the constraints related to fabrication.

3.4.1 Fabricability

McCrae et al. [MSM11] use planar slices to generate shape abstractions of 3D models, which can later be fabricated. From a user study, they observe that humans have a consistent and similar notion for abstracting 3D objects using planar sections and the planar sections are correlated to the geometric features of the objects. Based on this, they develop a method that selects planes to capture the geometric features maximally. Hildebrand et al. [HBA12] also introduce an algorithm for abstracting 3D shapes with intersecting planes and generating cardboard sculptures by sliding the planes that are cut with a laser cutter. As the complexity of the slice insertion ordering and direction increases exponentially with respect to the number of the planes, it is infeasible to figure out the order manually. Therefore, they extend a binary space partitioning (BSP) tree to represent 3D shapes with added information on the plane insertion directions and efficiently find the ordering of plane insertion using the tree. Schwartzburg and Pauly [SP13] take a similar approach that abstracts 3D shapes and produces interlocking planar pieces representing the shapes. However, they further consider the physical stability of the fabricated objects by solving for it with respect to geometric constraints related to the stability. Mesh joinery method [CPMS14] extends these techniques to enable fabricating very complex objects using laser cutters. They align the planar elements with a cross-field, which is defined over a 3D model surface, to represent the global features of the surface well.

There is another line of research focusing on fabricating pop-up structures. A method for generating 3D paper buildings that are popped up from planar layouts is proposed [LSH$^+$10]. The method is based on the constraints that regions in a planar layout, comprising cut and fold lines, should remain rigid and non-intersecting when popped-up as well as the 3D structure should stand stably. Li et al. [LJGH11] investigate the mechanisms of a more general class of pop-up structures. They provide sufficient conditions to make pop-uppable paper structure, i.e., it should be flat and within the page border when closed and no extra forces are needed except for holding two patches of the paper when opening and closing. Using these conditions, they develop a system that allows the user to design pop-ups interactively.

However, these methods focus on the design of static objects. In the domain of moving objects, Bächer et al. [BBJP12] propose a method for generating 3D printable single-material articulated characters with joints from skinned meshes. The characters generated from the method consist of piecewise rigid parts and approximate the kinematics of the input skinned meshes. They use ball-and-socket or hinge joints with friction for the fabricated models to be posable. Calì et al. [CCA$^+$12] achieve a similar goal with a different approach, and they particularly experiment heavily on finding a single versatile joint design which can be adapted depending on the limitations or specifications regarding 3D printers, e.g., precision, use of support material, etc. Constructable [MLB12] is an interactive fabrication tool which enables the user to create functional mechanical devices using a laser cutter. Especially, the tool allows the user to work directly on the pieces that are fabricated, instead of working via 3D modelling tools and then, separately, fabricating after designing. Zhu et al. [ZXS$^+$12] introduce a technique to generate mechanical toys operated by gears and cranks from the motion of their components specified by the user. They use simulated annealing to optimise parameters regarding the shapes of parts in a toy for generating the desired motions. Ceylan et al. [CLM$^+$13] use motion capture sequences to automatically generate mechanical automata which approximate the input motions. They focus on generating humanoid automata so assume that the motion of the limbs are close to being planar. Then, the motions are realised using a set of links, revolute joints, gears, etc. in automata. There is another method that can also create mechanical characters but the motions are specified by the curves drawn by the user instead of using motion capture data [CTN$^+$13]. As the solution space is vast, they exploit a database storing a sampling of parameter spaces for mechanical assembly types such as hinge joints or gears. This database is then used to find the best types of mechanical assembly and initial parameters given the motion curves specified by the user. Schulz et al. [SSL$^+$14] develop a data-driven interactive system that helps the user design fabricable 3D models. They create a database of design templates which are parameterised from the designs created by mechanical engineers. These templates have information on geosemantic relationships between parts of the models as well as on the connection between parts for fabrication. Then, the user can design new models by selecting parts from the design template database and further adjusting relevant parameters. To simplify fabrication, Fu et al. [FSY$^+$15] suggest a method to generate a globally-interlocking furniture assembly, where parts in furniture interlock each other. This global interlocking furniture enables easy disassembly/reassembly of furniture without using glue, screws, etc.

Although all of these methods produce working and physically realisable models, they mainly focus on the problem of adding joints, gears or linkages *after* the models have been designed. In contrast, our work is to guide the user to refine the designs based on constraints such as functional relationships between parts or material usage.

3.4.2 Material Considerations

As discussed in Section 1.1.2, wasted materials during fabrication process can have an economically, either in terms of cost or time, and environmentally negative implication. Especially, materials used for 3D printing are still not very cheap so various approaches have been developed to economically and efficiently 3D print designed objects. For example, a method that adaptively hollows out interiors and adds struts to create durable yet cost-effective 3D printouts [SVB$^+$12] was proposed. This method performs a structural analysis with 3D medial axis approximations to detect problems, i.e., areas having high stress, in the structure of a model. Then, the model is updated until all stress areas are corrected to have a sound structure by hollowing, thickening and strut insertion. Prévost et al. [PWLSH13] hollow the shape interiors in conjunction with shape deformation to ensure stability of a model while maintaining the original surface shape of the model. The aim of this method is not about reducing the material usage but making the model stably stand when fabricated. However, it can help using less material because the method hollows the interior of objects instead of adding a large base to the objects for them to be able to stand stably. Lu et al. [LSZ$^+$14] perform FEM analysis and Voronoi tessellation to carve out the interior of 3D models into honeycomb structures. Honeycomb structures can exhibit endurance for stress with minimal material usage. Therefore, the method proposed finds the maximal hollowing of the interior given stress on objects, to achieve efficient material usage and durability.

Wang et al. [WWY$^+$13] convert input 3D models into skin-frame structures, which use less material but are still physically stable and fabricable as well as approximate the input shapes closely. They also address a problem during fabrication using 3D printers without support materials, which is the inability to fabricate overhanging parts correctly, by adding extra struts where necessary. Dumas et al. [DHL14] aim at reducing material usage and building time needed particularly for generating support structures for overhanging parts. They build bridges between vertical pillars to create scaffoldings for stable fabrication of overhanging parts. Similarly, Hu et al. [HJW15] propose to optimise and deform the shape of a 3D model so that the amount of support structures used during 3D printing can be reduced.

Alternatively, methods have been developed to decompose and pack 3D models for reducing assembly cost, support material, printing time or making big objects printable on small 3D printers. Luo et al. [LBRM12] break one large 3D model into smaller pieces so that they can fit and be printed in small 3D printers. The smaller pieces that are fabricated separately can be assembled to yield the original object. When breaking the object into pieces, constraints such as the assemblability (the pieces being able to be assembled into the original object), structural stability (seams being away from high stress regions), aesthetics (seams being not visually obtrusive), etc. are taken into account. PackMerger [VGB$^+$14] converts an input 3D model into a set of small shells that can be tightly packed so that printing time and supporting material usage can be reduced. The printed shells are glued together to form the original object. Yao et al. [YCL$^+$15] also seek to find an optimal partitioning of an input 3D model to save time or enable a large model to be printed in a small 3D printer. They analyse the partitioning quality using metrics such as stress load, packed size, assembling, etc. and then the quality is optimised using level set methods. Dapper [CZL$^+$15] also employs a decompose-and-pack approach for minimum assembly cost, support material and build time when using 3D printers. It breaks 3D objects into pyramidal primitives, then finds good packing configurations to achieve the goal.

Saving material usage in 2D material space, e.g., when we use laser cutting, is a classical 2D bin packing problem which has been vastly investigated. Though the bin packing is an NP-hard problem, there are numerous heuristics that can generate optimal solutions (see [Jyl10] for more detail). Recently, Saakes et al. [SCMI13] proposed an interactive system to allow the user to interactively layout parts using 2D rigid body simulation.

Methods mentioned above, however, do not explicitly modify the original designs in order to improve material usage in contrast to our approach presented in Chapter 5.

Chapter 4

Guidance by Function

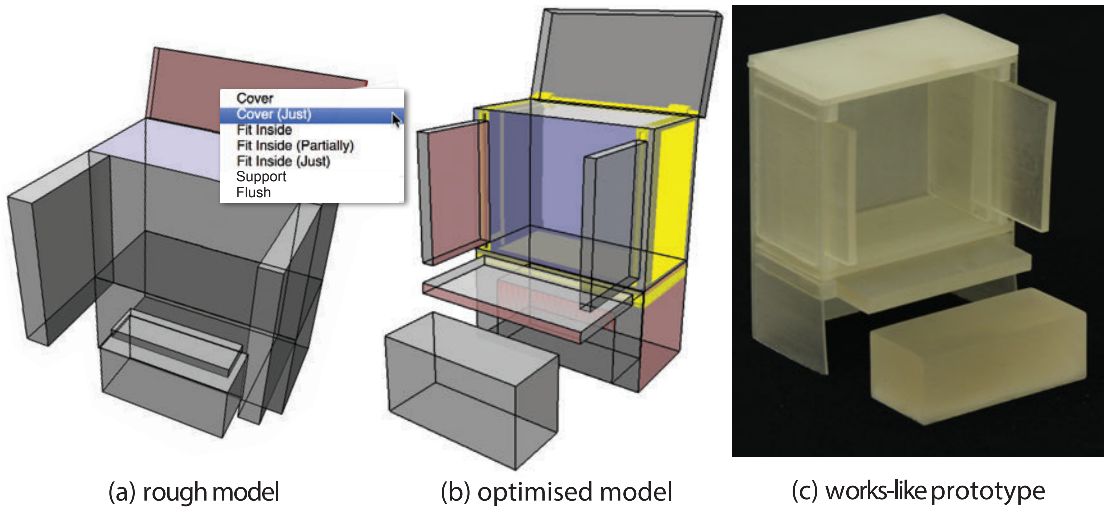


Figure 4.1: Creating works-like prototypes. *Users start by creating a rough 3D model of a design and then specify the desired functional relationships between parts (a). Our system optimises part and joint parameters to generate a working model (b) that can be fabricated as a physical prototype (c).*

Creating physical prototypes is an integral part of the product development process (see Section 2.2). Prototypes help designers evaluate and refine potential designs, explore multiple approaches in parallel, and communicate designs to others [KLP01, ES09, Hal12]. Designers create different types of prototypes for different purposes. Early in the design cycle, they often create *works-like* prototypes like the one shown in Figure 4.2 that embody the functional aspects of a design. Such prototypes typically contain working mechanical joints, e.g., hinges, but simplified part geometry so that designers can focus on the mechanical "architecture" (i.e., how parts move and fit together) of the product. Also, this is because it will take much time and resources if they fabricate the prototypes with exact geometric fidelity and materials, which hinders speedy iterations of the design process. Later in the cycle, however, designers may create *looks-like* prototypes that convey the detailed shape and/or colour-material-finish (CMF) of a design. Such prototypes help designers (and clients) understand the

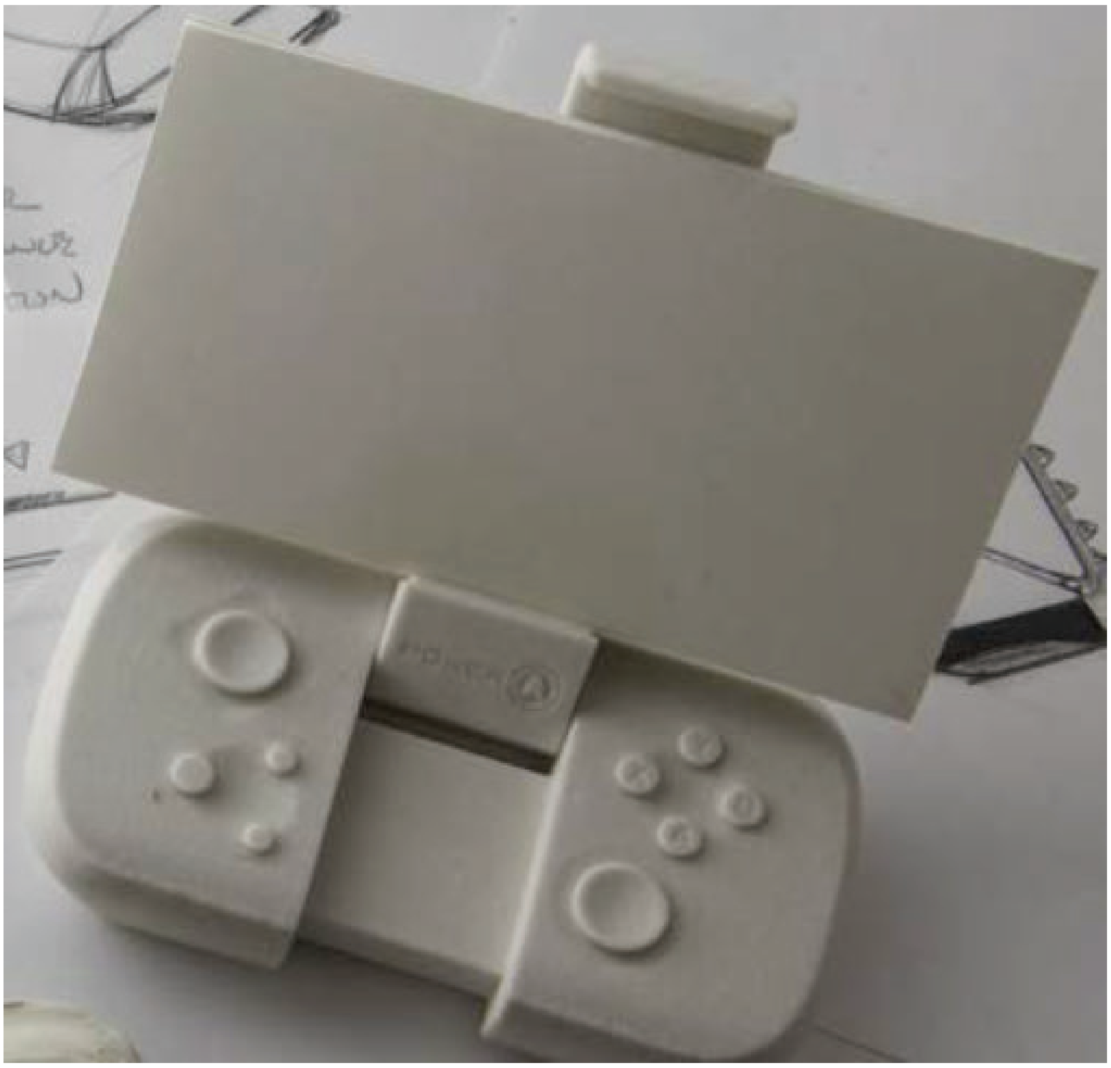


Figure 4.2: Works-like prototype. *Prototypes are used to check if the design functions correctly when fabricated.*

intended appearance of the product.

In this chapter, we focus on the task of creating works-like prototypes and aim at introducing a novel method to help the user design and fabricate those prototypes. Designers are increasingly turning to 3D printing as a tool for fabricating physical prototypes. Moreover, as the precision of the shapes that 3D printers can achieve increases, designers started to explore fabricating objects with mechanical joints that approximate the intended functionality of a design, not just static objects. Previously, the quality of fabricated mechanisms was bad, e.g., hinges can wobble because of the excessive gap between the moving parts or can be stuck during operation because of the lack of room between the moving parts, due to the low precision of 3D printers. Now, 3D printers do not generally suffer from this problem, yet creating working mechanisms requires expertise and time [UE07, Hal12]. Designers must carefully specify part proportions and joint parameters to ensure that all moving parts fit together in the intended manner without interference. This task often involves non-trivial geometric calculations, even when the parts are composed of simple primitives (e.g., cuboids). This difficulty mainly comes from the fact that we need to manipulate low-level geometric details such as vertices, edges and faces of 3D models in order to achieve high-level goals like functional relationships between parts in the models. Moreover, designers almost always iterate and refine prototypes by modifying certain parameters (e.g., size of a part, joint types), which requires updating the part and joint parameters to ensure a working prototype. Due to these challenges, designers often have to work with skilled CAD engineers to help them create physical works-like prototypes. This added friction in the ideation-prototyping-evaluation cycle significantly limits the number of iterations designers can make and often leads to shallow exploration of the design space.

To address this problem, we present an interactive system that helps designers create functioning works-like prototypes. The user starts by creating a rough 3D model of all the parts in the design and specifying the types of joints that connect the parts. Then, instead of tweaking part and joint parameters to produce a working model, our system allows users to directly specify high-level *functional relationships* between parts (e.g., part A fits inside part B, parts C and D support part E, etc.). Based on these relationships, our system automatically adjusts part proportions and joint parameters to produce a physically realisable working model (Figure 4.1c). To aid in design exploration, the system propagates edits (e.g., users may add/remove parts or modify part proportions) throughout the design to ensure that all the specified functional relationships are preserved. By allowing users to work primarily at the level of functional relationships (high-level) rather than at that of joint and part geometry (low-level), our system helps designers create new prototypes more quickly and experiment with variations of existing designs.

We used our system to fabricate works-like prototypes for a variety of objects, including articulated devices (folding tablet), appliances (printer), and furniture (kiosk cabinet, sofa bunk). Figure 4.1 shows the 3D printed prototype of a cabinet design generated by our system, and Figure 4.16 shows all of our examples with the moving parts in different configurations. It took less than 10 minutes of interaction time in our system to create working 3D models for these results.

Our work makes two main contributions:

- Identifying common functional relationships among many product designs and defining these relationships as a set of low-level geometric constraints that can be satisfied by standard constraint solvers.

- Semi-automatic tools for realising the functional relationships with a small amount of the user annotation.

4.1 Overview

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

4.1.1 Parts

In our system, each part consists of one or more axis-aligned connected cuboids as shown in Figure 4.1a. Since works-like prototypes focus on mechanical functionality rather than detailed appearance, representing parts with sets of cuboids is often sufficient. We provide two part modelling interfaces: (1) users draw 2D boxes from one of three orthogonal viewpoints (top, front, side) and then extrude them into cuboids; (2) users annotate an input concept sketch by clicking on feature points (e.g., corners) to create cuboid parts (we used the method described in Shao et al. [SLZ$^+$13]). In addition to modelling solid parts, users can also create cuboidal or cylindrical cavities within a part by placing cuboids or cylinders where the cavities should be created. We refer to the initial positions and orientations of all the extracted proxies as the *base configuration* of the object.

4.1.2 Joints

Our system supports four types of mechanical joints (Figure 4.3). *Hinges* are attached between the two edges of the two faces, which are from the two different moving parts, and enable rotation around an axis. *Sliding joints* allow two parts to translate linearly along a sliding vector with respect to one another. *Sliding hinges* enable both rotation and translation. *Double pivot joints* allow rotations around two axes separated by a rigid link. To add joints, the user first selects two adjacent parts and then does one of the fol-

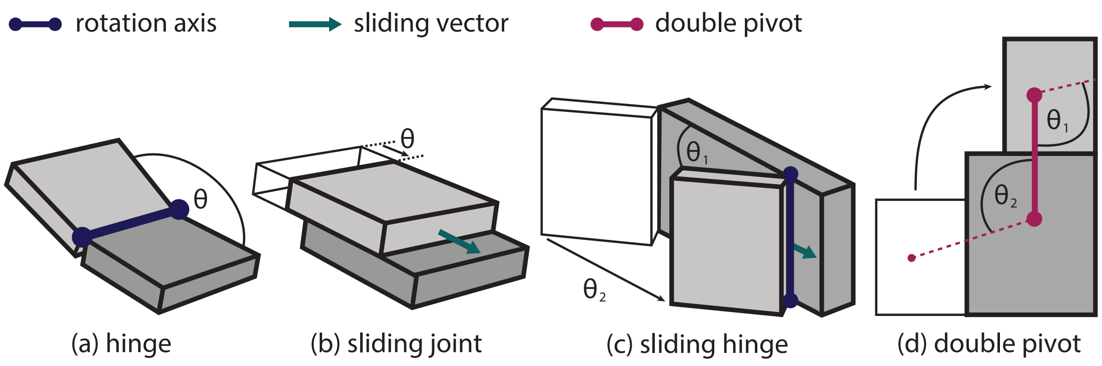


Figure 4.3: Different types of joints and their parameters. *We identified common joint types existing in product designs (e.g., a cabinet, sofa bunk and printer) and incorporated them into our system.*

lowing: add a hinge by selecting any cuboid edge that touches both parts; add a sliding joint by selecting a cuboid face whose normal defines a sliding vector; add a sliding hinge by selecting both an edge and a face; add a double pivot by clicking points on two coplanar faces (one on each part) that define the pivot positions. We represent the pose of a joint j as $j(\theta)$. For hinges, θ represents the angle between the pair of attached faces (Figure 4.3a). For sliding joints, θ is the signed offset along the sliding vector between the two connected parts with respect to the base configuration (Figure 4.3b). For sliding hinges, θ is a tuple that includes both the rotational and translational parameters of the joint (Figure 4.3c). For double pivots, θ is a tuple that includes the rotations around both pivots (Figure 4.3d). We write the combined pose of a set of joints $J = (j_1, j_2, \ldots, j_n)$ with corresponding joint parameters $\Theta = (\theta_1, \theta_2, \ldots, \theta_n)$ as $J(\Theta)$.

4.2 Defining Functional Relationships

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

4.2.1 Cover

In many products, certain parts are designed to cover either other parts or cavities. For example, the top half of a clamshell phone must cover the bottom half, and the lid of a container covers its opening. The relationship stipulates that specific faces of the covering part must be the same size or larger than specific faces of the covered part or cavity (Figure 4.4). While the simplest examples involve a single face covering another single face, in general, cover relationships can involve a set of faces covering another set of faces, based on a *corresponding faces graph* that indicates which subsets of faces correspond. We say that the faces are in their *covered configuration* when all corresponding faces lie in the same plane, the covered faces lie within the covering faces, and there are no gaps or overlaps between the covering or covered faces.

We define a cover relationship as $Cover(F^A, F^B, M, J, \Theta)$, where the set of faces F^A cover the set of faces F^B, M is a corresponding faces graph linking each face in F^A to the corresponding faces in F^B that it must (fully or partially) cover, and $J(\Theta)$ are the set of joints and parameters that put the faces into their covered configuration. Under

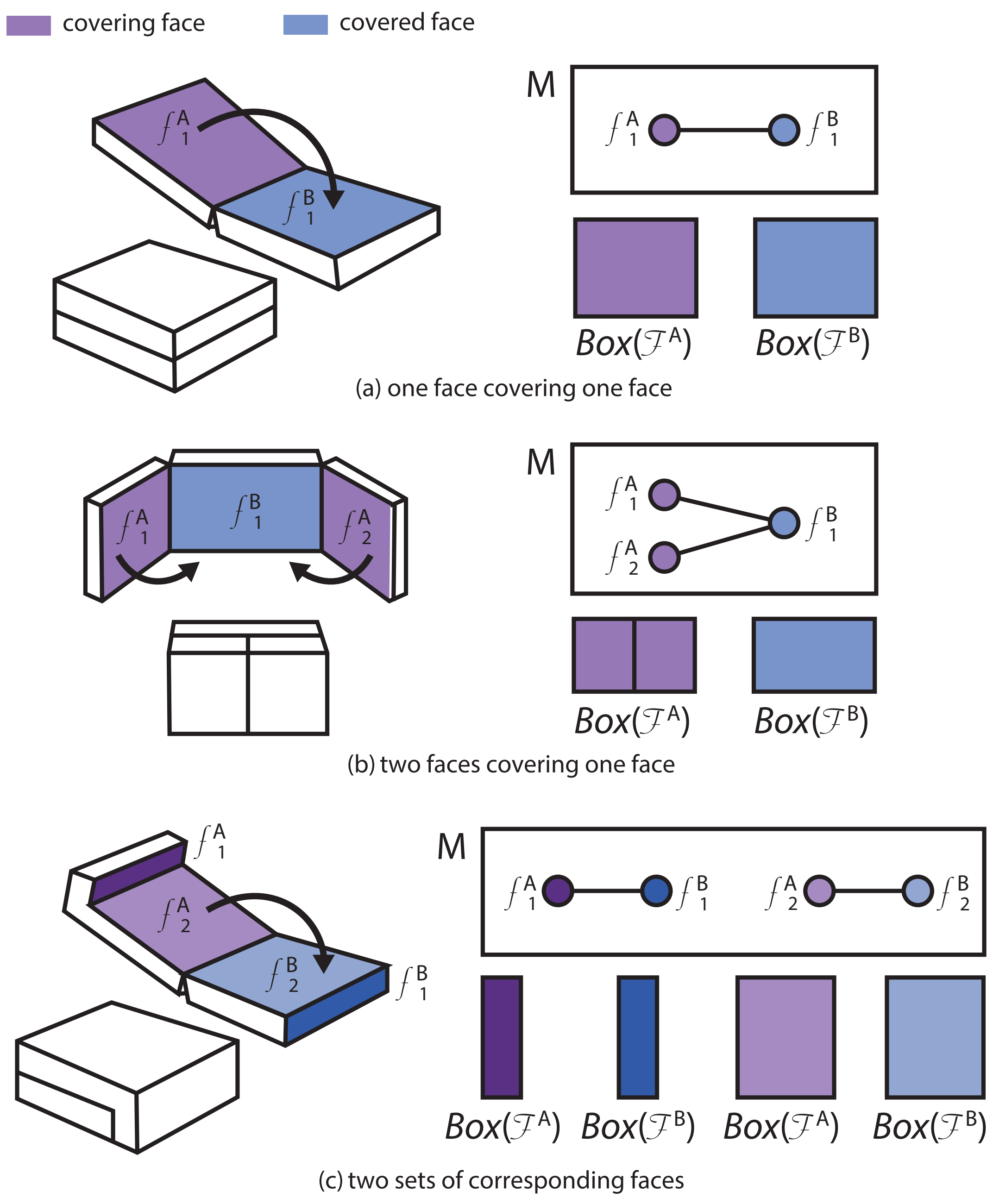


Figure 4.4: Example cover relationships. *Each example shows the covering faces $F^A(\ni f_n^A)$ and covered faces $F^B(\ni f_n^B)$ in their base and covered configurations, corresponding faces graph M, and the bounding boxes of the corresponding faces.*

this definition, each connected component of M contains two sets of faces $\mathcal{F}^A \subseteq F^A$ and $\mathcal{F}^B \subseteq F^B$, and in the covered configuration, these corresponding faces lie in a plane where $\mathcal{F}^A$ covers $\mathcal{F}^B$. Thus, for the cover relationship to hold, these faces must meet the following geometric constraints:

$$\begin{aligned} Box(\mathcal{F}^A)_l &\leq Box(\mathcal{F}^B)_l \qquad & Box(\mathcal{F}^A)_r &\geq Box(\mathcal{F}^B)_r \\ Box(\mathcal{F}^A)_b &\leq Box(\mathcal{F}^B)_b \qquad & Box(\mathcal{F}^A)_t &\geq Box(\mathcal{F}^B)_t \end{aligned} \tag{4.1}$$

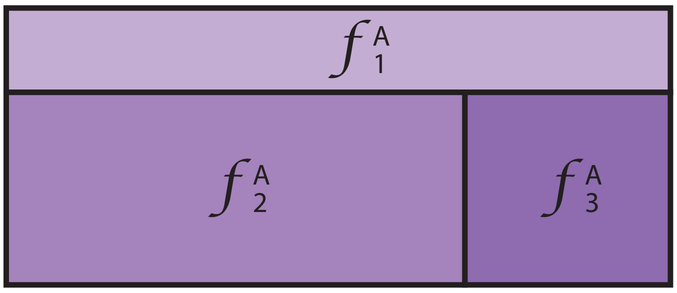


Figure 4.5: Example packing constraints. *Packing constraints stipulate a tight packing between neighbouring faces.*

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

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

$$\begin{aligned} Box(f_1^A)_b &= Box(f_2^A)_t \\ Box(f_1^A)_b &= Box(f_3^A)_t \\ Box(f_2^A)_r &= Box(f_3^A)_l \end{aligned} \tag{4.2}$$

as in the example in Figure 4.5.

4.2.2 Fit Inside

Another common functional relationship involves one or more parts fitting inside another (Figure 4.6). For example, drawers must fit inside the body of a dresser, and a pocket door must fit inside its housing. Some designs include parts that fit partially inside other parts without being completely contained. In some cases, the inside part is designed to be just small enough to fit inside the container part in certain dimensions. We say that the parts are in their *fitting configuration* when the appropriate part fits inside the other.

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

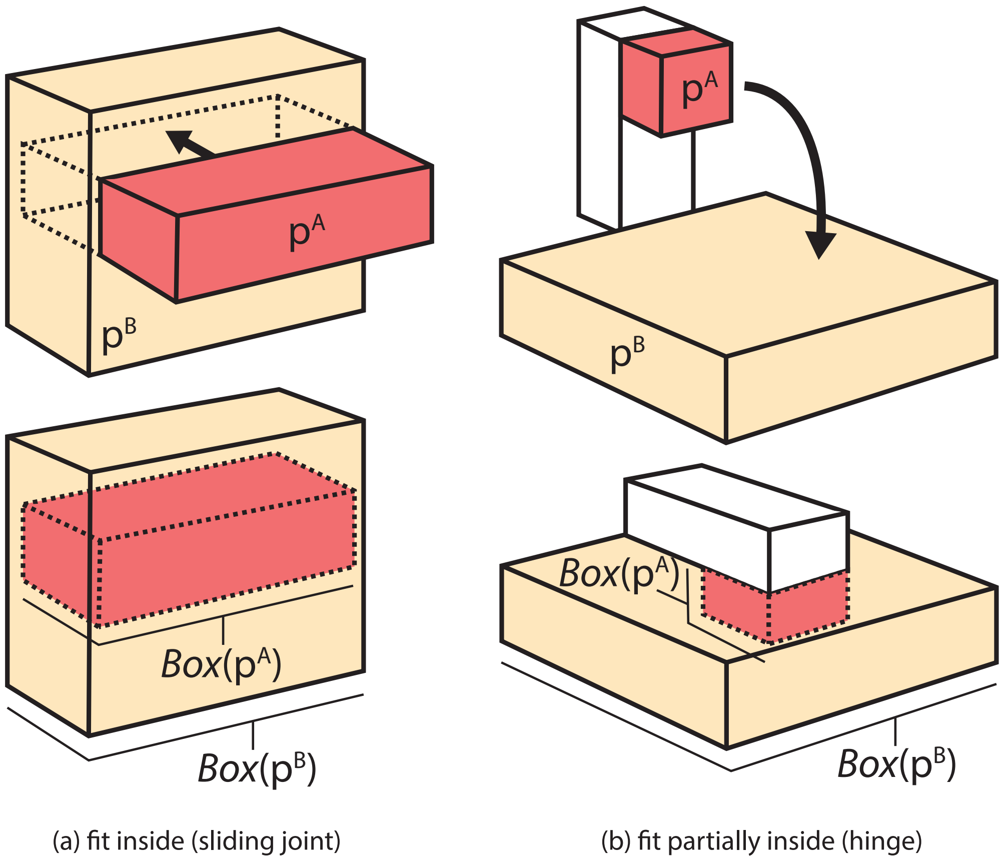

pA
pB
Box(pA)
Box(pB)
(a) fit inside (sliding joint)
pA
pB
Box(pA)
Box(pB)
(b) fit partially inside (hinge)

Figure 4.6: Example fit inside relationships. *Each example shows the inner part p^A and enclosing part p^B in their base and fitting configurations. Setting p^A to be a portion of a part specifies a "fit partially inside" relationship (b).*

constraints must hold:

$$\begin{aligned} Box(P^A)_l &\geq Box(p^B)_l & Box(P^A)_r &\leq Box(p^B)_r \\ Box(P^A)_b &\geq Box(p^B)_b & Box(P^A)_t &\leq Box(p^B)_t \\ Box(P^A)_n &\geq Box(p^B)_n & Box(P^A)_f &\leq Box(p^B)_f \end{aligned} \tag{4.3}$$

where $Box(P^A)$ is the 3D bounding box of the geometric union of parts P^A, $Box(p^B)$ is the 3D bounding box of p^B, and $Box(p)_l$, $Box(p)_r$, $Box(p)_b$, $Box(p)_t$, $Box(p)_n$, $Box(p)_f$ are the left, right, bottom, top, near and far coordinates of a bounding box. Both bounding boxes are defined with respect to the coordinate system of p^B and with all parts in their fitting configuration. To represent a relationship where a part $p^A \in P^A$ "fits partially inside" p^B, we set p^A to be the specific portion of the part that should fit inside p^B (Figure 4.6b). Note that changing any of the inequalities above into an equality constraint indicates that P^A should have enough room to "just fit inside" p^B in one or more dimensions.

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

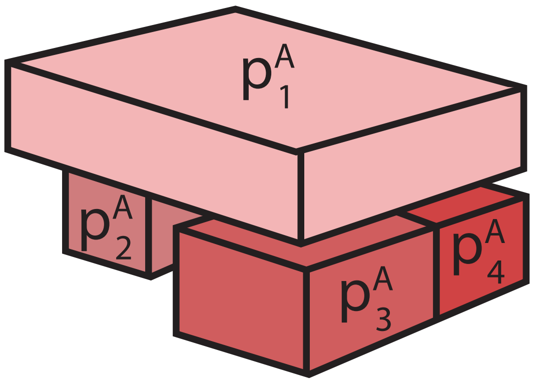


Figure 4.7: Example non-overlapping constraints. *Non-overlapping constraints prevent neighbouring parts from overlapping.*

constraints to adjacent parts:

$$\begin{array}{ll} Box(p_1^A)_b \geq Box(p_2^A)_t & \\ Box(p_1^A)_b \geq Box(p_3^A)_t & Box(p_3^A)_f \geq Box(p_4^A)_n \\ Box(p_1^A)_b \geq Box(p_4^A)_t & Box(p_3^A)_l \geq Box(p_2^A)_r \end{array} \tag{4.4}$$

as the example in Figure 4.7.

4.2.3 Support

In some objects, certain parts are designed to support other parts in specific configurations. For example, in a folding table, the legs support the tabletop when the table is opened. In the simplest case, the relationship stipulates that one of the top faces of a supporting part must be in the same *supporting plane* and in contact with one of the bottom faces of the supported part. However, as with cover relationships, the general case involves a set of faces from multiple parts supporting a set of faces on another collection of parts, based on a corresponding faces graph (Figure 4.8). We say that the faces are in their *supported configuration* when all corresponding faces are in the same plane and in contact. Note that our definition of support does not consider whether the supported part maintains static equilibrium on top of the supporting parts.

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

$$\begin{array}{lll} f_l^A < f_r^B & f_t^A < f_b^B & \\ & & c_t^A = c_b^B \\ f_r^A > f_l^B & f_b^A > f_t^B & \end{array} \tag{4.5}$$

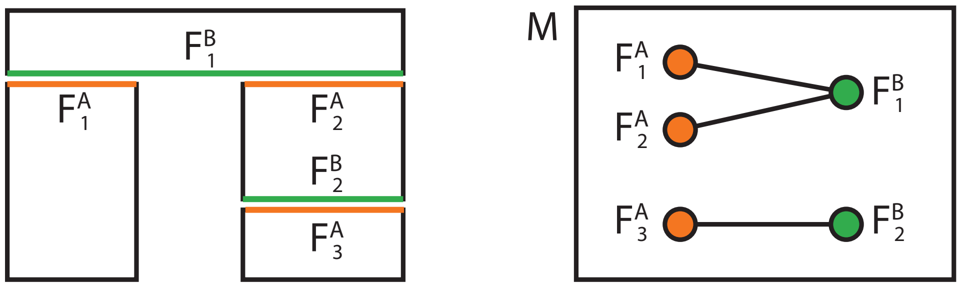


Figure 4.8: Example support relationship. *A set of supported faces (green) sit on top of their corresponding supporting faces (orange).*

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

4.2.4 Flush

Finally, many designs include parts that fit together such that one or more of their faces are flush (Figure 4.9). For example, folding access panels on the side of a printer are usually flush with the printer body when closed. We say that faces are in their *flush configuration* when they lie in the same plane, and we define the relationship as $Flush(f^A, f^B, J, \Theta)$, where f^A and f^B are the two faces that are flush, and $J(\Theta)$ are the set of joints and parameters that put the faces into their flush configuration. The flush relationship constrains the coordinates of the parent cuboids of f^A and f^B such that the two faces are coplanar in the flush configuration.

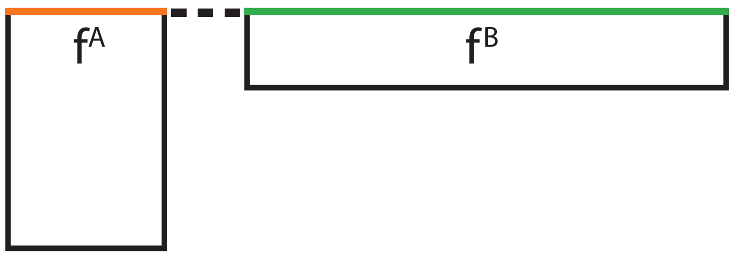


Figure 4.9: Example flush relationship. *Two faces (orange and green) should be coplanar.*

4.3 Specifying Functional Relationships

To help users specify functional relationships for a given design, our system provides interactive tools that automatically infer the appropriate low-level geometric constraints given only a small amount of user interaction (Figure 4.10). The following describes how the users specify the relationships and the system infers the corresponding constraints.

4.3.1 Specifying Cover Relationships

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

Once M has been determined, our system generates the geometric constraints de-

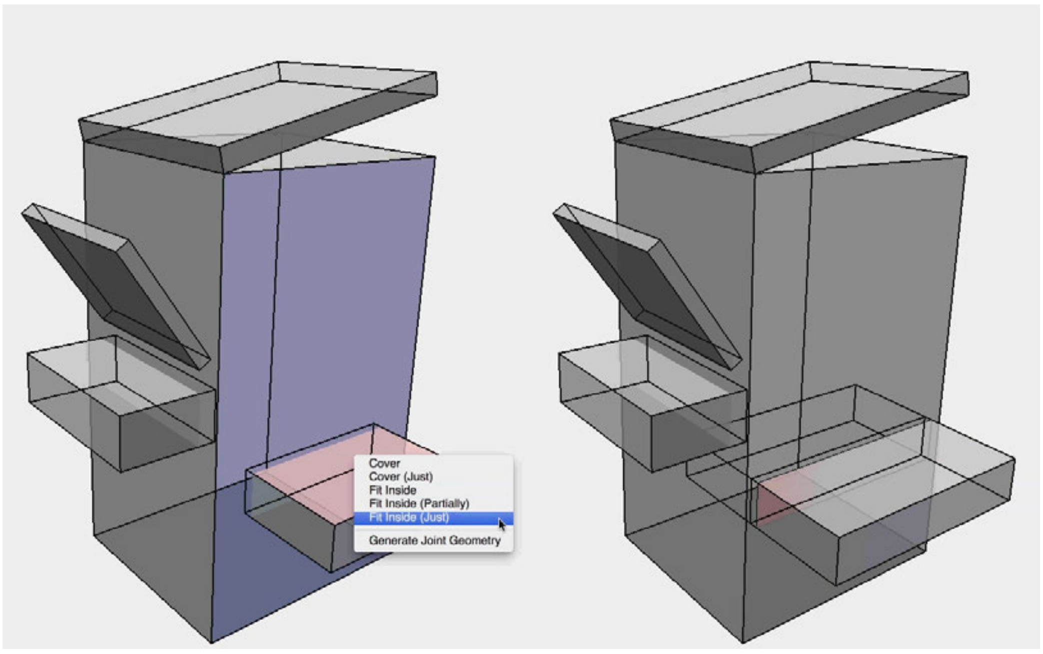
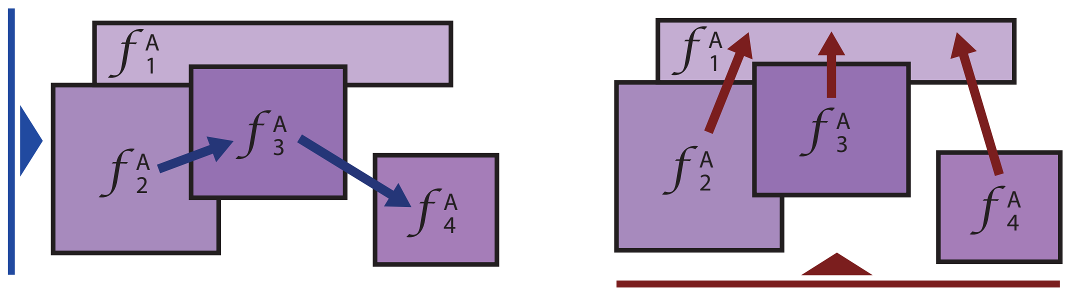


Figure 4.10: Specifying functional relationships. *The user selects appropriate parts and chooses a relationship from the list (left). Then, the system optimises the part parameters to satisfy the constraints for the relationship (right). The figure shows a fit inside (just) relationship. The parts with the blue and red face are the enclosing and inner part respectively.*

scribed in Section 4.2.1 for each set of corresponding faces. To compute all the relevant packing constraints (which eliminate gaps or overlaps between each set of covering or covered faces), we start by determining a spatial ordering over the faces using a plane-sweep approach [NP82]. We sweep a vertical line from left to right and compute intersections between the line and face bounding boxes. We keep track of boxes that intersect overlapping segments of the line (ignoring overlaps that are less than a small threshold) to determine a partial ordering of the boxes in the x dimension (Figure 4.11a). We then sweep a horizontal line from bottom to top to compute a partial ordering in the y direction, ignoring any pair of faces that are already ordered in x (Figure 4.11b). Finally, for any remaining pair of faces that are not ordered in x or y, we sort them in one of the two dimensions based on their centroid coordinates. Given this ordering, we generate packing constraints between consecutive faces in each dimension, and we also constrain the coordinates of the leftmost, rightmost, bottommost and topmost bounding boxes to be equal, which results in a rectangular packing of all the face bounding boxes (Figure 4.11c).

Next, we compute the constraints on the 2D bounding boxes $Box(\mathcal{F}^A)$ and $Box(\mathcal{F}^B)$ defined in Equation 4.1. Since we now have a spatial ordering for each set of covering and covered faces, we can express $Box(\mathcal{F}^A)$ and $Box(\mathcal{F}^B)$ in terms of



(a) horizontal sweep

(b) vertical sweep

$$Box(f^A_1)_l = Box(f^A_2)_l$$
$$Box(f^A_2)_r = Box(f^A_3)_l$$
$$Box(f^A_3)_r = Box(f^A_4)_l$$
$$Box(f^A_4)_r = Box(f^A_1)_r$$

$$Box(f^A_2)_b = Box(f^A_3)_b \quad Box(f^A_2)_t = Box(f^A_1)_b$$
$$Box(f^A_3)_b = Box(f^A_4)_b \quad Box(f^A_3)_t = Box(f^A_1)_b$$
$$Box(f^A_4)_b = Box(f^A_2)_b \quad Box(f^A_4)_t = Box(f^A_1)_b$$

(c) constraints

Figure 4.11: Plane-sweep. *We use a plane-sweep approach to order covering faces horizontally (a) and vertically (b) and then generate constraints between consecutive faces (c).*

their constituent faces as follows:

$$
\begin{aligned}
Box(\mathcal{F}^A)_l &= Box(f_l^A)_l & Box(\mathcal{F}^B)_l &= Box(f_l^B)_l \\
Box(\mathcal{F}^A)_r &= Box(f_r^A)_r & Box(\mathcal{F}^B)_r &= Box(f_r^B)_r \\
Box(\mathcal{F}^A)_b &= Box(f_b^A)_b & Box(\mathcal{F}^B)_b &= Box(f_b^B)_b \\
Box(\mathcal{F}^A)_t &= Box(f_t^A)_t & Box(\mathcal{F}^B)_t &= Box(f_t^B)_t
\end{aligned}
$$

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

4.3.2 Specifying Fit Inside Relationships

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

Based on the specified parts and fitting configuration, our system generates the constraints defined in Section 4.2.2 to ensure that the inner parts P^A fit inside p^B. We use a similar strategy as with the cover relationship constraints. In particular, we use the same plane-sweep approach described earlier, but this time in three dimensions to determine a spatial ordering over all the parts in x, y and z. We then generate non-overlapping constraints between consecutive parts in each dimension and constrain the coordinates of the leftmost, rightmost, bottommost, topmost, nearest and farthest part bounding boxes to be equal. Finally, we rewrite the constraints from Equation 4.3 on the sizes and positions of $Box(P^A)$ and $Box(p^B)$ to obtain a set of linear constraints on the sizes and positions of the individual part cuboids in P^A and p^B.

4.3.3 Specifying Support Relationships

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

Given M, our system automatically generates the geometric constraints described in Section 4.2.3 for each pair of corresponding supporting/supported faces and their parent cuboids. The system also identifies all parts with supporting faces but no supported faces and constrains the bottom coordinates of those parts to be equal.

4.3.4 Specifying Flush Relationships

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

4.3.5 Specifying Additional Geometric Constraints

Although the primary goal of works-like prototypes is to represent mechanical functionality, there may be some aesthetic properties of a design (e.g., symmetry) that the designer wants to enforce. To address such cases, we allow users to directly add low-level geometric inequality and equality constraints between the dimensions of part cuboids. For example, we constrain all the parts in the crate bed prototype (Figure 4.16) to have the same thickness.

4.3.6 Double Pivot Joint Constraints

As described earlier, users add double pivot joints between parts by specifying the positions of the pivots on two coplanar faces. However, in most cases, these initial pivot placements will either cause the parts to interfere as they move between the relevant user-specified configurations and/or end up in the wrong positions/orientations (Figure 4.12a–c). To address these problems, our system automatically generates additional geometric constraints for each double pivot joint. In particular, for each double pivot joint j with pivots u^A and u^B attached to faces f^A and f^B of parts p^A and p^B, we constrain the two pivot positions to ensure that the parts can successfully move between

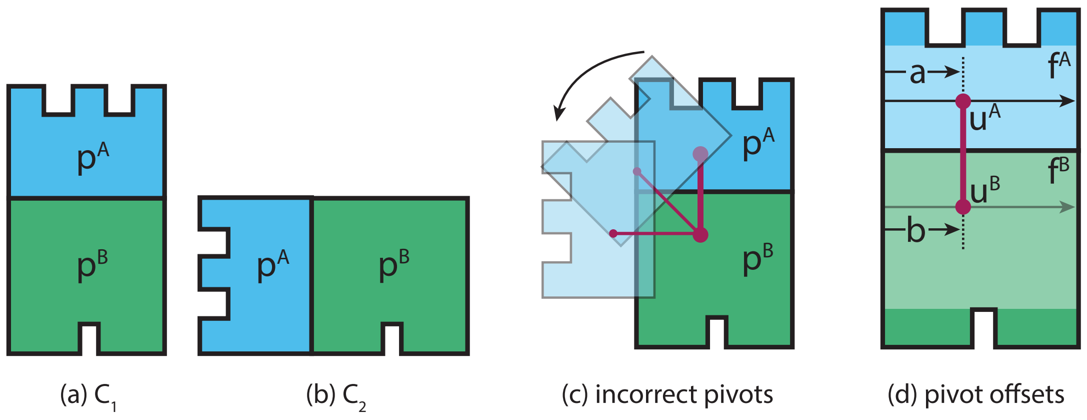


Figure 4.12: Double pivot joints. *Given user-specified part configurations C_1 (a) and C_2 (b), a naive placement of the pivots results in interference and an incorrect ending position for part p^A (c). We parameterise the pivot positions by pivot offsets a and b and impose constraints that enforce good pivot placements.*

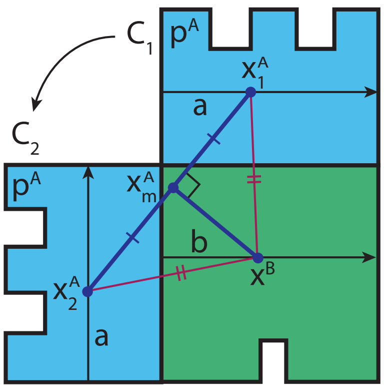


Figure 4.13: Position constraint. *Configurations C_1 and C_2 impose a geometric constraint on the pivot offsets a and b.*

all the relevant configurations. To simplify our formulation, we restrict the possible position of each pivot to an offset vector that passes through the initial user-specified position and is parallel to the cuboid axis that corresponds to the largest cuboid dimension. Thus, we can parameterise the positions of u^A and u^B with scalar offsets a and b (Figure 4.12d). Here, we describe the constraints on the pivot offsets imposed by a single pair of part configurations, C_1 and C_2. If the user specifies additional part configurations, we add the corresponding constraints.

4.3.6.1 Position Constraint

Given the positions and orientations of p^A and p^B in configurations C_1 and C_2, we can derive the following geometric constraint between the pivot offsets a and b. As illustrated in Figure 4.13, any value of a defines two positions x_1^A and x_2^A for pivot u^A that correspond to the two part configurations. Since the distance between u^A and u^B (which corresponds to the rigid link of the pivot) must remain fixed, it follows that u^B has to lie on the perpendicular bisector of the line segment that connects x_1^A and x_2^A. In addition, as explained above, we restrict u^B to lie on its offset vector. Thus, the position x^B of u^B is defined by the intersection of its offset vector and the perpendicular bisector. Based on this construction, we derive the following constraint:

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

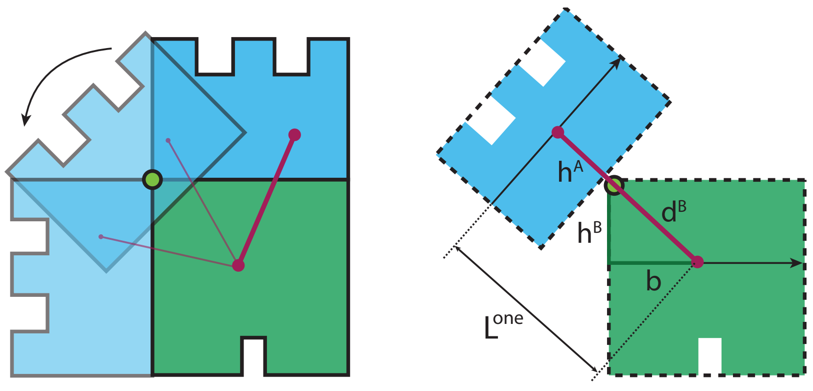
Since x_1^A, x_2^A, x_m^A and x^B all depend linearly on a and b, the constraint is quadratic with respect to the pivot offsets.

4.3.6.2 Motion Constraint

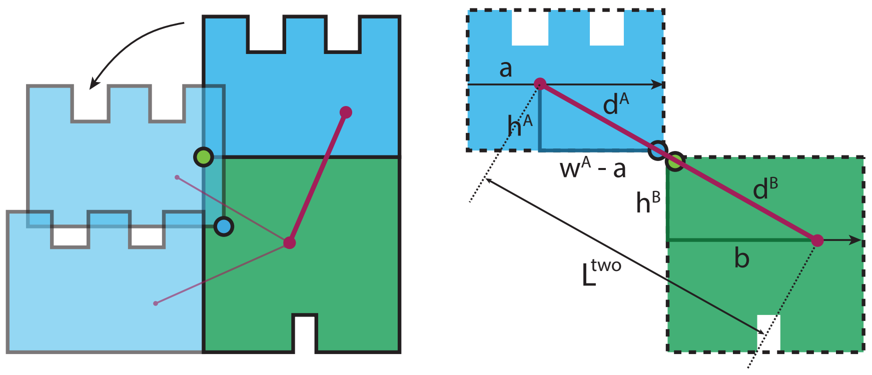
While the position constraint ensures that the pivots can, in theory, move p^A and p^B to the specified configurations, the distance between the pivots may not provide sufficient clearance to allow the two parts to rotate into the appropriate configurations without interfering (Figure 4.14 left). To enforce interference-free motion, we consider the bounding boxes of the two parts projected onto the pivot plane (i.e., the plane that contains f^A and f^B). Based on how the parts move between the two configurations, we determine what combination of bounding box corners the pivot link can potentially pass over, which allows us to derive a conservative lower bound on the length of the link (Figure 4.14 right). We formulate these minimum length requirements as linear constraints over a and b by representing the distance from the pivot positions to the bounding box corners (d^A and d^B in Figure 4.14b) with linear lower bounds. For example, we write the pivot length constraint for the two examples in the figure as follows:

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

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



(a) link length accounts for one corner



(b) link length accounts for two corners

Figure 4.14: Motion constraint. *We automatically generate a motion constraint that ensures the pivot link is long enough to allow the parts to rotate into the appropriate configuration. We determine which bounding box corners to take into account based on the motion of the parts between the start and end configurations.*

4.4 Computing Part and Joint Parameters

Once the user has specified all the necessary functional relationships, we solve the constraint system to update the model. The computation proceeds in two stages. First, we consider the cuboid parameters, which we denote as $B = \{B_i\}$, where B_i represents the left, right, bottom, top, near and far coordinates of the i-th cuboid in the model. We solve for B under all the constraints related to part geometry (Sections 4.3.1–4.3.5) to determine the appropriate size and position of each part for all user-specified configurations. Then, we fix the cuboid parameters and solve for the double pivot parameters, which we denote as $L = \{L_i\}$, where L_i represents the two pivot positions of the i-th double pivot joint. We solve for L under the constraints described in Section 4.3.6. Since each stage updates different sets of parameters, we do not need to iterate.

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

$$E(B, L) = \sum_i \|B_i - \bar{B}_i\|^2 + \sum_j \|L_j - \bar{L}_j\|^2 \tag{4.6}$$

where $\bar{B}$ and $\bar{L}$ are the values of the cuboid and double pivot parameters in the base configuration. With this energy function, we can formulate both the cuboid and double pivot parameter optimisations as quadratic programming problems, which we solve using Matlab's $fmincon$ function. As the user keeps changing part/joint parameters during a design session, the last solution from $fmincon$ function is used for the new initial guess for the function. Please note that the optimisation works well without weights applied in Equation 4.6. This is because all the variables have the same metric. If there is no valid solution, the system tells the user that there are conflicting constraints.

Given that most of our constraints are linear (except for the quadratic position constraint on double pivot joints) and the fact that we typically have a relatively small number of variables, our system solves for both part and joint parameters at interactive rates. This allows users to explore design variations either by modifying the dimensions of any part cuboid, changing the desired configurations of parts, or performing discrete edits such as adding/removing parts or changing their joint types. Once the user performs an edit, the system automatically updates the rest of the model based on the geometric dependencies imposed by the specified functional relationships. Figure 4.15 shows some design variations that we generated with our system.

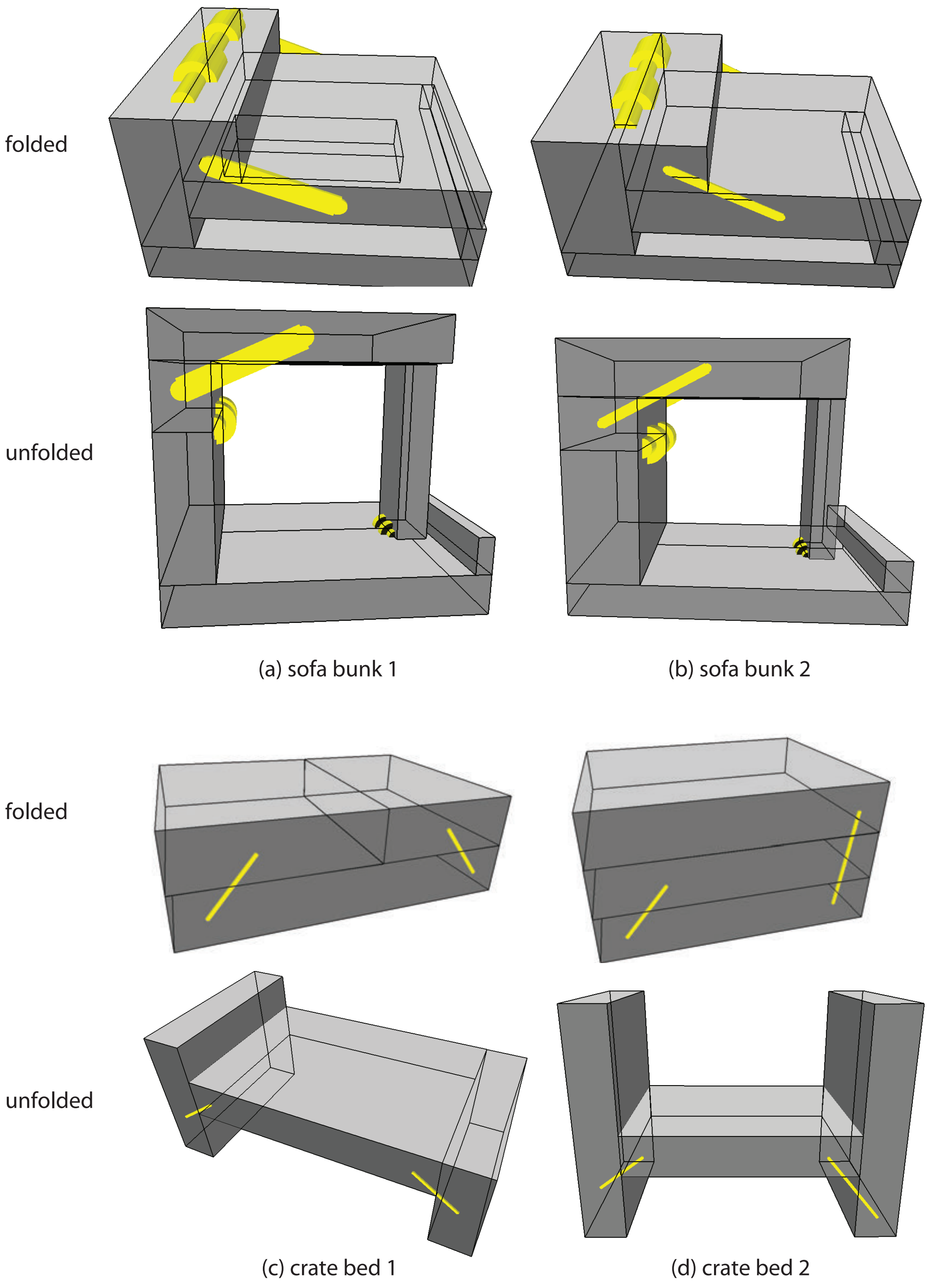


Figure 4.15: Variations of sofa bunk and crate bed. *Our system helps the user explore the design space by modifying part dimensions or configurations.*

4.5 Generating Fabricable Geometry

Once we have determined all the cuboid and double pivot parameters, our system converts the design into fabricable form. This involves three steps: adding cavities to ensure that parts can move into their fitting configurations, generating working joint geometry, and creating gaps where necessary to prevent interferences. We integrated OpenSCAD, a modelling tool where the user writes scripts to generate 3D CAD models, into our system to perform those steps and generate fabricable 3D models.

4.5.1 Cavities

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

4.5.2 Joint Geometry

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

4.5.3 Gaps

Since our joints are designed to be 3D printed in fully assembled form, we add a small gap around all touching faces to ensure that parts do not fuse together and that the support material can be removed after printing. We also add gaps around cavities and all the moving components of the synthesised joints. The size of gaps depends on the precision of 3D printers used and should be calibrated accordingly. Otherwise, the joint might wobble if the gap in the joint is too large. We export the final printable geometry as an STL file that can be sent directly to the 3D printer for fabrication.

4.6 Results

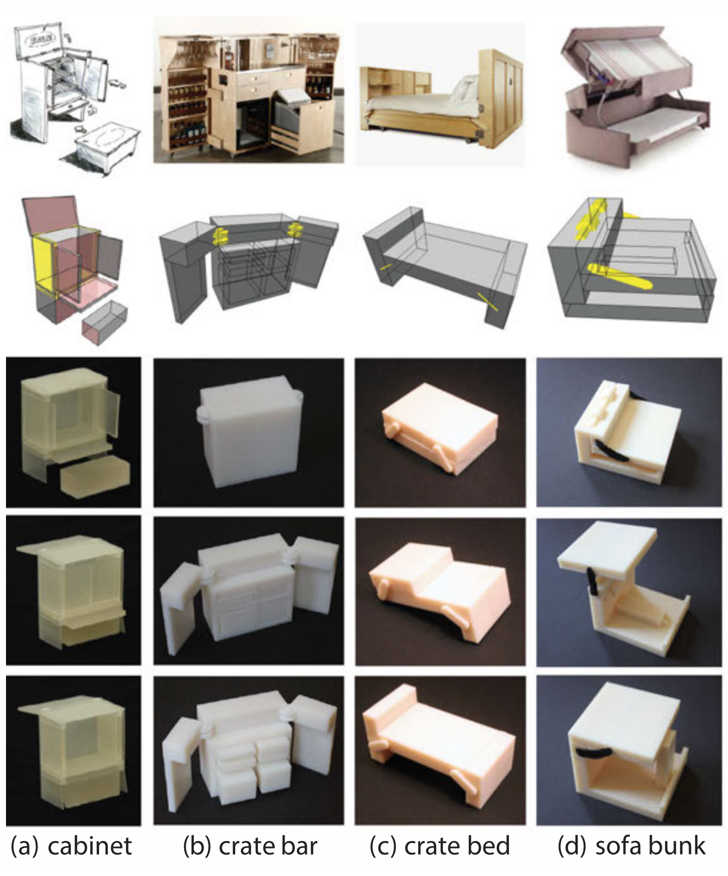
We used our system to create works-like prototypes for 8 different designs (Figure 4.16). Creating each working model in our system took between 1–10 minutes of user interaction. Modelling the parts and adding joints took about three quarters of the time, and the remaining time was spent specifying functional relationships. In addition to cover, fit inside, support and flush relationships, we added 4 geometric constraints: 2 symmetry constraints for the top two cabinet doors and the two folding sections of the crate bar; and 2 to make the folding parts of the tablet and crate bed the same thickness. Table 4.1 summarises various statistics for all the results.

While some of our prototypes may appear simple, mechanisms with even a small number of moving parts often involve non-obvious geometric dependencies. Satisfying these geometric dependencies is the user's responsibility in the existing modelling tools, which is time-consuming and difficult. By automatically maintaining the specified functional relationships, our system helps users create an initial working design from a rough input model and modify that design to generate variations interactively. For example, the crate bed only has three parts, but creating variations of the design with different arrangements of the head and footboards requires non-trivial adjustments to the double pivot joint parameters (Figures 4.15c–d). The folding sofa also has interesting dependencies due to the fact that the top bunk must be supported when the model is unfolded into the bed configuration and then fit together with the headrest and base when folded into the sofa configuration. Figures 4.15a–b show two variations that we created by changing the height of the headrest. The system automatically updated the other part and joint parameters appropriately to maintain the functional relationships. Our system also supports discrete edits. For example, the user can add or remove a part and joint, which leads to changes in remaining parts in different ways depending on which part or joint was manipulated.

While the most common use case for our system is to help designers generate working models that can be 3D printed, the optimised part and joint parameters can also be used as instructions for hand-built prototypes. For example, Figure 4.17 shows a variation of the crate bed design that we built out of paper using measurements com-

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

Table 4.1: Statistics for fabricated prototypes. *Users spent up to 10 minutes to design a working model with various functional relationships in our system (Int. Time: user interaction time, S-Hinge: sliding hinge, D-Pivot: double pivot, Fxn: functional constraints, Geom: geometric constraints).*



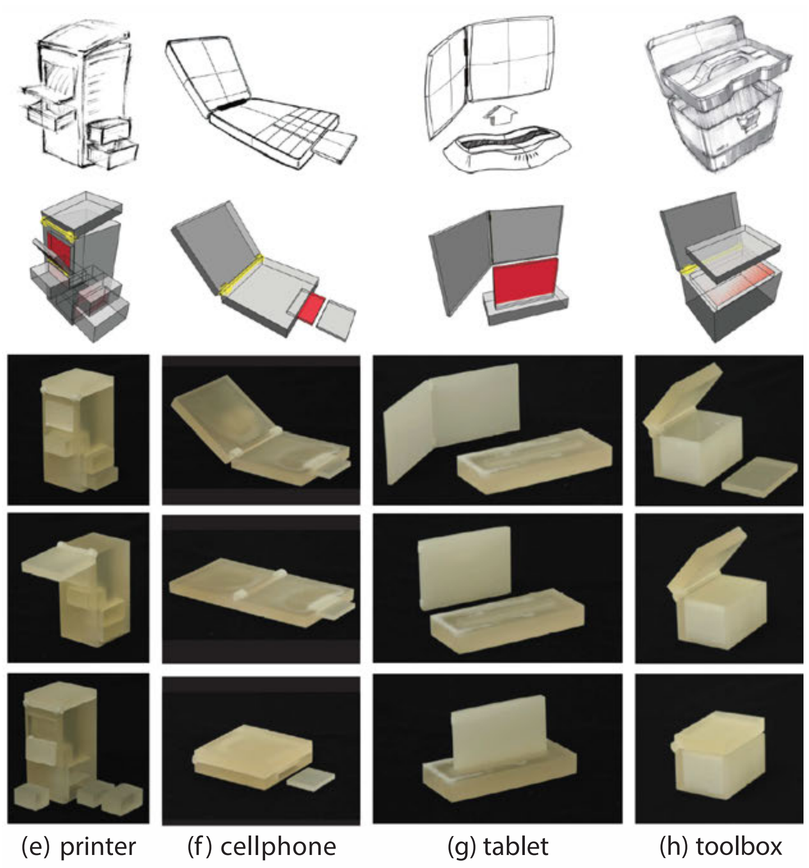


Figure 4.16: Various works-like prototypes created using our system.

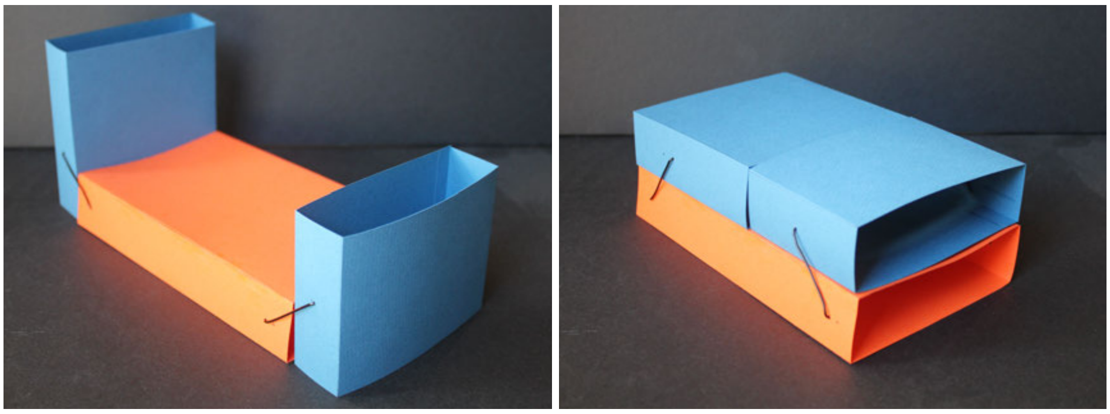


Figure 4.17: Hand-built prototype. *Prototypes can be built by other methods than 3D printing using the measurements from our system, by hand using paper in this case. There will be a new set of challenges that are interesting to address in fabrication using the other methods.*

puted by the system.

4.6.1 Designer Feedback

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

4.6.2 Limitations

The main limitation of our approach is the restricted set of part primitives and joint types that we support in the system, i.e., cuboids for part primitives and only four types of joint. Due to this, we cannot model a bicycle, which requires cylindrical moving parts (wheels) and a new joint type (rotational axes). While compositions of cuboids are sufficient representations for many works-like prototypes, where the functional aspects of prototypes are more important than the appearance, designers sometimes want higher fidelity geometry to understand the relationships between the form and function of a design more precisely. Similarly, other joint types and part interactions could be useful for prototyping certain classes of products, including ball joints, snapping features, threads, and simple gears. Our current system can certainly be extended to handle more complex geometry and a wider range of joints, but this would require modifications to the constraints imposed by our functional relationships.

4.7 Conclusions and Future Work

In this work, we present a new approach to authoring works-like prototypes with functional mechanisms that can be fabricated with 3D printing. By providing a 3D modelling interface where geometry is determined by high-level functional relationships between parts in an object, we allow users to work in a top-down fashion and focus on the functional goals of the design rather than working bottom-up from low-level geometric details, which is unintuitive and difficult. This workflow can help the users accelerate the design process and thus explore the design space more effectively. Our fabricated results demonstrate that our system can generate functional models with only a small amount of high-level user interaction, and the informal feedback from professional designers suggests that our approach could significantly improve existing prototyping workflows. Given the recent advances in 3D printing, we see a huge potential for modelling tools which aim at the physical prototypes that product designers create. Our work takes a small step in this direction, but we see many opportunities for future work in this vein. We also hope that our effort can draw more attention to this paradigm of modelling using high-level goals.

However, other computational fabrication methods such as laser cutting or CNC milling can also pose a different set of important challenges. Especially, because laser cutting and CNC milling are subtractive manufacturing methods, as opposed to 3D printing which is an additive manufacturing method, there can be remaining materials after fabrication. This is not a problem when these leftover materials can be reused for another fabrication, but it is not always possible to do it depending on the shape of the leftover materials. For instance, the remaining materials should probably be thrown away if they take too small, narrow or very irregular shapes. These wasted materials have an economically and environmentally negative impact. Therefore, reducing material waste during fabrication is a problem that is worth exploring as more and more hobbyists employ laser cutting and CNC milling for fabrication.

The next chapter concerns this problem of reducing material waste when fabricating objects using laser cutters or CNC milling machines and our new approach to address the problem will be proposed.

Chapter 5

Guidance by Material Usage

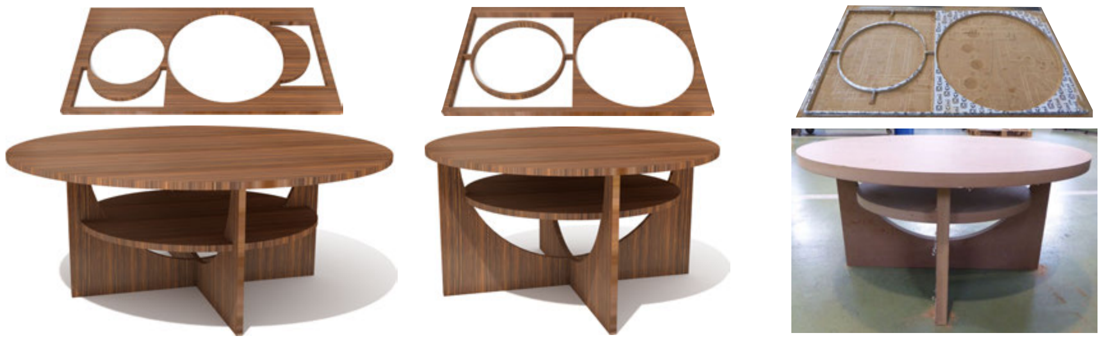


Figure 5.1: Waste-minimising furniture design. *We introduce waste-minimising furniture design to dynamically analyse an input design (a) based on its 2D material usage (see inset) and design specifications to assist the user through (b) multiple design suggestions to reduce material wastage (see inset). The final user design can directly be exported for laser cutting and be assembled (c). In this case, wastage was reduced from* 22% *to* 11%*.*

Design is an exercise in finding a solution to problems in terms of form, function and fabricability (see Section 2.1). Finding a design that solves these by satisfying the constraints on the three elements is not trivial because the elements affect each other in a sophisticated way. Also, there are few available tools that support the user to incorporate the function and fabricability of a design with the form seamlessly and effectively. Typically, therefore, design variations are manually explored by a mixture of guesswork, prior experience and domain knowledge. Without appropriate computational support, such an exploration is often tedious and time-consuming. Especially, it can result in wasteful choices in terms of material usage when we fabricate using subtractive manufacturing methods such as laser cutting or CNC milling (Section 1.1.2).

In furniture manufacturing, both for mass production and for customised designs, material wastage plays a deterrent role. This not only leads to increased production cost (typically 5-15% wastage due to off-cuts), but also hampers ongoing efforts towards green manufacturing [DO09]. For an extensive report, please refer to the guideline for wood waste recycling published by the British Furniture Manufacturers [BFM03]. Hence, there has been a growing interest in *zero-waste furniture* in an effort to reduce material wastage. Computational support, i.e., modelling tools, for designing such waste-reducing furniture, however, is largely lacking.

Material considerations are typically appraised only *after* a shape has been designed. While this simplifies the design process, it leads to unnecessary wastage: at design time, the user can at best guess to account for how the shape will be physically realised, and can easily fail to effectively adjust the design to improve material utilisation.

In recent years, algorithms have been developed to economically 3D print given designs. For example, approaches have been proposed to cleverly break up a given shape into parts that better pack together in print volumes [LBRM12, VGB+14, CZL+15, YCL+15], adaptively hollow shape interiors to save print materials [SVB+12, PWLSH13, WWY+13, DHL14], explore parameter space variations for manufacturable forms [SSM15], or design connector geometry to remove the need for any secondary connector parts [FSY+15]. However, improving material utilisation by explicitly allowing design changes has been less studied.

In this chapter, we introduce the problem of *waste-minimising furniture design*, and investigate it in the context of flatpack furniture design (cf., [GJMB06]) using laser cut wooden parts (Figure 5.1). Specifically, we study the interplay between furniture design exploration and cost-effective material usage. Note that by cost-effective we refer to reducing material wastage during the fabrication process. By directly coupling the design exploration and material wastage minimisation, we empower the users to make more informed design decisions, which leads to economically and environmentally better designs. Note that this is fundamentally different from locking a designed shape, and then trying to best fabricate it.

Figure 5.2 shows the pipeline of our proposed system. The user starts with an initial design. *Design constraints* such as symmetry, desired height, etc. can be specified by the user. Our system analyses *material usage* by computing a dynamic 2D layout of the parts and proposes design modifications to improve material usage without violating specified design constraints, i.e., the user's design intent. Also, *design effectiveness* that determines if the object is usable (e.g., fits in a given region, can hold objects of certain size, etc.) is considered at the same time. Note that such adaptations are often in the form of synchronous movement and shape change of multiple parts affected by both design and material layout considerations, which are difficult to mentally imagine. The user can select any of the suggestions, either in its entirety or in part. She can further update the set of design constraints by locking parts of the current design, and the process continues. Thus, the user scopes out a design space via constraints, and our

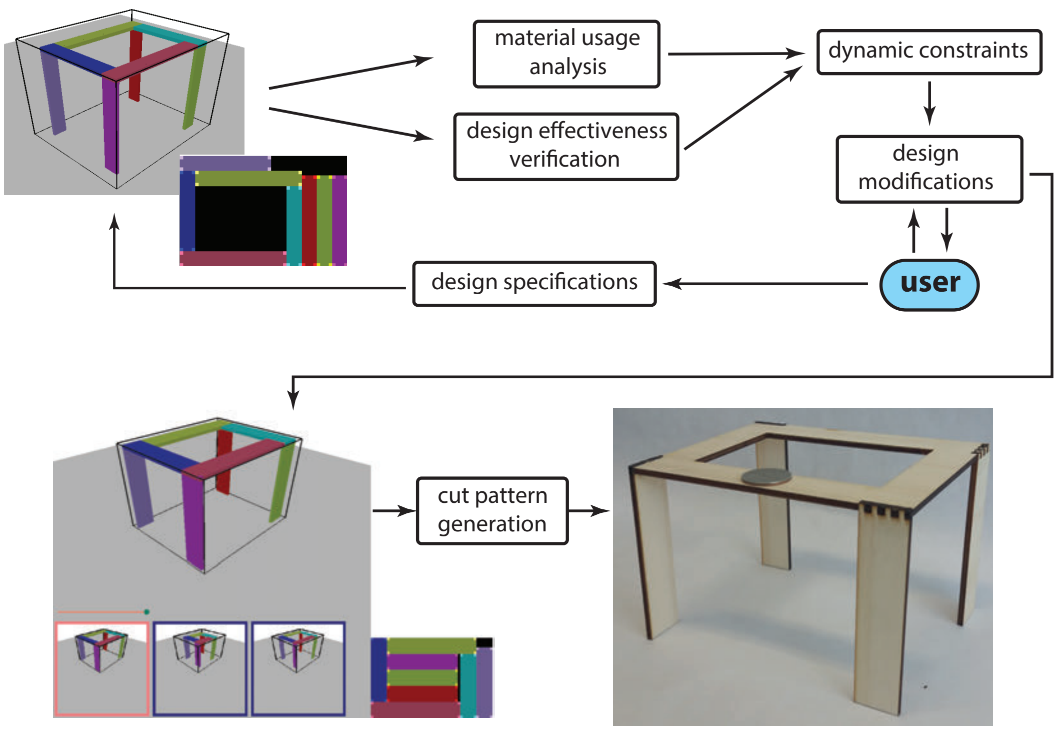


Figure 5.2: System pipeline. *We present an interactive guided-design framework to provide suggestions to the user to adapt her design based on material usage and design effectiveness (i.e., inner volume and outer volume constraints). The cutting code for the final design is then used for the fabrication. Insets show initial and final material space usage with black areas denoting wasted material.*

algorithm refines the design to reduce material wastage while restricting changes to the indicated design space.

Technically, we achieve the above by using the current material layout to *dynamically* discover a set of relevant layout constraints. The algorithm has a discrete aspect involving *which* part to change based on the current 2D layout, and a continuous aspect involving *how* to adapt the part attributes based on the current material space layout without violating user-specified design constraints. Further, we analyse the current design in 3D to judge its effectiveness. Specifically, we support two constraints: (i) objects should fit inside target volumes (outer volume); (ii) shelves should be sufficiently large to hold target objects (inner volume). We enforce the dynamically generated constraints by limiting deformations based on a (local) deformation basis extracted by analysing the material layout constraints and design effectiveness. Even for a fixed design, exploring the space of all possible packings is a combinatorial NP-hard problem. Instead, we locally analyse a set of candidate packings to determine which parts to modify and how to change them to optimise material utilisation. We demonstrate that by dynamically analysing a set of current packings, we can efficiently and effectively couple the 2D layouts and the constrained 3D designs. The user is then presented with different waste-reducing design variations.

We evaluated the system to create a variety of simple and complex designs, and fabricated a selection of them. We also performed a user study with both designers and novices to evaluate the effectiveness of the system. The performance benefits were particularly obvious in case of complex designs involving different design constraints. In summary, we:

- introduce the problem of material waste minimising furniture design; and
- propose an algorithm that dynamically analyses 2D material usage and 3D design effectiveness to suggest design modifications to improve material usage without violating user-specified constraints.

5.1 Design Workflow

Our goal is to propose design variations that minimise material wastage without violating original design intent. In this section, we present the proposed system as experienced by the user, and describe the main algorithmic details in the subsequent sections. Here we particularly focus on how the user encodes her design intent.

Figure 5.3 shows the system interface. We call a sheet of material which is used for cutting a *master board*. The user starts by choosing the desired material (i.e., thickness of wooden parts) and the number and dimensions of the master board(s). The thickness of materials matters because materials that we buy from resellers have only a selection of different thicknesses and the gaps in the joint for assembly of furniture generated by our system depend on the thickness of materials. Multiple master boards can be used for a design that cannot be laid out on a single master board (Figure 5.17d). Or, they are useful when we would like to fabricate multiple copies of the same design, which do not fit in a single master board (Figure 5.4).

Our system considers rectangular master boards — in practice these can represent new boards or leftover rectangular spaces in already used boards. This is because we usually buy materials of rectangular shape and the shape of the remaining material produced by our system is rectangular as well. The user starts by loading an initial part-based 3D object design, either created in a modelling system or as a parametric model. Each part of the object is a plane whose thickness is the same as defined by the user. The parts can be rectangular or have curved boundaries. Alternatively, the user can create the design directly in the system but the shape is limited to rectangular ones in this case. The user designs in *object space*. She can add/delete parts and connect them with other parts using edge- or face-contacts. The user also indicates a set of *design constraints* among the parts, applied to pairs or larger sets of parts. In our implementation, we support: equal edge length (e.g., $l_i = l_j$), sum of edge lengths (e.g., $l_i + l_j + \dots = l_k + \dots$), fixed edge length (e.g., $l_i = c$), equal position, symmetric

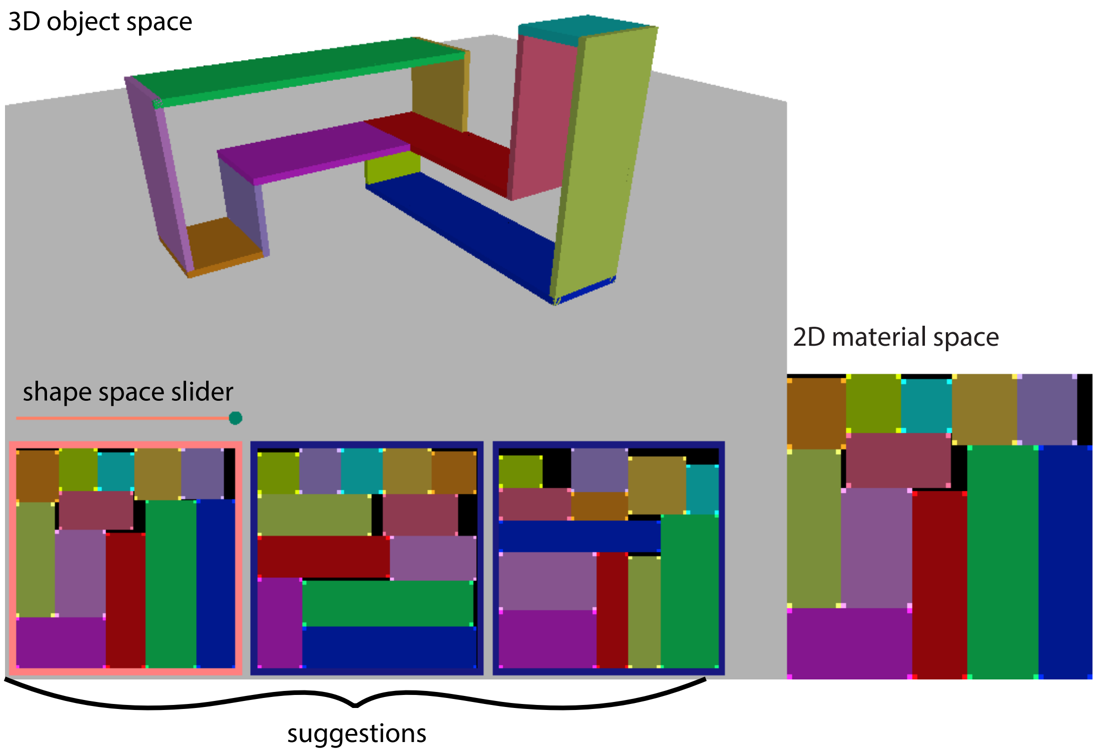


Figure 5.3: System interface. *The user can freely design in the 'object space' pane by adding/deleting parts, or directly editing part dimensions. The user design is dynamically analysed by laying it out in the 'material space' and design adaptations are proposed to improve material usage (i.e., layout-based suggestions), or design effectiveness (i.e., inner/outer volume constraints). The user can select a suggestion, and use the shape space slider to navigate along the proposed edit path. The user can accept a suggestion (or part of it) and continue to edit. Once she is happy with a design, she can directly ask for cutting pattern to be generated along with necessary finger/cross joint specifications.*

parts, ground touching, edge snapping and coplanarity among indicated parts. The user can additionally specify the following usage specifications that are analysed for *design effectiveness* (see Section 5.3.5.1): (i) internal space in the form of *inner volume* indicating minimal shelf dimensions; (ii) environmental specification in the form of *outer volume* where the design should fit. At all times, current usage of the master board(s) is continuously updated using a packing algorithm and shown on the *material space*, and violations of design effectiveness are flagged directly on the object.

The user can update the design by dragging any part directly in the 3D object space. Then, the system suggests multiple design variations that all satisfy the design specifications but achieve different material usages. We measure material usage based on the fraction of the master board(s) utilised. More precisely, the area of parts divided by that of bounding rectangle of the layout. The top three suggestions are presented as thumbnails. If the user mouse-overs any thumbnail, the system animates the proposed design modifications. The user can preview the object- and material-space views, and

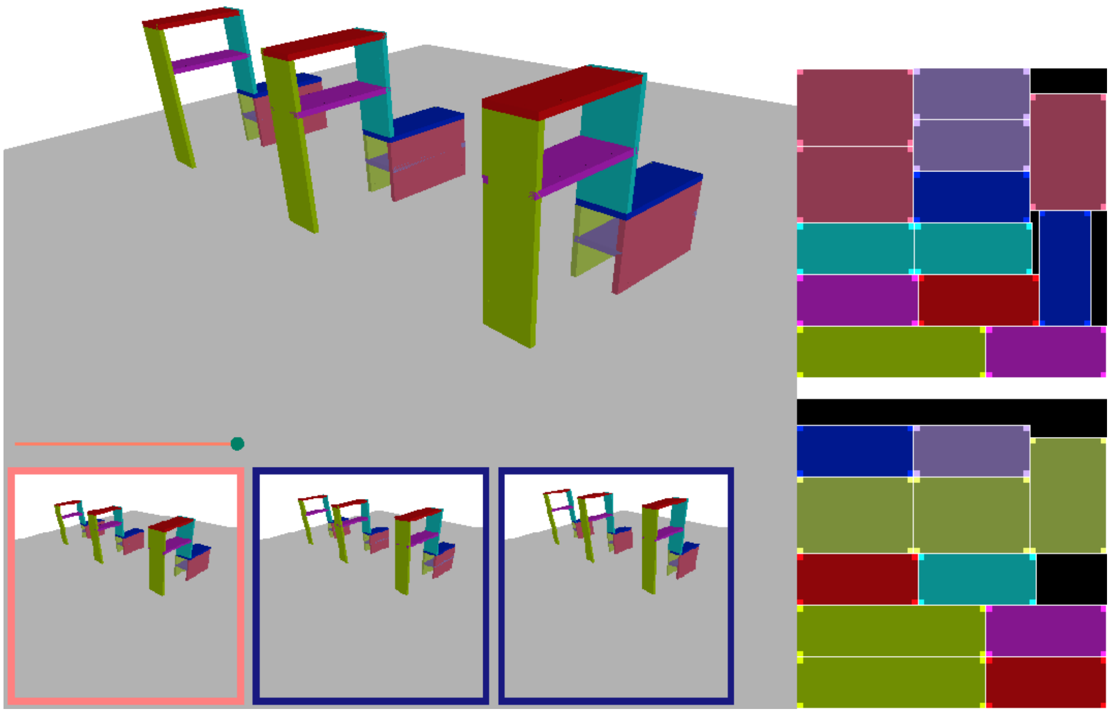


Figure 5.4: Multiple copies of a design. *Multiple master boards are useful when multiple copies of a design cannot fit in one master board.*

select her preferred design suggestion. Note that each thumbnail effectively represents a design exploration path pursued by the algorithm. We provide a slider to move along this path, which is particularly useful for making incremental updates to the design (see Figure 5.5 and Section 5.2).

The user either selects a suggested design variation or picks part configurations from a suggested shape as additional design constraints (e.g., user can lock the proposed sizes of certain parts). Thus, effectively the user appends or updates the current set of specified design. Note that the new constraints are trivially satisfied by the current design, which is critical for subsequent design space exploration (e.g., M_5 is in both shape spaces $\mathcal{S}_1$ and $\mathcal{S}_2$). Then, the user can change the current design again and/or specify additional constraints, and this process repeats.

Once satisfied with a design, she requests for the cutting patterns. She can investigate the design, the material space usage and the cutting patterns, and send the patterns directly for laser cutting.

5.2 Algorithm Overview

Our goal is to analyse design aspects arising from material considerations, and investigate how changes in design affect such considerations. Specifically, we ask how to adapt a furniture design so that it makes better utilisation of material in the resultant

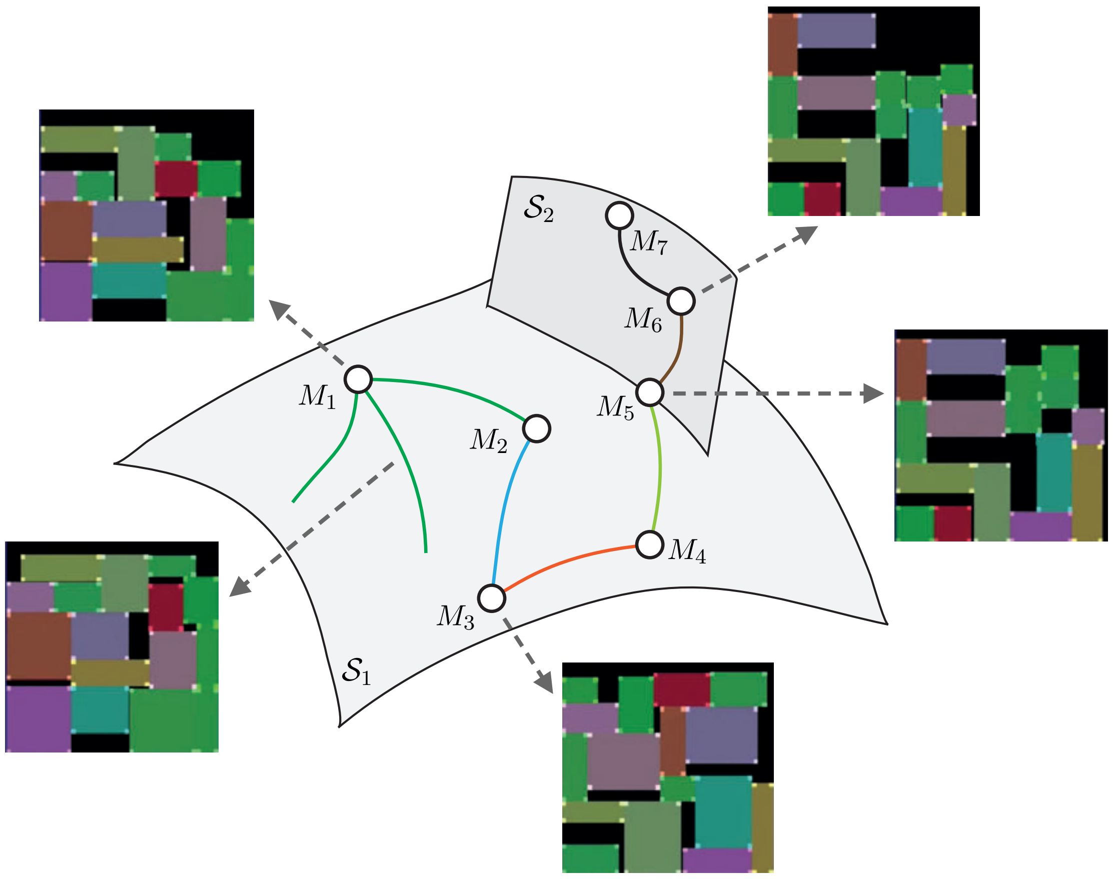


Figure 5.5: Design variations in shape space. *Our algorithm discovers design variations in shape space. The user starts from a design M_1 along with indicated design constraints, and the algorithm seeks for wastage minimising variations by interleaving between topologically different material layouts (indicated by changes in curved paths) or continuous changes to the layouts (indicated by same coloured curves). For example, paths (M_i, M_j) denote continuous design changes, while points M_i denotes designs where new layouts are explored (i.e., branch points). The user can switch to another shape space by picking an updated set of design constraints (shape M_5 here). Note that by construction M_5 belongs to both shape spaces $\mathcal{S}_1$ and $\mathcal{S}_2$. See Algorithm 7.*

design layout. Note that this is the inverse of the design rationalisation problem, i.e., instead of taking a design as fixed and best fabricating it, we adapt the design so that the resultant rationalisation makes better utilisation of available material. First, we introduce some notations.

5.2.1 Parameterised Designs

The design is considered as a function $D(\mathbf{X})$ that produces the geometry of a fixed number of parts, given a configuration vector $\mathbf{X}$. The parts can be assembled into a final furniture design.

We make no assumption as to how D is implemented – we demonstrate in Section 5.5 applications using both constrained based furniture design and parametric designs modelled by CSG. We however expect a continuous behaviour from $D(\mathbf{X})$, i.e.,

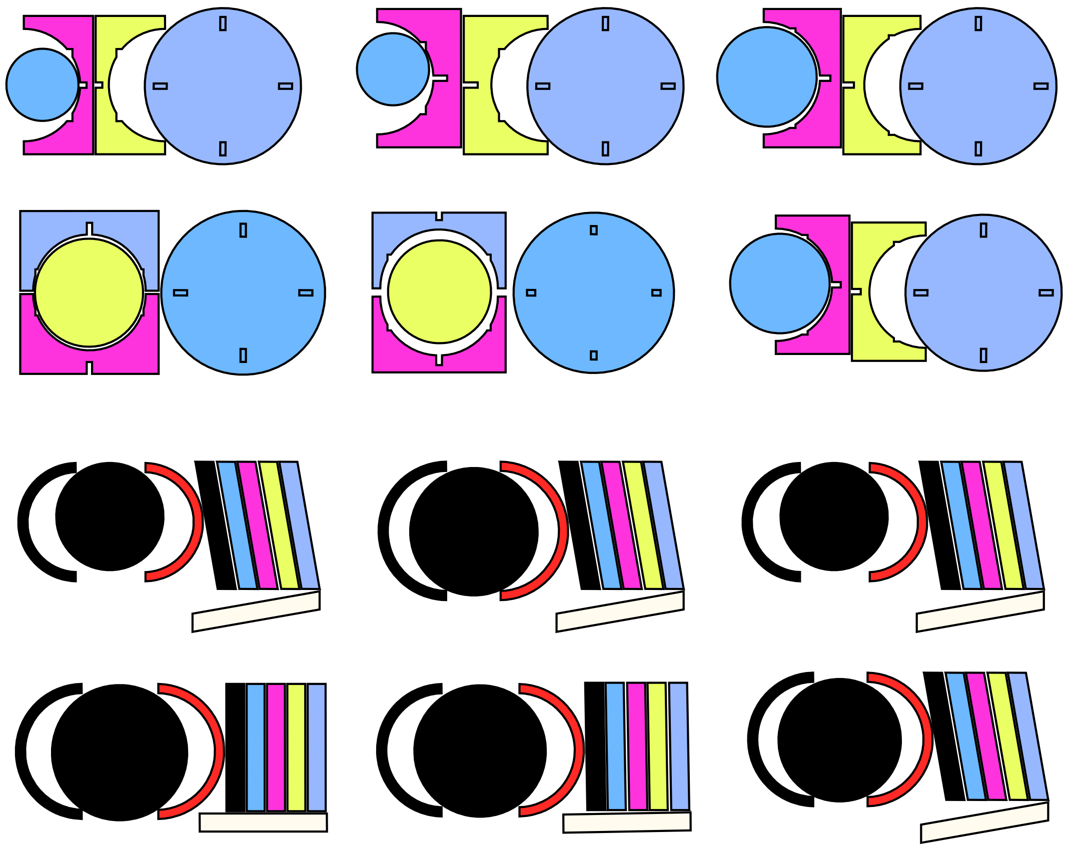


Figure 5.6: Shape variations. *Evolution of shape variations across a run of our algorithm (shown in a clockwise order for each model) on the coffee table (top) and low chair (bottom) models.*

small changes in $\mathbf{X}$ result in small changes in the part shapes. Parametric modellers generally offer such continuity to smoothly navigate the space shape.

During wastage optimisation, our algorithm will change the value of $\mathbf{X}$ so as to explore whether changes in part shapes reduce wastage. Since we focus on laser cut furniture construction, we assume the parts to have the same thickness τ. The parts are thus represented as planar polygonal contours extruded orthogonally.

The geometry of a part p_i lies within a bounding box which we represent by a six dimensional vector encoding the box center $\mathbf{p}_i$ and the lengths of its three sides l_i^x, l_i^y, τ – the Z axis being aligned with part thickness by convention.

5.2.2 Material Space

Since we focus on laser cut furniture, any 3D design given by a configuration vector $\mathbf{X}$ is realised as a layout (i.e., cutting plan) in the material space. Material space is characterised by the largest *master board* that the machine can possibly cut, a rectangle of size $W \times H$. In this space, each part i is associated with a position (u_i, v_i) and an orientation $o_i \in \{0, \pi/2, \pi, -\pi/2\}$.

We use w_i, h_i as extent of a part bounding box in the material space along the x-

and y-axis, respectively. The part box lengths in material space are given by the two part dimensions other than thickness. For a part i, of orientation o_i, we get one of the two cases:

$$\begin{array}{lcll} o_i = 0, o_i = \pi & \Rightarrow & w_i = l_i^x & h_i = l_i^y \\ o_i = -\pi/2, o_i = \pi/2 & \Rightarrow & w_i = l_i^y & h_i = l_i^x \end{array}$$

The material space positions and orientations are variables in the layout optimisation algorithm, alongside the design parameters $\mathbf{X}$ (see Section 5.4).

When wastage is not a concern and a design easily fits within material space, the variables (u_i, v_i, o_i) are independent of the design, i.e., they simply adapt to changes in part sizes. However, as we seek to maximise utilisation of the material space, the material space variables become tightly coupled with the design parameters. Our layout optimiser therefore jointly optimises for material space variables and design parameters to minimise wastage (see Section 5.4)

We next discuss what makes a desirable layout from the point of view of furniture fabrication.

5.2.3 Properties of a Good Design Layout

Rectangular master boards can be sourced in a large choice of sizes and thicknesses from resellers. Therefore, our goal is to achieve a full utilisation of rectangular spaces, so that the user can use boards of exactly the right size and minimise wastage. The machine dimensions determine the maximum extent of a single board.

We measure wastage as the fraction of the space not utilised by the design in its material space bounding rectangle. Ideally, we want to achieve full utilisation, i.e., null wastage.

An ideal packing is one that tightly packs all the parts to perfectly fill up one or more rectangular master boards (like a puzzle). Our system helps the user achieve this by automatically exploring changes improving material space usage (see Figure 5.7).

5.3 Interactive Design Layout Optimisation

This section describes an interactive method where the user can actively explore the design space by changing the design parameters $\mathbf{X}$ and getting immediate feedback from the system while Section 5.4 introduces a more automatic approach. Our approach has evolved over time from the former to the latter. The interactive method allows the user to have the suggestions/feedback from the system immediately as she changes the design. The more automatic method is slower in terms of giving feedback to the user but it can deal with parametric models and more complex shapes with curves (c.f. the interactive method can only deal with rectangles or composites of rectangles).

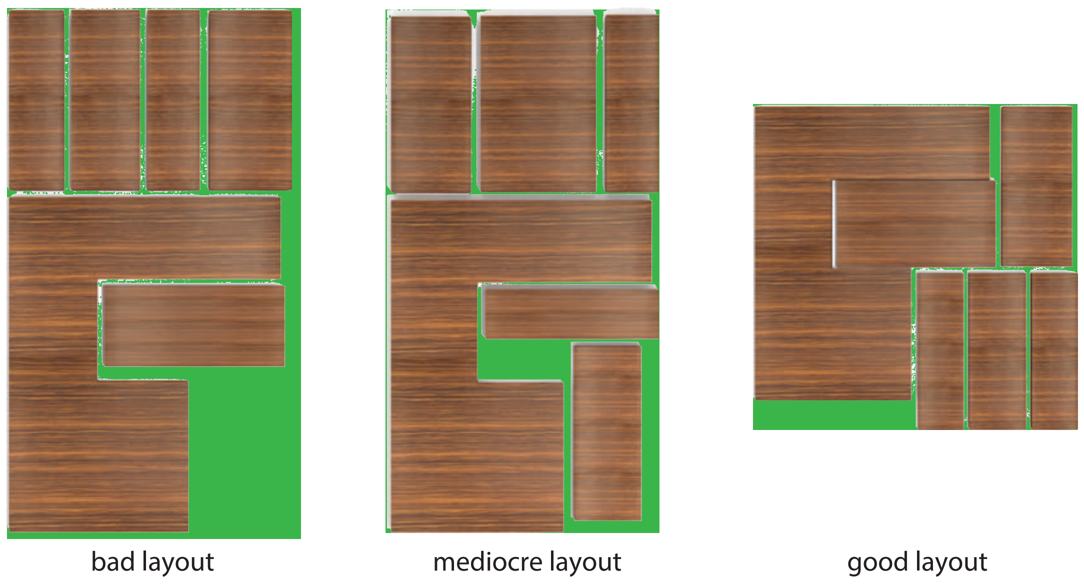


Figure 5.7: Layout refinement. *Examples of stages of layout refinement, from bad to mediocre to good. A good layout is characterised by less area of material wasted (shown in green).*

5.3.1 Initial Design Layout

We initialise a material space packing by using a classic bin packing algorithm with a Bottom-Left heuristic [Cha83]. Rectangular parts are simply inserted in order of decreasing largest extent, which allows smaller parts to fill gaps between previously inserted larger parts. In case of multiple master boards, we randomly assign parts to the different master boards with probability proportional to the remaining board area (among those with sufficient area left). Note that we assume that the initial design fits the master board(s). We run the bin packing algorithm separately for each master board. While a variety of other heuristics exists (see a survey [Jyl10]), we found this approach to work well in conjunction with the rest of our pipeline. Note that our algorithm does not depend on the choice of heuristic, but a poor initial packing will of course complicate the task of the user.

5.3.2 Dynamic Material Usage Constraints

The limited extent of the material space leads to cases where the design no longer fits, especially in the case of complex designs with multiple parts. Our system assists the user in recovering a valid design. Instead of simply reverting and cancelling user edits, we take this situation as an opportunity to improve the packing by *saturating* the material space dimension that has been exceeded.

The complete algorithm for dynamic packing is given in Algorithm 1. In the initial design stage, the user creates the parts while specifying their connector relations. We encode these part dimensions and the spatial arrangement of the parts as a system of m linear constraints. We call these the user-defined design constraints and represent

Algorithm 1: DYNAMIC PACKER

```
Input: Set of parts $\mathbf{X}$, previous successful packing $\mathbf{Q}$, constraints $\mathbf{C}$, s
Output: Packing variables for all parts: $\mathbf{P} = \{..., (u_i, v_i, o_i), ...\}$
P = BinPacker(X);
if IsValid(P) then
    // No need to adjust.
    return P;
active_axis = x;
P = Q;
while not IsValid(P) and less than K iterations do
    $\mathcal{N} \leftarrow$ Expand(Sweep(X,P,active_axis));
    (C,s) = AddSnappingConstraints(C,s,$\mathcal{N}$,active_axis);
    X = Solve(C,s);
    (C,s) = RemoveSnappingConstraints(C,s);
    P = Slide(P,$\mathcal{N}$,active_axis);
    active_axis = (active_axis == x) ? y : x;
if not IsValid(P) then
    // Revert.
    P = Q;
return P;
```

them as a matrix $\mathbf{C}$ of size $m \times n$, and a vector s of m entries. A design satisfies the constraints if and only if $\mathbf{CX} = \mathrm{s}$. We expect $m \leq n$ so that the system is under-constrained, otherwise there is a single solution and no design freedom. The dynamic packing algorithm starts by calling the standard BINPACKER (line 1). If this succeeds, there is no need to further adjust the packing (line 3). If the packing is invalid, we start by detecting the sets of parts which need adjustment. Then, the system makes the parts fit in the material space by adding constraints to the system $\mathbf{CX} = \mathrm{s}$ and solving it. This is performed along each axis in sequence in three steps: *dimension violation detection*, *snapping* and *sliding*. We now detail each of these steps.

5.3.2.1 Dimension Violation Detection

We determine how the parts contribute to the size of the material space layout in Algorithm 1: line 7. This is performed by two sub-algorithms, SWEEP and EXPAND, which detect which parts have to fit side by side (see Figure 5.8). This information is used to identify the parts whose sum of lengths now exceeds the material space dimension along the considered axis.

SWEEP (Algorithm 2) detects the parts which appear on a same line swept in the direction orthogonal to the current axis. Note that this set may appear through transitive neighbouring relationships as illustrated in Figure 5.8. The set is therefore expanded to obtain all combinations of parts that add up along the considered axis.

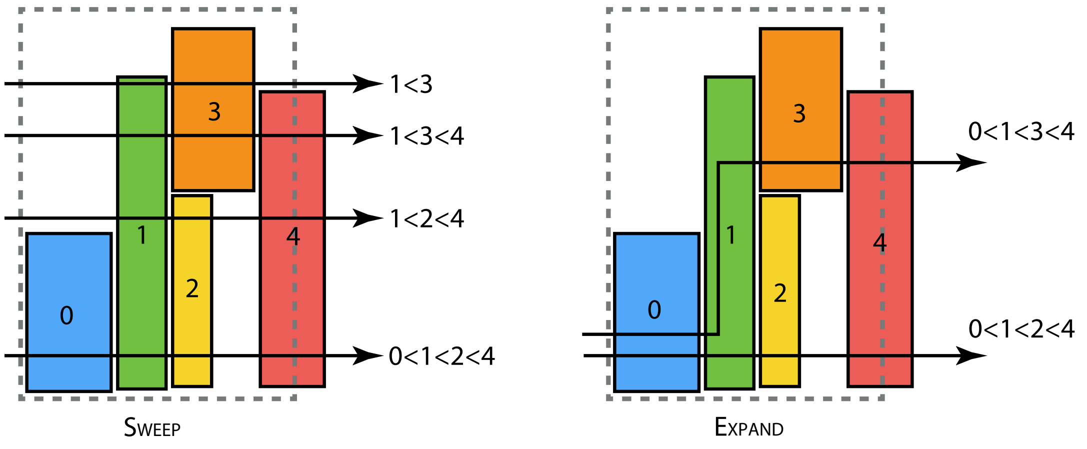


Figure 5.8: Transitive neighbouring relationships. *Given a material space layout, we perform* SWEEP *to extract a set of constraints, and then unwind them to make explicit the transitive relations using the* EXPAND *step. In this example, we only show* SWEEP *and* EXPAND *along the x-axis. Note that at the end of this stage, we get a set of dynamically generated layout constraints to ensure that the parts remain inside the master board.*

This is performed by EXPAND (Algorithm 3), which unfolds the dependencies between parts to obtain a new set of part lists.

In line 7, we utilise the previous successful packing to detect neighbouring relationships between parts. Note that we ensure that each newly added part can be fit in the material space to guarantee the existence of an initial packing.

5.3.2.2 Snapping

We use the neighbouring relationships computed by SWEEP and EXPAND to verify whether the sum of part lengths exceeds the material space dimensions. For each part combination for which it is the case we add a snapping constraint making the sum of their lengths equal to the material space extent in order to make the parts fit in the material space again.

Snapping is performed in lines 8 to 10 in Algorithm 1. A snapping constraint is added between all parts whose combined sum exceeds the material space dimensions along `active_axis` axis being considered by ADDSNAPPINGCONSTRAINTS. The design is then solved for, using the algorithm SOLVE described in Section 5.3.5. The constraints are subsequently removed after the part lengths have been recomputed.

ADDSNAPPINGCONSTRAINTS and REMOVESNAPPINGCONSTRAINTS are adding/removing the snapping constraints for *all* part lists whose sum exceeds the material space dimensions along the axis being considered, i.e., `active_axis`. Snapping provides us with a new set of part lengths that fits the material space along the axis.

Algorithm 2: SWEEP

```
Input: Set of parts $\mathbf{X}$, packing $\mathbf{P}$, active axis $a$
Output: Set $\mathcal{L}$ of lists of parts appearing on a same line along $a$
$E \leftarrow \emptyset$;
foreach part $p$ do
    $m_p =$ min coordinate of $p$ along $a$ in $\mathbf{P}$;
    $M_p =$ max coordinate of $p$ along $a$ in $\mathbf{P}$;
    $E \leftarrow (m_p, in, p)$;
    $E \leftarrow (M_p, out, p)$;
if $E = \emptyset$ then
    return $\emptyset$;
sort $E$ by increasing lexicographic order $(in < out)$;
$\mathcal{L} \leftarrow \emptyset$;
$\mathcal{A} \leftarrow \emptyset$;
$(x, t, p) \leftarrow$ pop first from $E$;
while true do
    repeat
        if $t = in$ then
            $\mathcal{A} \leftarrow \mathcal{A} \cup \{p\}$;
        else
            $\mathcal{A} \leftarrow \mathcal{A} \setminus \{p\}$;
        if $E = \emptyset$ then
            return $\mathcal{L}$;
        prevx = x;
        $(x, t, p) \leftarrow$ pop first from $E$;
    until $prevx \neq x$;
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{A}\}$;
```

5.3.2.3 Sliding

We recover positions by sliding each part to the leftmost position that does not overlap with a part located before, as given by the neighbouring relationships. This is done line 11 in Algorithm 1, by calling SLIDE (Algorithm 4). The process is illustrated in Figure 5.9.

This simple approach guarantees that no overlap exists along the x-axis, and that the dimensions no longer exceed the material space. However, after sliding, parts might still overlap along the y-axis. We therefore iterate the process, this time along the y-axis.

5.3.2.4 Termination

The dynamic packing algorithm typically terminates in one iteration as material space violations are quickly identified as we continuously check from them in the background. There are two failure cases: First, if decreasing the dimensions of some parts

Algorithm 3: EXPAND

```
Input: Set of parts $\mathbf{X}$, packing $\mathbf{P}$, active axis $a$, set $\mathcal{I}$ of lists of parts appearing
       on a same line along $a$
Output: Expanded set $\mathcal{O}$ of lists of parts appearing on a same line along $a$
1  foreach $\mathcal{A} \in \mathcal{I}$ do
2      prev $\leftarrow \emptyset$;
3      foreach part $p \in \mathcal{A}$ // In order along $a$.
4       do
5          if prev $\neq \emptyset$ then
6              Succ[prev] $\leftarrow$ Succ[prev] $\cup$ {p};
7          Pred[p] $\leftarrow$ prev;
8          prev $\leftarrow p$;
9  $\mathcal{O} \leftarrow \emptyset$;
10 foreach part p such that Pred[p] = $\emptyset$ do
11     push(Stack, (p,$\emptyset$));
12     while Stack not empty do
13         (q,$\mathcal{B}$) $\leftarrow$ pop(Stack);
14         if Succ[q] = $\emptyset$ then
15             $\mathcal{O} \leftarrow \mathcal{O} \cup \{\mathcal{B}\}$;
16         else
17             foreach $s \in$ Succ[q] do
18                 push(Stack, (s, $\mathcal{B} \cup \{s\}$));
19 return $\mathcal{O}$;
```

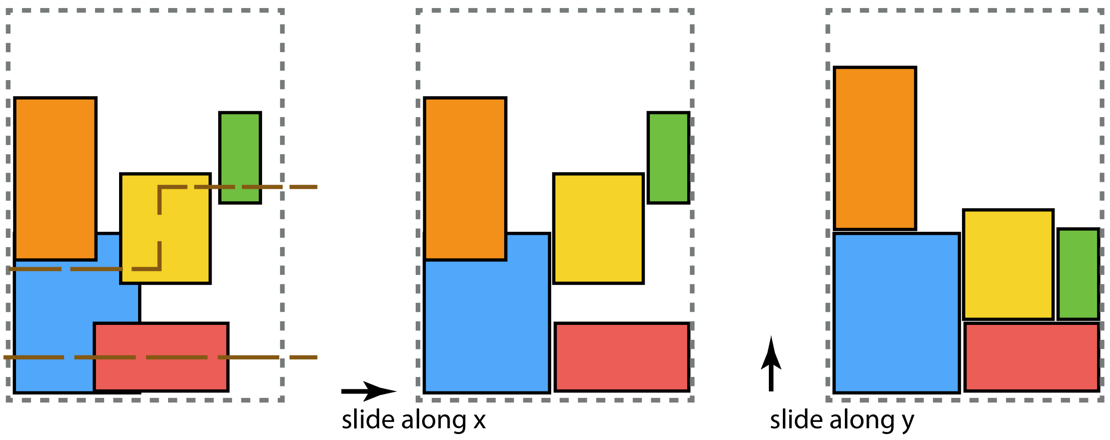


Figure 5.9: Sliding. *The dynamic layout constraints (shown with dashed brown lines) detected using the* SWEEP *and* EXPAND *steps are used to solve for new part parameters. (Left) These updated parts can overlap in the material space. (Middle-right) We use* SLIDE *routine to reposition them first along x-direction, and then along y-direction to arrive at a non-overlapping layout.*

Algorithm 4: SLIDE

```
Input: Set of parts $\mathbf{X}$, packing $\mathbf{P}$, active axis $a$, set $\mathcal{L}$ of lists of parts
       appearing on a same line along $a$
Output: Modified packing so that parts do not overlap along $a$
1 foreach $\mathcal{A} \in \mathcal{L}$ do
2     pos = 0;
3     foreach part $p \in \mathcal{A}$ // In order along $a$.
4     do
5         var ← variable for position of $p$ along $a$ in $\mathbf{P}$;
6         if value(var) < pos then
7             value(var) ← pos;
8         sz = dimension of $p$ along $a$ in $\mathbf{P}$;
9         pos = pos + sz;
```

increases the dimensions of others in the design, the algorithm may result in a race condition. This is, however, discouraged as the constraint solver (Section 5.3.5) prioritises changes to positions before changes to part lengths. Second, in extreme cases, the parts may reach their minimal sizes, preventing any further snapping. In such a scenario, the algorithm is terminated after a fixed number of iterations (four in our tests), and revert to the previous successful packing to cancel user changes.

5.3.3 Suggestions Based on Design Layout

While the user edits the design, our system constantly searches for possible improvements to the design layout in material space. This is done by exploring the design space in multiple parallel threads (maximum of 8 threads). Each thread traverses the part list in a different, random order. For each part, it attempts to reduce and enlarge the part length in all dimensions while Algorithm 1 is called at each change in part sizes.

Whenever an improving move is found the algorithm continues modifying this same variable until the design layout stops improving or a constraint is violated. This has two purposes: First, focusing the change on a few parts makes it easier for the user to understand the changes made by the algorithm. Second, this increases the differences between the designs obtained by random exploration of different threads.

To ensure that the suggestions are sufficiently different from one another, we launch multiple random explorations and select only three to expose to the user. We use farthest point based picking to select diverse options to the user. Note that we only compare part lengths in this selection process.

5.3.4 Handling Coplanar Parts

Up to this point we only considered parts as independent rectangles. However, many designs require combinations of rectangular coplanar parts, as illustrated in Figure 5.10.

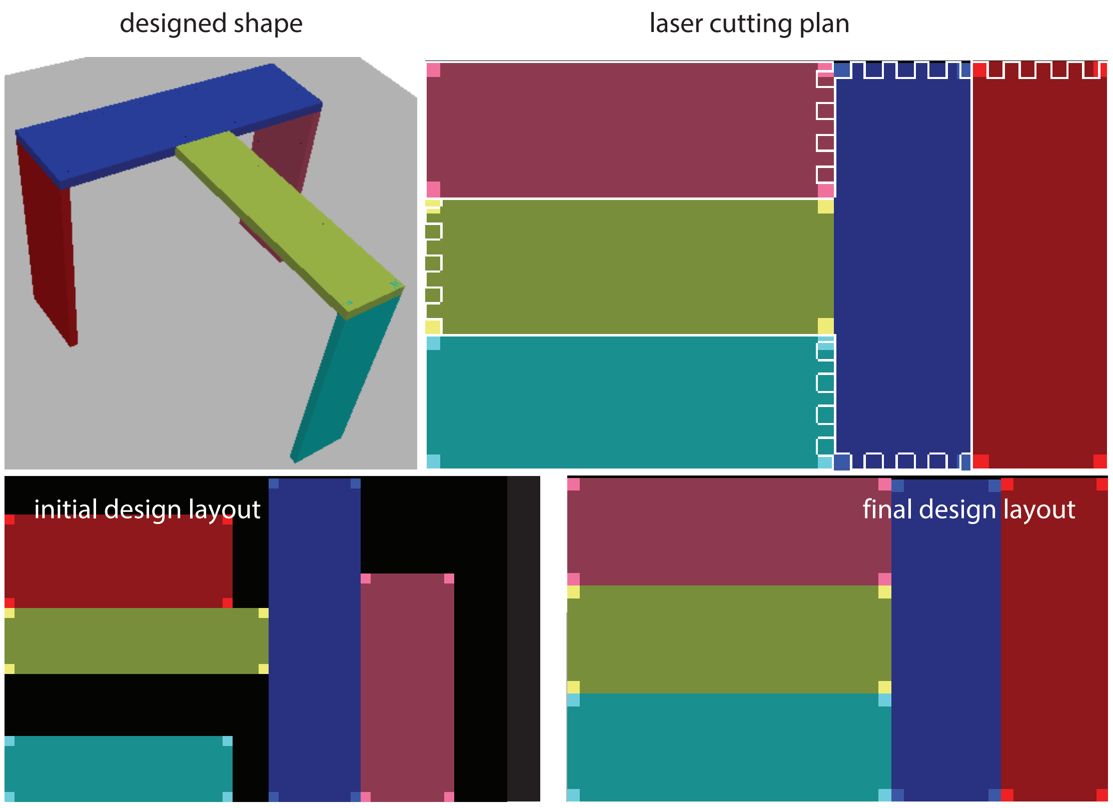


Figure 5.10: Coplanar parts. *We support designs (top-left) involving coplanar (in 3D) parts with shared edges. In this example, the T-shape is formed by the green and blue part. In the material space, we maintain the relative configuration among such coplanar pieces, while still allowing other parts to interleave and fill in the in-between spaces. Bottom row shows the material space configuration before and after guided-design, while top-right shows the final cutting plan. Note that the T-shape is cut as a whole.*

For the sake of clarity we simply refer to each such configuration as *coplanar parts*, a set of parts that are in the same plane and in contact so that they form a single connected component. For each such set we will consider the bounding rectangle of the configuration, which is the bounding rectangle of the union of all the coplanar parts.

The support of coplanar parts requires an adaptation of two of our algorithms: BINPACKER and SLIDE. The packer is modified to proceed in two steps. In the first step the coplanar parts are packed using the bounding rectangle of their configuration. In the second step, all the remaining (simple) rectangles are added to the material space layout. They are free to fill in voids *within* the coplanar part configurations. This process is illustrated Figure 5.10.

The sliding algorithm is changed so that whenever a part slides, all the parts in the same coplanar configuration are moved accordingly. A slight complication to this algorithm is that it requires several iterations to ensure all the dependencies are properly taken into account. It is otherwise very similar to Algorithm 4.

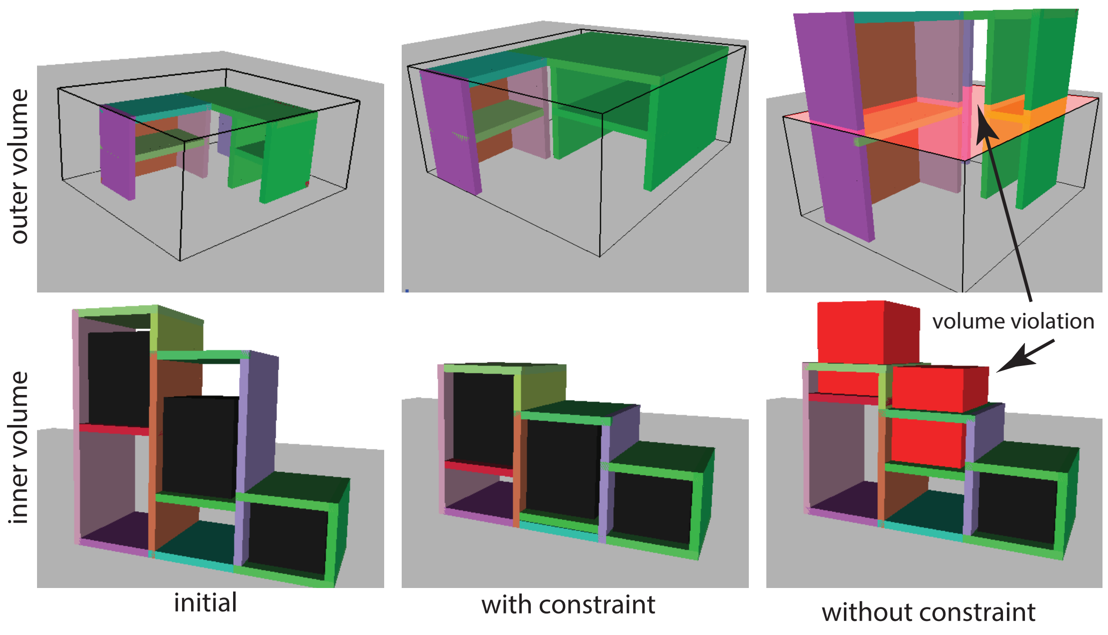


Figure 5.11: Design effectiveness constraints. *We show effects of designing with (middle column) or without (right column) the respective constraints activated. First row: the cuboid in black line stipulates the upper bound volume of the design if the constraint is enabled. Without the constraint, a suggested design can be larger than intended (right column). Second row: the black boxes stipulate the lower bound volume between parts in the design if the constraint is enabled. It is useful when we need certain space between parts in the design.*

5.3.5 Furniture Design with Constraints[1]

During interaction with the system, either by the user or an algorithm optimising the design, new design points $\mathbf{X} + \mathbf{u}$ that violate the constraints are produced, with $\mathbf{u}$ a vector of changes imposed. Typically, $\mathbf{u}$ has only few non-zero entries, e.g., the user or the optimiser increases a single part length. We define a boolean function $E(\mathbf{X})$ with a value of false denoting that design effectiveness constraints have been violated. Note that $\mathbf{X} + \mathbf{u}$ might violate either the design effectiveness constraints (i.e., $E(\mathbf{X} + \mathbf{u}) = false$), or the design constraints (i.e., $\mathbf{C}(\mathbf{X} + \mathbf{u}) \neq \mathrm{s}$), or both. We seek a correction vector $\mathbf{d}$ such that $\mathbf{X} + \mathbf{u} + \mathbf{d}$ is a valid design point which preserves the changes applied by the system, i.e., $d_i = 0$ for all i such that $u_i \neq 0$.

Our approach is based on two nested algorithms. The outer one solves for design effectiveness constraints (see Section 5.3.5.1), while the inner one solves for design constraints (see Section 5.3.5.2).

5.3.5.1 Design Effectiveness Constraints Enforcement

Our system supports two types of design effectiveness constraints: inner volume and outer volume constraints (Figure 5.11). The inner volume constraint preserves space between parts while the outer volume constraint prevents the bounding volume of the

[1]The work in this section is done by Jean Hergel (INRIA Nancy - Grand-Est).

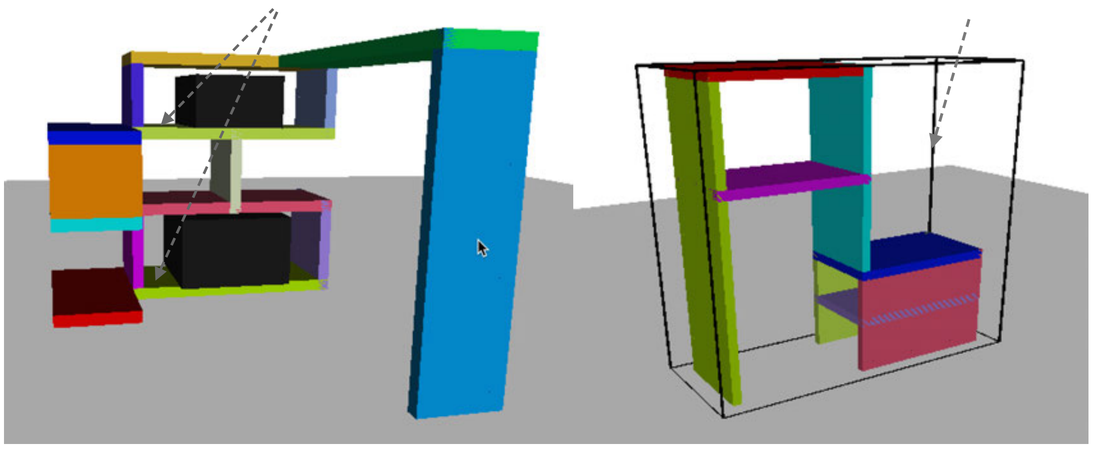


Figure 5.12: Specifying design effectiveness constraints. *Left: For inner volume constraints, the user selects parts (marked by arrows in the figure) and puts the boxes on them. Using the keyboard, she can change the size of the boxes, which corresponds to the inner volume to be preserved. Right: For outer volume constraints, using the keyboard the user can change the size of the bounding box (marked by an arrow in the figure), which the design must not exceed.*

design from growing larger than specified by the user. The user can set the inner volume constraints by placing boxes and changing their dimensions which reflect the required inner volumes (Figure 5.12 left). She can also specify the outer volume constraint by adjusting the size of bounding box of the design which stipulates the upper bound volume of the design (Figure 5.12 right).

We observe that all the effectiveness constraints can be enforced by dynamically adding or removing equality constraints. For instance, when a part is about to enter an inner volume, a constraint can be inserted to maintain the part at the correct distance. A key difficulty is when and in which order to insert the constraints. Note that a straightforward approach can quickly over-constrain the problem, preventing to find a feasible solution.

Our scheme is inspired by collision detection in physics simulations, where all contacts within a potentially large time step have to be found. The intuition is to detect the first time a constraint violation occurs, while progressively applying the change vector $\mathbf{u}$. That is, we parameterise the problem with a variable $\alpha \in [0, 1]$ and find the smallest value α_1 such that $E(\mathbf{X} + \alpha_1 \mathbf{u}) = false$. We then add appropriate equality constraint to $\mathbf{C}$, s so that E evaluates to true again. The algorithm then recurses until reaching $\alpha = 1$, or encountering contradictory constraints (see Section 5.3.5.3). During this entire process we take care of enforcing design constraints while exploring along $\mathbf{u}$. All constraints added during the process are discarded upon termination. The full algorithm is given in Algorithm 5.

The function `CorrectForDesignConstraint` produces a correction vector recovering from design constraint violations, and is described in Section 5.3.5.2. The

Algorithm 5: DYNAMICEFFECTIVENESSCONSTRAINTS

```
Input: Valid design $\mathbf{X}$, vector of changes $\mathbf{u}$, position $\alpha \leq 1$
Output: A correction vector $\mathbf{d}$ such that $\mathbf{C}(\mathbf{X}+\mathbf{u}+\mathbf{d}) = \mathbf{s}$ and $d_i = 0$ where
        $u_i \neq 0$ and $E(\mathbf{X}+\mathbf{u}+\mathbf{d}) = true$
1  $\mathbf{d} \leftarrow$CorrectForConstraint($\mathbf{C}, \mathbf{s}, \mathbf{X}, \mathbf{u}$);
2  if $E(\mathbf{X}+\mathbf{u}+\mathbf{d})$ then
       // Found a solution, return.
3      return $\mathbf{d}$;
   // Search for first violation, bisection on $\alpha$.
4  $\mathbf{l} \leftarrow \alpha$;
5  $\mathbf{r} \leftarrow 1.0$;
6  while $|l - r| > \epsilon$ do
7      $\mathbf{m} \leftarrow (\frac{l+r}{2}) \cdot \mathbf{u}$;
8      $\mathbf{d} \leftarrow$CorrectForDesignConstraint($\mathbf{C}, \mathbf{s}, \mathbf{X}, \mathbf{m}$);
9      if $E(\mathbf{X}+\mathbf{m}+\mathbf{d})$ then
10         $l \leftarrow \frac{l+r}{2}$;
11     else
12         $r \leftarrow \frac{l+r}{2}$;
13 $\mathbf{D}, \mathbf{t} \leftarrow$AddDynamicConstraints($\mathbf{X} + l \cdot \mathbf{u}, \mathbf{X} + r \cdot \mathbf{u}, \mathbf{C}, \mathbf{s}$);
14 if $\mathbf{D}, \mathbf{t} = \mathbf{C}, \mathbf{s}$ then
       // No new constraint could be added: return best found so
          far.
15     return $l \cdot \mathbf{u}$;
   // recurse.
16 return DynamicEffectivenessConstraints($\mathbf{X}, \mathbf{u}, l$)
```

function `AddDynamicConstraints` considers the points just before and just after constraint violation, and adds the equality constraints required to resolve the case. For instance, if a part collides an inner volume of height H, it adds a constraint to keep the colliding part at a distance of H of the part supporting the inner volume. This is a very flexible approach allowing for a variety of design effectiveness constraints.

5.3.5.2 Design Constraints Enforcement

We now describe how to recover a valid point while attempting to preserve the latest changes made by the system. Let us consider a design point $\mathbf{X} + \mathbf{u}$ that violates some design constraints, i.e., $\mathbf{C}(\mathbf{X} + \mathbf{u}) \neq \mathbf{s}$. We seek a correction vector $\mathbf{d}$ such that $\mathbf{C}(\mathbf{X} + \mathbf{u} + \mathbf{d}) = \mathbf{s}$ while preserving the changes applied by the system, that is $d_i = 0$ for all i such that $u_i \neq 0$. Note that we can ignore design effectiveness constraints ($E(\mathbf{X})$) as they are taken care of by the parent algorithm presented in Section 5.3.5.1.

In general, there are many possible solutions as the system is under-constrained. This ambiguity amounts to valid design choices. In addition to preserving the value of the variables being manipulated, it is also desirable to limit the number of variables being adjusted by $\mathbf{d}$, so that variables seemingly not involved in the current user edits remain unchanged. In other words, we seek a *sparse* correction vector $\mathbf{d}$. This concentrates the changes on a few parts in the design, which results in a more predictable interaction.

Our approach is inspired by the works of Bokeloh et al. [BWSK12] and Habbecke and Kobbelt [HK12]. The key idea is to solve for d using only a subset of the variables in $\mathbf{X}$. Let us denote canonical basis vectors $\mathbf{B_k} = (0, ..., 1, ...0)$ with $k-1$ leading zeros, and Λ a set of variable indices that are allowed to change. Then, we can express

Algorithm 6: CorrectForDesignConstraint

```
Input: Linear system of constraints $\mathbf{C}$, s, a valid design point $\mathbf{X}$, a vector of
       changes $\mathbf{u}$
Output: A correction vector $\mathbf{d}$ such that $\mathbf{C}(\mathbf{X}+\mathbf{u}+\mathbf{d}) = \mathbf{s}$ and $d_i = 0$ where
        $u_i \neq 0$
1  $\Lambda \leftarrow \emptyset$;
2  $\mathbf{d} = \mathbf{0}$;
3  $\mathbf{r} = \mathbf{C}(\mathbf{X}+\mathbf{u}+\mathbf{d}) - \mathbf{s}$;
4  while true do
5      $\mathcal{N} \leftarrow$ {variables from non-satisfied constraints ($r_i \neq 0$)};
6      $\mathcal{T} \leftarrow \mathcal{N} \setminus (\Lambda \cup \{i | u_i \neq 0\})$;
7      if $\mathcal{T} = \emptyset$ then
           // Cannot solve: revert.
8          Fail;
9      foreach $v_i \in \mathcal{T}$ do
           // Estimate how variable will move.
           // Variable $v_i$ corresponds to column $c_i$ in $\mathbf{C}$.
10         $m = -\mathbf{c}_i^\intercal \mathbf{r} / \mathbf{c}_i^\intercal \mathbf{c}_i$;
11         $score(v_i) = 0$;
12         if $v_i \in \{l^x, l^y, l^z\}$ and $m$ violates min length on $v_i$ then
               // Preserve min edge length constraint.
13             $score(v_i) = -\infty$;
14         else if $v_i \in \{l^x, l^y, l^z\}$ then
               // Penalise change in length.
15             $score(v_i) = -|m|$;
16         if $score(v_i) > score(b)$ then
17             $b \leftarrow v_i$;
18     $\Lambda = \Lambda \cup \{b\}$;
19     $\mathbf{d}$ = SolvePartialCorrectionVector($\Lambda$,C,s,$\mathbf{X}+\mathbf{u}$);
20     $\mathbf{r} = \mathbf{C}(\mathbf{X}+\mathbf{u}+\mathbf{d}) - \mathbf{s}$;
21     if $||\mathbf{r}|| < \epsilon$ then
22         return $\mathbf{d}$;
```

the correction vector $\mathbf{d}$ as: $\mathbf{d}_\Lambda = \sum_{k\in\Lambda} d_k \mathbf{B_k}$. Hence, the system takes the form:

$$\begin{aligned} &\text{for all } k \in \Lambda \\ &d_k && = \textstyle\sum_i \lambda_i c_{ik} \\ &\textstyle\sum_k d_k \cdot c_{ik} && = s_i - \mathbf{p} \cdot \mathbf{c}_i \end{aligned}$$

The system is often under-determined after removing variables. We solve it by QR decomposition and computing the pseudoinverse of the system matrix. Given a set Λ, solving for $\mathbf{d}_\Lambda$ results in a residual $\varepsilon(\mathbf{d}_\Lambda) = \|\mathbf{C}(\mathbf{X} + \mathbf{d}_\Lambda) - s\|$. We accept solutions when $\|\varepsilon(\mathbf{d}_\Lambda)\| < \epsilon$ with ϵ a small constant (10^{-9} mm in our implementation). We use the Eigen library for solving all linear systems.

Our approach is inspired by Orthogonal Basis Pursuit [CDS98], which greedily selects the next variable based on the residual, until it becomes small enough. We follow a similar greedy strategy using the following criteria to prioritise:

1. Change as few variables as possible, which is minimising $|\Lambda|$.

2. Avoid changes that would violate the minimal part lengths, which is where $l_i^a + d_i < M_i^a$.

3. Avoid changes to the part lengths, and favour changes in part positions instead. Indeed, length variables are generally more important than the positional variables: most furniture designs are fully connected, and hence part lengths are enough to describe the full design up to a translational degree of freedom.

The algorithm for variable selection is given in Algorithm 6. Function `SolvePartialCorrectionVector` implements the projection using a subset of the variables. We initialise Λ with the indices of the variables of newly added parts, so that they are changed first and do not impact the current design. Thus, Λ will never contain indices such that $u_i \neq 0$, and thereby protect the changes made by $\mathbf{u}$.

5.3.5.3 Failure Cases

The system can fail to solve (see Algorithm 6, line 8 and Algorithm 5, line 15). This typically happens if the user makes contradictory changes (e.g., attempting to modify the length of a part having a fixed length constraint), or when the inner/outer volume constraints cascade into a similar contradiction.

In such cases, the user can choose to violate the constraints. The system then provides a visual feedback of the constraints in error. Alternatively, the user may also choose to accept the partial solution proposed by the system. When the system is calling the solver, as is the case for the suggestion system described in Section 5.3.3, the latter option is always used.

5.4 Automatic Design Layout Optimisation[2]

The wastage of a layout depends essentially on two factors. The first factor is the quality of the packing that can be achieved, given a fixed set of design parts. The second factor is the set of parts itself, which can be changed through the design parameters $\mathbf{X}$.

In our approach, we pack the parts using a deterministic docking algorithm that always produces the same result for a same ordering of the design parts. Therefore, a first optimisation variable is the order in which the parts are sent to the docking algorithm. The second optimisation variable is the vector of design parameters $\mathbf{X}$. These two variables have different natures: finding an ordering is a combinatorial problem while the design parameters can be continuously explored.

We therefore proceed in two main steps, first determining a set of good orderings that then serve as starting points for continuously evolving the design, reducing wastage. The overall approach is described in Algorithm 7. The subroutine IMPROVEDESIGN is described in Section 5.4.2 while EXPLOREORDERINGS is described in Section 5.4.3. The process restarts for a number of iterations (we use $G = 3$) to jump out of local minima reached by the continuous design exploration. This results in the shape space exploration illustrated in Figure 5.5. The process returns the K best found layouts and designs and presents them to the user in thumbnails. She can then select her favourite design, and if desired update the constraints and restart the exploration from this point — which simply calls MINWASTAGE again.

[2]The work in this section is done by Jean Hergel (INRIA Nancy - Grand-Est).

Algorithm 7: MINWASTAGE

```
Input: Design function $D$, starting design parameters $\mathbf{X}_s$
Output: Set of best layouts found $\mathcal{L}$
1  $O_s \leftarrow$ identity ordering ; // 1,2,3,...
2  $\mathcal{X} \leftarrow \{(\mathbf{X}_s, O_s)\}$;
3  for $G$ iterations do
4      foreach $(\mathbf{X}, O) \in \mathcal{X}$ do
5          $\mathcal{O} \leftarrow$ EXPLOREORDERINGS($\mathbf{X}, O$);
6          foreach $O \in \mathcal{O}$ do
7              $\mathcal{X} \leftarrow \mathcal{X} \cup \{($IMPROVEDESIGN$(\mathbf{X}, O), O)\}$;
8      $\mathcal{X} \leftarrow$ KEEPBESTS($K, \mathcal{X}$);
9  $\mathcal{L} \leftarrow \emptyset$;
10 foreach $(\mathbf{X}, O) \in \mathcal{X}$ do
11     $\mathcal{L} \leftarrow \mathcal{L} \cup$ DOCKING($D(\mathbf{X}), O$);
12 return ($\mathcal{L}$);
```

Algorithm 8: IMPROVEDESIGN

```
Input: Starting design parameters $\mathbf{X}$ and ordering $O$
Output: Modified design parameters $\mathbf{X}_b$ with reduced wastage
1  $L \leftarrow$ DOCKING(D($\mathbf{X}$),$O$);
2  $\mathbf{X}_b \leftarrow \mathbf{X}, L_b \leftarrow L$ ;
3  $\mathbf{X}_c \leftarrow \mathbf{X}, L_c \leftarrow L$ ;
4  for $N$ iterations do
5      $\mathbf{X}_b, L_b \leftarrow$GROWPARTS($\mathbf{X}_b, L_b$,$\mathbf{X}_c, L_c, O$);
6      $\mathbf{X}_c \leftarrow$SHRINKPARTS($\mathbf{X}_b, L_b$);
7      $L_c \leftarrow$ SLIDE($L_b$,$D(\mathbf{X}_c)$);
       // Check for improvement over current.
8      if $W(L_c) < W(L_b)$ then
9          $\mathbf{X}_b = \mathbf{X}_c, L_b = L_c$;
10 return ($\mathbf{X}_b$);
```

5.4.1 Bitmaps

During optimisation, we regularly call the parameterised design function $D(\mathbf{X})$ to obtain a new set of parts after changing parameters. The layout optimisation represents parts internally as bitmaps: each part contour is rasterised at a resolution τ, typically 0.5 mm per pixel. This enables fast manipulation of the parts within the layout. Each part thus becomes a bitmap having either 1 (inside) or 0 (outside) in each pixel. The size of the bitmap matches the part extents in material space w_i and h_i. Every time the design is refreshed a new set of bitmaps is computed for the parts. The master board is similarly discretised into a regular grid of resolution τ.

5.4.2 Design Optimisation for Wastage Minimisation

The design optimisation improves the design parameters $\mathbf{X}$ to minimise wastage in the layout, keeping the docking ordering fixed. It appears as the subroutine IMPROVEDESIGN in Algorithm 7. The pseudocode for this step is given in Algorithm 8. Our objective is to suggest design changes that reduce wastage, progressively improving the initial layout. The algorithm performs a guided local search by changing the parts – through the design parameters – to reduce wastage.

Prior to considering which parts to modify, we have to answer two questions: First, how to drive the design parameters $\mathbf{X}$ to change only a given part (Section 5.4.2.2). This is achieved by relying on the gradients of the part size with respect to $\mathbf{X}$. Second, we have to decide on how to evolve the layout when parts are changed (Section 5.4.2.3). We rely on a sliding algorithm that avoids jumps in the layout configuration, thus producing only small changes in the wastage function when small changes are applied to the part sizes.

5.4.2.1 Overall Strategy

Our approach changes the size of parts iteratively with two different steps in each iteration: *grow* (line 5) and *shrink* (line 6). These steps progressively modify the design and keep track of the design of smallest wastage encountered so far.

The grow step (Section 5.4.2.4) attempts to enlarge the part lengths so as to reduce wastage. Each part is considered and its size is increased for as long as the growth further reduces wastage. When no further improvement can be obtained, we create further opportunities by shrinking a set of parts (Section 5.4.2.5). However, randomly shrinking parts would be inefficient, as most parts would grow back immediately to their original sizes. Other parts are tightly coupled to many others in the design D, and shrinking these would impact the entire design. Therefore, we analyse the layout to determine which parts have a higher probability to result in wastage reduction.

5.4.2.2 Changing Part Sizes

During design space exploration, the algorithm attempts to vary the part sizes w_i and h_i individually. These dimensions vary as a function of design parameters $\mathbf{X}$. In the remainder, we use $\mathrm{s}(\mathbf{X})$ to designate the vector of all part sizes assembled such that $s_{2i} = w_i$ and $s_{2i+1} = h_i$.

Let us denote λ the change of size desired on s_i. Our objective is to compute a design change Δ such that $s_i(\mathbf{X} + \Delta) = s_i(\mathbf{X}) + \lambda$. We denote the vector of changes as $\Lambda = \mathrm{s}(\mathbf{X} + \Delta) - \mathrm{s}(\mathbf{X})$. In this process, only the size s_i should change while others remain unchanged whenever possible, i.e., $\Lambda_{s_j, j \neq i} = 0$ and $\Lambda_{s_i} = \lambda$.

Parts are not independent in the design and therefore there is no trivial link between $\mathbf{X}$ and $s_i(\mathbf{X})$. We therefore analyse the relationship through the gradients $\frac{\partial s_i(\mathbf{X})}{\partial x_j}$. These are computed by local finite differencing (depending on the design, analytical expressions may be available). Each non-null gradient indicates that parameter x_j influences s_i. Multiple parameters may influence s_i and parameters typically also influence other variables: there exists $k \neq i$ such that $\frac{\partial s_k(\mathbf{X})}{\partial x_j} \neq 0$.

To compute Δ, we formulate the following problem. Let us consider the components of $\Delta = (\delta_0, ..., \delta_{|\mathbf{X}|-1})$. The change in part sizes due to Δ can be approximated in the first order through the gradients as $\mathbf{\Lambda} = \sum_i \delta_i \cdot \frac{\partial \mathrm{s}(\mathbf{X})}{\partial x_i}$. We solve for Δ such that $\Lambda_{s_i} = \lambda$ and $\Lambda_{s_j, j \neq i} = 0$.

If there are less parameters than part sizes, the problem is over-constrained and solved in the least-squares sense, minimising $||\mathbf{\Lambda} - (0, ..., \lambda, ..., 0)||^2$. If there are more parameters than part sizes, the problem is under-constrained and solved in the least-norm sense, minimising $||\Delta||$. We rely on a QR decomposition of the system matrix to solve for both cases, accounting for possible rank deficiencies due to overlapping parameters in $\mathbf{X}$.

We implement this process as a subroutine CHANGEPARTSIZE($\mathbf{X}$,s_i,λ), with

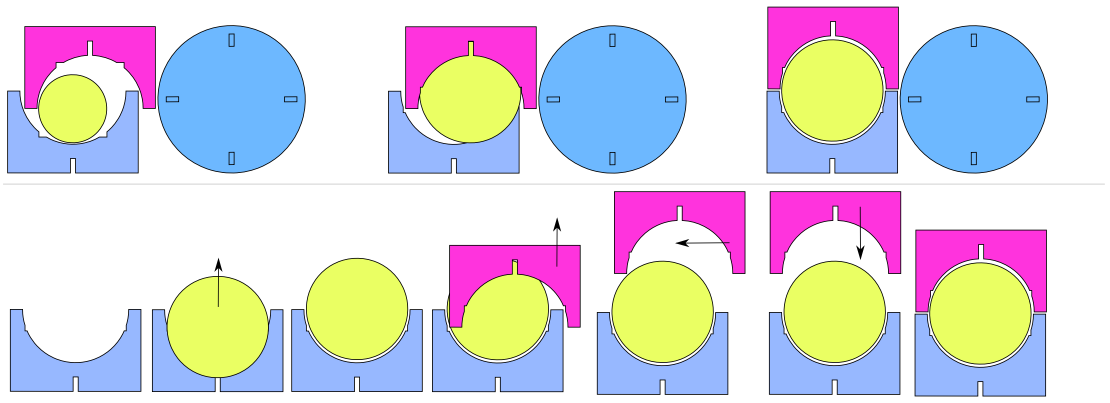


Figure 5.13: Sliding. *Sliding a layout after a change of part sizes. Top: From left to right, initial layout, same after change revealing overlaps, layout after sliding. Bottom: Moves performed on the three first parts during sliding.*

$\mathbf{X}$ the current design parameters, s_i the part size to change and λ the change to apply. It returns the new design parameters $\mathbf{X} + \Delta$. A second subroutine CHANGEPARTSIZES($\mathbf{X}$,Λ) allows to change the size of multiple parts at once.

5.4.2.3 Updating Layouts by Sliding

As the shapes and sizes of the parts change, the layout has to be updated. One option would be to restart the docking process after each change. However, for a small change the docking process can produce large discontinuities in the wastage function. This makes a local search difficult. Instead, we propose to rely on a sliding operation that attempts to continuously update the position of the parts after each change. Note that performing such an update while optimising for a given objective (i.e. wastage) is a very challenging combinatorial problem, as each part can move in four directions (left-/right/top/bottom) and multiple cascading overlaps have to be resolved. We propose a heuristic approach that works well for small changes in the part shapes.

The algorithm is based on the following principle. After changing the part shapes, we reintroduce them in an empty layout in order of docking. However, each time a part is reintroduced it may now have empty space to its left/bottom or it may overlap with previously placed parts. Both cases can be resolved by a single horizontal or vertical move. However, a single move is generally not desirable as empty space may remain along the other direction. We therefore perform a limited sequence of horizontal/vertical moves. At each iteration we select between vertical or horizontal by favouring moves that result in the smallest layout bounding box. In case of a tie, we favour moves to the left/bottom versus displacements to the top/right. This is illustrated in Figure 5.13.

The pseudocode is given in Algorithm 9. In the algorithm, we denote by L the layout and denote by $L \lhd_{pos} p_i$ the layout obtained when adding part p_i at position pos in the master board grid of L. $A(.)$ measures the area, $box(L)$ is the bounding rectangle

Algorithm 9: SLIDE

```
Input: current layout $C = (u_0, v_0, ...)$ and set of changed parts $parts$
Output: updated layout $L$
$L \leftarrow \emptyset$
foreach part $p_i \in parts$ in docking order do
    for $N$ iterations do
        $\Delta_x \leftarrow -smallestLeftFreeInterval(L, p_i)$;
        if $\Delta_x = \emptyset$ then
            $\Delta_x \leftarrow smallestRightDecollision(L, p_i)$;
        $pos_x \leftarrow (u_i + \Delta_x, v_i)$ ;
        $\Delta_y \leftarrow -smallestBottomFreeInterval(L, p_i)$ ;
        if $\Delta_y = \emptyset$ then
            $\Delta_y \leftarrow smallestTopDecollision(L, p_i)$ ;
        $pos_y \leftarrow (u_i, v_i + \Delta_y)$ ;
        if $pos_x = \emptyset$ and $pos_y = \emptyset$ then
            ; // cannot fit in the master board.
            return $\emptyset$ ; // $W(\emptyset) = 1$
        if $pos_x = pos$ and $pos_y = pos$ then
            break;
        if $A(box(L \lhd_{pos_x} p_i) < A(box(L \lhd_{pos_y} p_i))$ then
            $(u_i, v_i) \leftarrow pos_x$
        else if $A(box(L \lhd_{pos_x} p_i) > A(box(L \lhd_{pos_y} p_i)$ then
            $(u_i, v_i) \leftarrow pos_y$
        else
            if $\Delta_x < \Delta_y$ and $|\Delta_x| > 0$ then
                $(u_i, v_i) \leftarrow pos_x$
            else
                $(u_i, v_i) \leftarrow pos_y$
    $L \leftarrow L \lhd_{(u_i, v_i)} p_i$
return $(L)$;
```

of the layout. The algorithm iterates over all parts in docking order (line 2). It then performs a fixed number of sliding operations on each part (line 3) – we use $N = 4$ in our implementation. Lines 4-7 compute a horizontal move, favouring moves to the left that collapse newly created empty spaces. Lines 8-11 similarly compute a vertical move. Lines 16-24 decide whether to select a horizontal move pos_x or vertical move pos_y.

The process may fail if parts can no longer fit in the master board. This can happen either because there is not enough remaining area, or because sliding cascades in large moves that prevent further insertion of parts. In such cases we return an empty layout which by convention has a wastage of 1 (worst possible), line 13.

5.4.2.4 Grow Step

The grow step is described in Algorithm 10. The algorithm iterates over all parts in random order (line 4) and progressively increases the size of a part in a loop (line 7). Note that the first iteration of the loop determines the starting wastage for growing this part (lines 5 and 12-13). The process continues until the growth results in an increased wastage (line 15).

After each change of parameters the design parts are recomputed (line 9, $D(\mathbf{X_e})$) and sliding is called to adapt the current layout to the change. The result is checked. If wastage decreases the process continues (line 13). If not, we first attempt to dock the parts again (line 11). This can help continuing the growth in cases where sliding fails to resolve overlaps by continuous changes. If wastage still does not improve, we stop the growth of this part size (line 15).

5.4.2.5 Shrink Step

The goal of the shrink step is to create further opportunities for design changes when no parts can further grow. The typical situation is that a subset of parts are forming *locking chains* between respectively the left/right and top/bottom borders. The parts belonging to these chains prevent any further growth. We therefore detect locking chains and select the parts to shrink among these. This often results in a change of aspect ratio of the master board, and new opportunities for other parts to grow.

The overall approach is described in Algorithm 11. It first determines which parts to shrink by calling SELECTPARTSIZESTOSHRINK and then computes a change of parameters using the approach described in Section 5.4.2.2.

The core component is the SELECTPARTSIZESTOSHRINK subroutine, described in Algorithm 12. The selection starts by gathering all contacts between parts in the layout – this is done efficiently in the discretised layout grid. We first draw the part images into the grid and then check pairs of neighbours belonging to different parts. This produces the set of left/right and top/bottom contacts between part sizes (the involved part size is deduced from the part orientation and the considered axis). The contacts are oriented from right to left (respectively top to bottom). We similarly detect which parts touch the borders. The contact detection is implemented in the GATHERCONTACTSALONGAXIS subroutine.

Once the contacts are obtained, we start from the right (respectively top) border and form locking chains. Starting from the border, we produce the set of chains iteratively. Each chain c is a sequence $(left, s_{first}, ..., s_{last})$. At each iteration, the chain spawns new chains for each contact pair (s_{last}, s_{next}) obtained by augmenting c as $(left, s_{first}, ..., s_{last}, s_{next})$. Potential cycles are easily detected as repetition of a same part in the chain and are ignored. The locking chain computation is implemented in the FORMCONTACTCHAINS subroutine.

We next randomly select part sizes to shrink until all locking chains are removed. The selection probability of each part is designed to avoid too large a jump in the design

Algorithm 10: GROWPARTS

```
Input: Best design parameters $\mathbf{X}_b$ and layout $L_b$ so far, current design
       parameters $\mathbf{X}_c$ and current layout $L_c$ being explored, ordering $O$.
Output: New best design and packing.
1  $improvement \leftarrow true$;
2  while $improvement$ do
3      $improvement \leftarrow false$;
4      foreach part size $s_i$ in random order do
5          $W_e \leftarrow 1$ ; // max wastage.
6          $\mathbf{X}_e \leftarrow \mathbf{X}_c$, $L_e \leftarrow L_c$ ;
           // Grow at first time and then continue as long as it
              improves.
7          while true do
8              $\mathbf{X}_e \leftarrow$ CHANGEPARTSIZE($\mathbf{X_e}$,$s_i$,1) ; // +1 pixel.
9              $L_e \leftarrow$ SLIDE($L_e$,$D(\mathbf{X_e})$);
10             if $W(L_e) > W_e$ then
11                 $L_e \leftarrow$ DOCKING($D(X_e)$,O);
12             if $W(L_e) < W_e$ then
13                 $W_e = W(L_e)$;
14             else
15                 break;
           // Check for improvement over current.
16         if $W_e < W(L_c)$ then
17             $\mathbf{X}_c = \mathbf{X}_e$, $L_c = L_e$;
18             $improvement \leftarrow true$;
   // Check for improvement over global best.
19 if $W(L_c) < W(L_b)$ then
20     $\mathbf{X}_b = \mathbf{X}_c$, $L_b = L_c$;
21 return ($\mathbf{X}_b$, $L_b$);
```

Algorithm 11: SHRINKPARTS

```
Input: Best design parameters $\mathbf{X}_b$ and layout $L_b$ so far.
Output: Shrunk design parameters.
1  $\mathbf{X}_s \leftarrow \mathbf{X}$;
2  $\mathcal{S} \leftarrow$ SELECTPARTSIZESTOSHRINK($L_b$);
3  $\Lambda \leftarrow (0, ..., 0)$;
4  foreach $s_i \in \mathcal{S}$ do
5      $\Lambda_i \leftarrow -1$ ; // -1 pixel.
6  $\mathbf{X}_s \leftarrow$ CHANGEPARTSIZES($\mathbf{X_s}$,$\Lambda$) ; // -1 pixel.
7  return ($\mathbf{X}_s$);
```

Algorithm 12: SELECTPARTSIZESTOSHRINK

```
Input: A layout $L$.
Output: Set of part sizes to shrink.
1  $\mathcal{K} \leftarrow \emptyset$;
2  foreach axis $a \in \{X, Y\}$ do
3      $\mathcal{C} \leftarrow$ GATHERCONTACTSALONGAXIS($a$) ;
4      $\mathcal{K} \leftarrow \mathcal{K} \cup$ FORMCONTACTCHAINS($\mathcal{C}$) ;
5  $\mathcal{S} \leftarrow \emptyset$;
6  while $\mathcal{K} \neq \emptyset$ do
7      $s_i \leftarrow$ DRAWPARTSIZEWITHPROBABILITY($\mathcal{K}$);
8      $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}$;
9      $\mathcal{K} \leftarrow \mathcal{K} \setminus$ KILLEDCHAINS($\mathcal{K}$,$s_i$);
10 return $\mathcal{S}$;
```

space. To achieve this, we consider two factors. First, we compute the number of occurrences of each part in the locking chains, $occ(p_i)$. A part with many occurrences is a good candidate as shrinking it will resolve multiple locking chains at once. Second, we seek to avoid shrinking part sizes that are tightly coupled with others in the design D. We compute the dependence of a part size by counting the number of non-zero entries in the $\mathbf{\Lambda}$ vector computed internally by CHANGEPARTSIZE($\mathbf{X_e}$,s_i,-1).

We select part sizes with the following random process. First, we select a number of occurrences o with probability $P(o) = \frac{\sum_{p_i, occ(pi)=o} occ(o)}{\sum_{p_i} occ(p_i)}$. Then, among the parts such that $occ(p_i) = o$ we select a part size s_i with probability $P(s_i | occ(s_i) = o) = 1 - \frac{dep(s_i)}{\sum_{p_i, occ(p_i)=o} dep(p_i)}$. This process is implemented by the DRAWPARTSIZEWITHPROBABILITY subroutine.

After each part size selection, we update the set of locking chain by removing all chains where the part size appears.

5.4.3 Exploring Orderings

The subroutine EXPLOREORDERINGS in Algorithm 7 performs a stochastic search of orderings resulting in low wastage layouts. The process starts from a random order and iteratively considers possible improvements by swapping two parts. At each iteration, we perform a swap and recompute a layout using the docking algorithm. If wastage is reduced, the swap is accepted. Otherwise, it is rejected. We apply the process for a number of iterations and keep the best ordering found as the starting point. We use $|D(\mathbf{X})|^2$ iterations, where $|D(\mathbf{X})|$ is the number of parts. For each ordering, we use a fast docking algorithm to compute a layout with low wastage.

5.4.3.1 Docking Algorithm

The docking algorithm places each part in order by 'dropping' the next part on the current layout either from the right, or from the top. It locally searches for the best

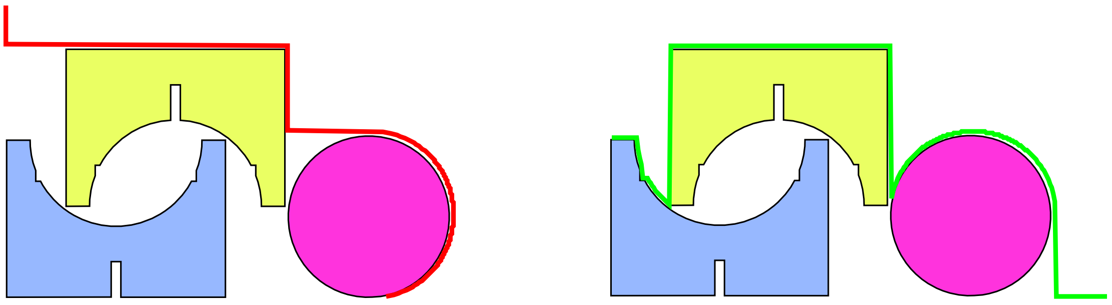


Figure 5.14: Height-fields. *Layout height-fields used to position the next part. Left: Height-field for dropping parts from the right (red curve). Right: Height-field for dropping parts from above (green curve). These height-fields are maintained every time a new part is added to the layout, and used for fast computation of the docking positions. Similar height-fields are pre-computed for the left/bottom of the parts.*

placement of each part, according to a criterion that minimises wastage. The result is a layout L including all parts.

Given the layout so far, our algorithm searches for the best orientation and best position for the next part. We denote by L_{i-1} the layout obtained for the $i-1$ first parts, and by $L_i \leftarrow L_{i-1} \triangleleft_{pos} p_i$ the layout obtained by adding the next part at position pos. The docking position pos is computed from a drop location (s, x, o), with $s \in \{top, right\}$, x a position along the corresponding axis and $o \in \{0, \pi/2, \pi, -\pi/2\}$ an orientation.

The pseudocode for the docking algorithm is given in Algorithm 13. The drop locations are ranked according to a docking criterion that we denote $D(L_{i-1}, p_i, pos)$, explained next. The docking positions are computed from the drop locations by the `ComputeDockingPosition` subroutine. It is efficiently implemented by maintaining the right/top height-fields of the current layout as illustrated in Figure 5.14. Whenever evaluating a drop location, we use the height-fields to quickly compute the docking positions that bring the part in close contact with the current layout.

5.4.3.2 Docking Criterion

The docking criterion considers wastage as the primary objective, where wastage is defined by the ratio of occupied area divided by the bounding rectangle area of the layout. We denote $W(L_i)$ the wastage of a layout including up to part i. It is obtained as $W(L_i) = \frac{\sum_{k=0}^{i} A(p_k)}{A(box(L_i))}$ where $A(.)$ measures area and $box(L)$ is the bounding rectangle of the layout. W is therefore the ratio between the area of the parts and the area of the bounding rectangle.

However, as the algorithm heuristically docks parts in sequence, it cannot foresee that some spaces will be definitely enclosed. In particular, for newly inserted *concave* parts, there are often multiple orientations of the part resulting in the same wastage: if the concavity remains empty, there is no preferred choice. However, some choices are

Algorithm 13: DOCKING

```
Input: Set of parts $P$, order $O$, master board dimensions $W \times H$
Output: A layout $L$
1  foreach part $p_i \in P$ following order in $O$ do
2      $best \leftarrow \emptyset$ ;
3      $bestscore \leftarrow 1$ ;
4      foreach drop location $(s, x, o)$ do
5          $pos \leftarrow$ ComputeDockingPosition$(p_i, (s, x, o))$ ;
6          $score \leftarrow D(L_{i-1}, p_i, pos)$ ;
7          if $score < bestscore$ then
8              $best \leftarrow pos$ ;
9              $bestscore \leftarrow score$ ;
10     $L_i \leftarrow L_{i-1} \lhd_{pos} p_i$ ;
11 return $L_n$;
```

indeed better than others. If the concavity faces an already placed object, then further docking *within* the concavity will never be possible. This is illustrated in Figure 5.15 left.

We therefore propose a second criterion that discourages these bad choices. The idea is to estimate the space that will be definitely enclosed when a part is added to the current layout. This is done efficiently by considering the enclosed space between the height-field of the current layout and the height-field of the added part, along both horizontal and vertical directions.

Let $H^r(L)$ (respectively H^t) be the right (respectively top) height-field of layout L and $A(H^r(L))$ the area *below* it. The enclosed area is then defined as:

$$E(L_{i-1}, p_i, pos) =$$
$$\sum_{s \in \{r,t\}} \max\left(0, A(H^s(L_{i-1} \lhd_{pos} p_i)) - A(H^s(L_{i-1})) - A(p_i)\right)$$

with $A(p_i)$ the area of part p_i. Note the $\max$ that clamps negative values: this is due to cases where the part nests in a concavity below the height-field of the other direction.

The enclosed space is used as a tie-breaker when docking positions produce the same wastage values; therefore $D(L_{i-1}, p_i, pos)$ returns the vector $(W(L_{i-1} \lhd_{pos} p_i), E(L_{i-1}, p_i, pos))$. The effect of the enclosed area criterion is shown in Figure 5.15.

5.5 Results

We used our system for various design exploration tasks. As the complexity of the designs grows beyond 4-6 parts, the utility of the system quickly becomes apparent. It is very difficult for novices, and even for professional designers, to predict the effect of

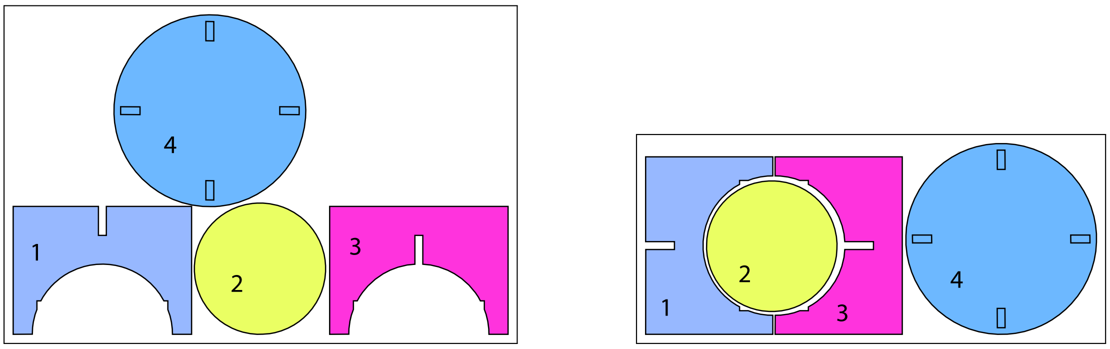


Figure 5.15: Docking. *Two layouts obtained with the same docking order. Left: Without taking enclosed area into account, the first part is placed with the concavity against the bottom packing border. This prevents the second part to nest within and cascades into a series of poor placements. Right: Taking into account enclosed areas results in a placement of the first part that allows nesting of the second part and produces a layout with lower wastage.*

changing a part size when there are many parts which have constraints on each other. A size change in one part may result in changes in multiple parts in the layout, which will lead to a change in the material usage of the layout. Therefore, it is very challenging and time-consuming to change the design while maintaining low material wastage without computational supports. Technically, the design constraints (see Figure 5.11), by coupling different object parts, make the optimisation challenging by preventing independent adaptation of part sizes. By offloading material usage considerations to the system, the user can focus on the design. Note that even when changes to the design are visually subtle, material utilisation often increases significantly.

5.5.1 Design Examples

We used our system to design and fabricate a range of examples comprising rectangular and/or curved parts. We fabricated full-scale and miniature models of designed furniture. Models were made from MDF of 3 mm thickness and MDF of 30 mm thickness. MDF refers to medium-density fibreboard. It is made from wood and is often used for laser cutting. The designs are easy to manufacture in batches since they typically fit master boards completely after design layout optimisation: there is no need to attempt to reuse leftover pieces of wood, and switching boards requires little clean-up.

Our system directly outputs the cutting plan for the laser cutter or CNC milling machine from the design layout. It also adds connectors for parts sharing an edge, if needed, for assembly. These are conveniently detected since parts exactly overlap on edges in the 3D design. The connectors are either *finger joints*, which are both strong after gluing and easy to assemble; *cross connectors* for interleaved parts, or dowel-jointed for thicker materials (20 mm and 30 mm thickness).

Figures 5.16 and 5.17 show various results. Also, Figure 5.19 shows a shelf in use, which was designed using our system. Please note that designs with curved parts

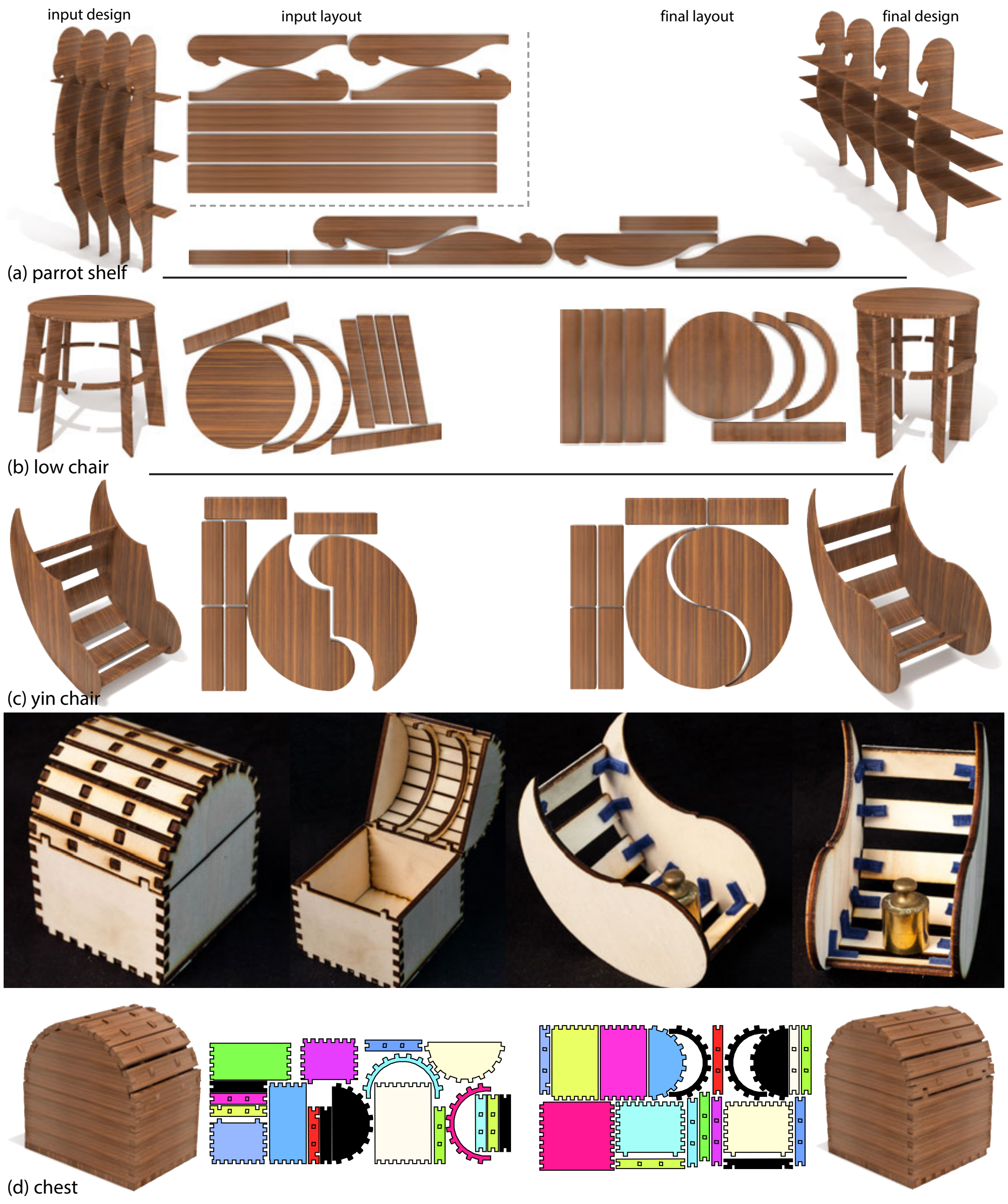


Figure 5.16: Curved designs. *Designs created using our system. Each design is shown with initial shape, starting layout, optimised layout and final design. (a) is a constrained model while (b)-(d) are parametric models. The fourth row shows the fabricated models of the chest (left) and yin chair (right).*

(Figures 5.1 and 5.16) are generated by the interactive method (Section 5.3) while those with rectangular parts (Figures 5.17 and 5.19) are by the automatic method (Section 5.4). Table 5.1 gives an overview of the complexity of each model, and the gains obtained by the layout optimiser. The system performs at interactive rates on a laptop taking from a few seconds to 3-4 minutes for the larger examples. Note that the speed depends on how many exploration threads are pursued.

Figures 5.1 and 5.16 show results for objects with curved parts. Figure 5.6 shows some intermediate shapes as the design evolves for the coffee table (Figure 5.1) and the low chair (Figures 5.16 top) examples. Figure 5.18 shows alternative designs discovered by the algorithm for the parrot shelf. While they have slightly lower material usage, they offer interesting variations that the user might prefer.

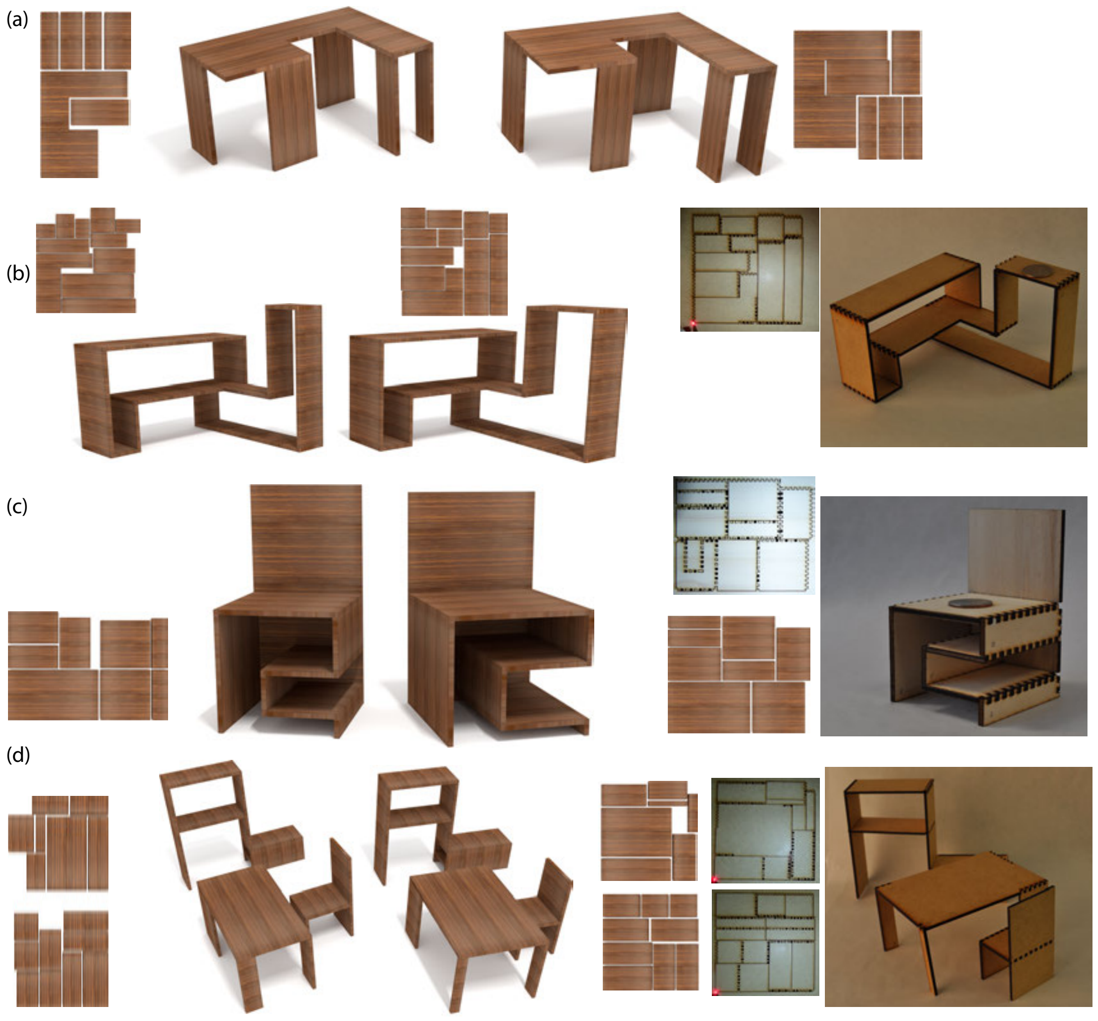


Figure 5.17: Fabricated examples. *Various material-driven design and fabrication examples. In each row, we show initial design (with material space layout inset), optimised design result (with material space layout inset), along with final cut-out assembled model. In (d), two master boards are used for the large design. Note that the design changes are often subtle, but still lead to significant improvement in material usage.*

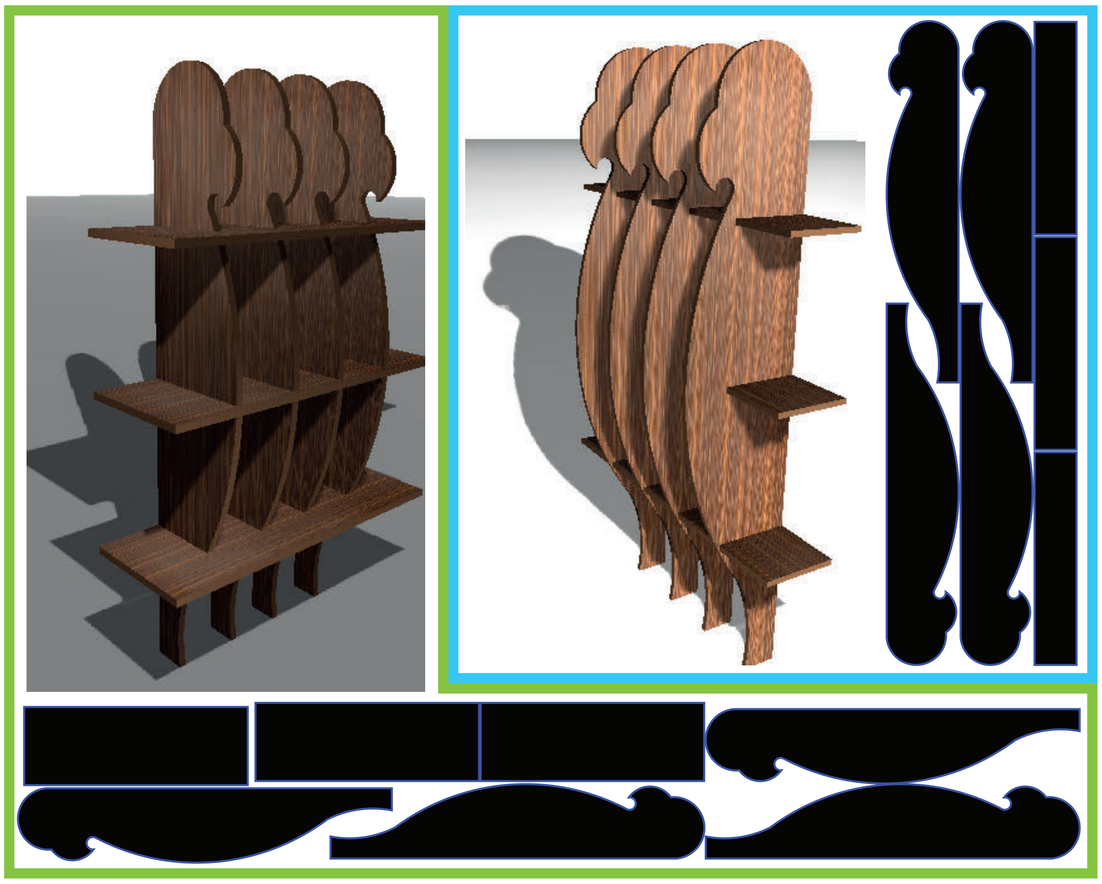
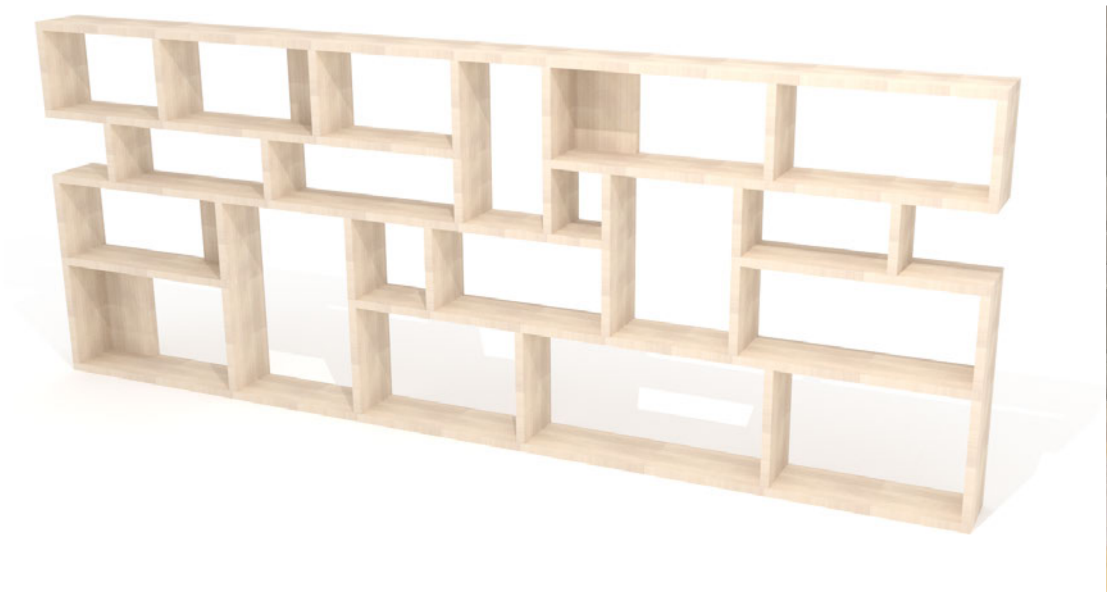


Figure 5.18: Multiple designs. *Two different design suggestions (green has ratio 0.86, blue has ratio 0.85) for the parrot shelf. Original design with another design suggestion is shown in Figure 5.16.*

Table 5.1: Statistics for cut designs. *Statistics for cut design showing the number of parts, number of constraints, material usage ratio before and after the design suggestions/optimisation.*

	#parts	#constraints	ratio before	ratio after
Figure 5.1	4	21	0.78	0.89
Figure 5.16a	7	33	0.66	0.92
Figure 5.16b	9	N/A	0.66	0.80
Figure 5.16c	8	N/A	0.76	0.83
Figure 5.16d	16	N/A	0.79	0.86
Figure 5.17a	6	22	0.85	0.96
Figure 5.17b	11	41	0.85	0.97
Figure 5.17c	8	13	0.74	0.97
Figure 5.17d	16	29	0.89	0.98

The example in Figure 5.1 was fabricated using a CNC milling machine. The optimised design achieved nearly 90% material usage, although one can achieve null wastage by deciding to pick a rectangular top (by editing the shape in the system) – a decision that can be made after layout optimisation as this opportunity is revealed. An



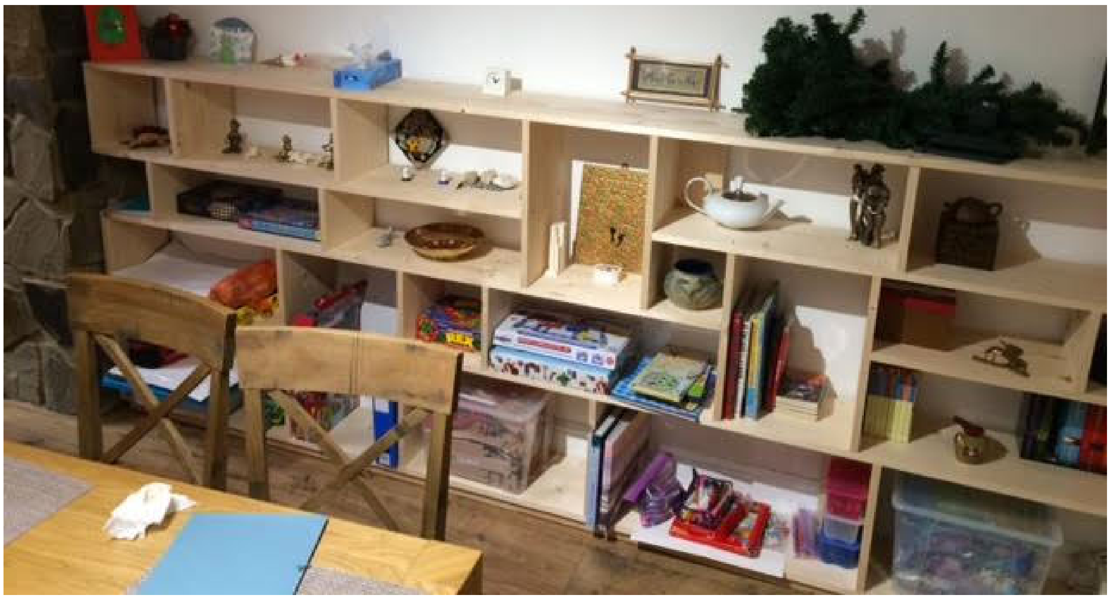


Figure 5.19: Fabricated furniture in use. *Optimised (top) and fabricated (bottom) shelf. Note that although the improvement in this case was* 10%+*, this is an easy example as large number of parts provide extra design freedom for the algorithm to explore.*

allowable range was specified for the height and the bases were marked symmetric as input design constraints. In the case of the parrot shelf (Figure 5.16a), the user indicated minimum and maximum range for the horizontal shelves along with desired range for the shelf heights.

As described, parametric designs are easily supported and optimised for in our framework. Figures 5.16b-d show three such examples. In each case, additional constraints were provided to keep the objects within a given volume. The parts of the objects are all tightly coupled making these challenging examples to optimise for.

Figure 5.17a shows a L-shaped worktable. The user specified a target height for

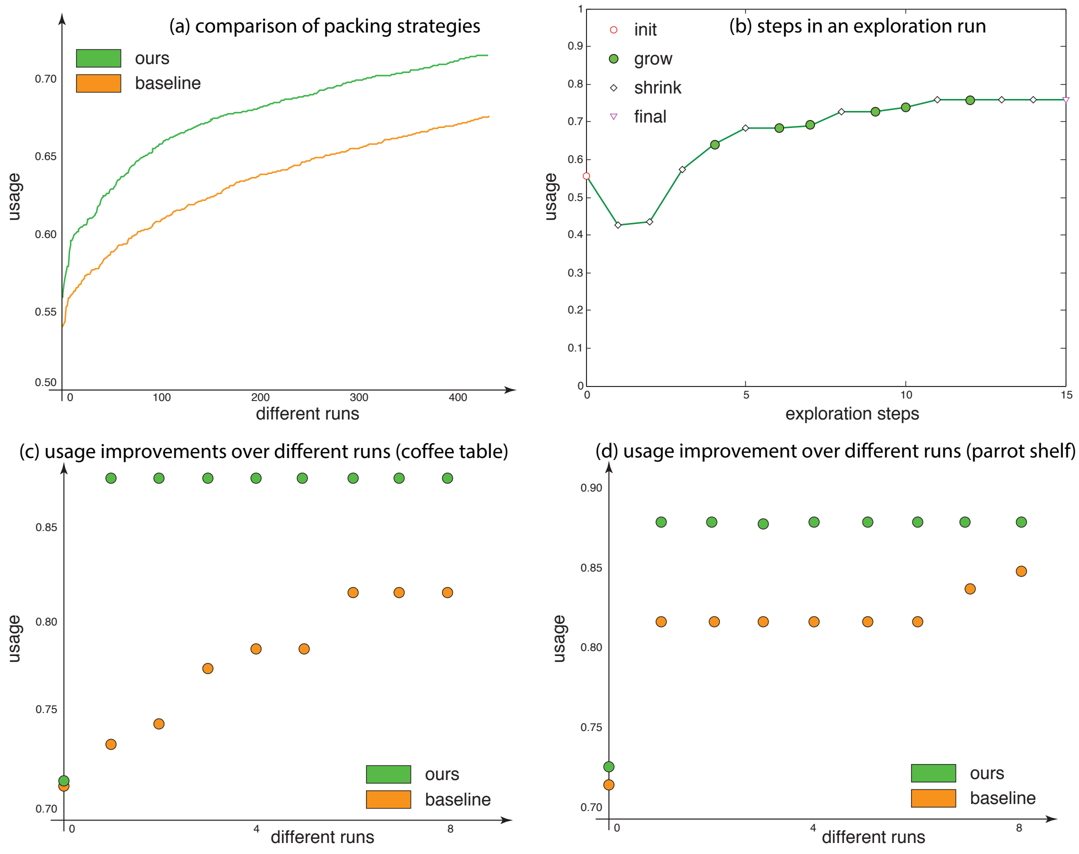


Figure 5.20: Comparison. *Comparison of our algorithm against baseline alternatives. Higher is better. Please refer to the text for details.*

the design and a maximum work volume. Note that the legs of the table were also constrained not to change more than 25% of original dimensions to prevent unwanted design changes. Figure 5.17b shows a coupled shelf and table design where the height of shelves and tabletop were similarly constrained. Figure 5.17c shows a stylised chair, where both the chair seat height and chair width were constrained not to change beyond a margin. Figure 5.17d shows multiple designs covering 2 master boards. The second master board is used as an overflow when docking can no longer fit a part in the first. The layouts are slid independently.

5.5.2 Comparison

We now evaluate the relative importance of the key algorithm steps. Figure 5.20a shows the importance of the docking criteria introduced in Section 5.4.3. We ran 500 random runs of our proposed packing algorithm with ('ours') and without ('baseline') the docking criteria on the coffee table example. We sort the runs based on resultant usage (no shape optimisation is performed here) and plot the two conditions. The docking criteria consistently resulted in 10-15% better usage.

Figure 5.20b shows usage improvement over one exploration run on the coffee table sequence. The legend explains which step (grow, shrink, etc.) is being performed.

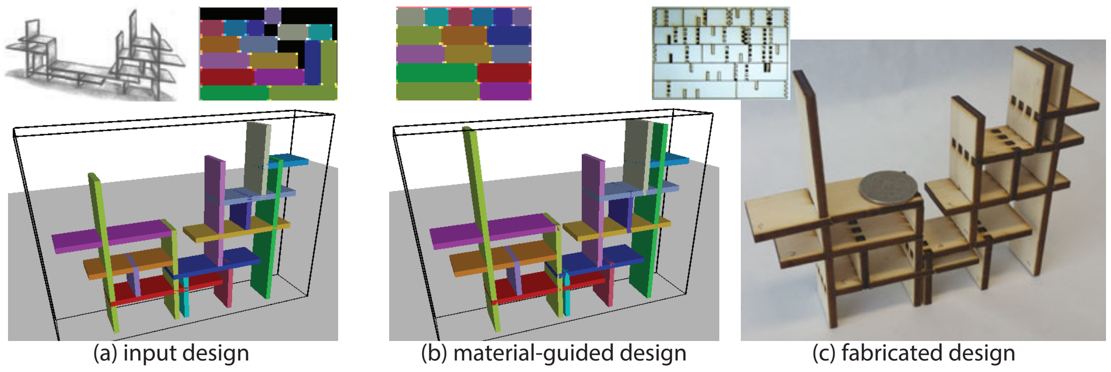


Figure 5.21: Example by an art student. *From the concept sketch and initial 3D model created out of the sketch (a), the art student generated a design with less material usage (b), which was fabricated later.*

While this is the result from a single thread, many similar threads are simultaneously explored. The few best results are then presented to the user as suggestions.

Figures 5.20c-d compare the importance of analysing the material space layout to decide which part to change and how. As a baseline, we selected parts at random and perform either a grow or shrink sequence with equal probability. Note that our method consistently outperforms the alternative approach and we only show the usage improvements up to 8 runs because the usages for our method and baseline do not change much with more runs.

5.5.3 Design Sessions

We asked second year art students (6 subjects) from a design college to try our system. Figures 5.17b-d show a selection of their designs. These particular students had performed a very similar task as part of their first year assignment – ‘design furniture of your choice making best use of the provided piece of MDF board.’ Hence, they were very aware of the implicit link between design and material usage. Previously, they had used commercial 3D modelling tools (Rhinoceros, SolidWorks, SketchUp) for designing and mainly Illustrator for manually laying out the designs. They recalled the frustration of having to switch between the different 2D-3D design representations and tools. First, the students sketched design concepts before using our system. Then, they used the exploration interface on their designs to reduce wastage. Note that visually the initial sketch and final design can look similar despite the increase in material utilisation, which is desirable because it means that the original design, i.e., design intent, is preserved (see Figure 5.21).

Overall, the feedback from the students was positive. They appreciated being able to easily move between 2D and 3D in one unified tool, and not having to explicitly worry about material utilisation. They appreciated the design suggestions as well, instead of previous attempts using trial-and-error iterations between various softwares to reduce material wastage while trying to retain the original design intent.

5.5.4 Limitations

Currently, the algorithm can only make topological changes only for parametric models. It will be an interesting future direction to allow topological changes for constrained models. Also, our docking approach cannot nest parts into holes in other parts because parts can only slide but cannot jump over other parts. An advanced algorithm would be required to achieve this task. A more material-induced restriction arises when the starting layout does not leave much space to optimise over. This effectively means that the degree of freedom for the design is low. Adding more parts does reduce this problem as it provides additional freedom to the system. However, beyond 25-30 parts, the exploration of the shape space becomes slow because there are too many paths to explore. One option is to limit the exploration to only a subset of parts at a time, but then again, very desirable design configurations may be missed.

5.6 Conclusions and Future Work

In this chapter, we investigated how design constraints and material usage can be linked together towards form finding. Our system dynamically discovers and adapts to constraints arising due to current material usage, and computationally generates design variations to reduce material wastage. By dynamically analysing 2D material space layouts, we determine which and how to modify object parts, while using design constraints to determine how the proposed changes can be realised. This interplay results in a tight coupling between 3D design and 2D material usage and reveals information that usually remains largely invisible to the designers in the currently available tools, and hence difficult to account for. We used our system to generate a variety of shapes and demonstrated wastage reduction by 10% to 15%.

Currently, the stability of the fabricated furniture nor the durability of the joints are not accounted for in the system. It will make the system more useful for the fabrication of full-scale furniture if these are integrated into the system as design constraints. A support for other materials that can bend such as plastics will help the user design and fabricate other types of interesting designs.

Chapter 6

Conclusions

We have been investigating the main challenges in computational fabrication and how to address them in this thesis. The design process for making products or fabricating customised objects is typically a set of iterations of analysis, ideation, prototyping and evaluation until the design converges to an optimal one, which solves the problems we would like to address with the objects. Though prototyping has become easier, cheaper and faster thanks to the recent developments in the technologies such as 3D printing and laser cutting, a main difficulty remains: *creating a design*. Creating a design which satisfies high-level goals, e.g., functional relationships between parts in the object or minimal material wastage, is not trivial even for professional designers. This is because low-level geometric features such as vertices, edges or faces, should be manipulated for achieving those high-level goals in the currently available modelling tools. The lack of the user guidance with respect to the relationship between function/material usage and the design in the current tools results in the trial-and-error approach in form finding, which requires much time and resources.

In this thesis, we try to address this problem by creating computational tools that the function and material usage are well integrated into the design process loop so that the numerous iterations can be reduced. This will help the user explore the design space more effectively, which will result in better designs. We verify this hypothesis by creating computational tools for this purpose. We developed an interactive tool that allows the user to design and fabricate mechanical prototypes by specifying high-level functional relationship between parts (Chapter 4). Various designs were created using the system and fabricated using 3D printers. Our system is able to reduce the effort of manipulating part dimensions and joint parameters laboriously, which leads to faster iterations of design process. We also introduced a tool which suggests designs and layouts that achieve minimal material wastage (Chapter 5). Leftover materials after fabrication using subtractive manufacturing methods such as laser cutting or CNC milling, have a negative effect economically and environmentally alike. Our system effectively optimises layouts and part shapes of a design while respecting design intent, specified as design constraints, so that the amount of remaining materials after cutting out necessary pieces can be minimised. Provided that the relationship between a design and its effectiveness of material usage is hard to assess before the actual fabrication of the design, our tool helps the user create a design with minimal material waste without taking the trial-and-error approach, i.e., a number of repetitions of designing and fabricating.

Throughout the thesis, we sometimes made assumptions to simplify the very challenging problems. Our systems cannot deal with some useful cases occurring in the real world. However, being the first effort trying to address the problems in these new approaches, we hope that our research can attract more efforts into this domain.

6.1 Contributions

The contributions of this thesis are:

- Clearly defined sets of constraints regarding function and material usage in the design and fabrication process.
- Novel algorithms for generating designs subject to the constraints.
- Computational tools which guide the users to design objects that satisfy the constraints.

6.2 Limitations and Future Work

As the design paradigms guided by function and material usage proposed in this thesis are rather new, there is much room for improvement and many new research directions.

Regarding the system guiding the user by function introduced in Chapter 4, the system supports only cuboid shapes though this abstract representation is often sufficient when the user would like to explore the design space focusing on the mechanisms of the moving parts. Supports for arbitrary shapes, however, will help the user understand the relationship between the function and design better because now the shape of the prototype is more accurate and closer to the final product. Also, the types of functions and types of corresponding joints supported in the system can be extended to include more complex mechanisms. This will greatly increase the usefulness of the system.

For the system guiding the user by material usage proposed in Chapter 5, we do not consider the stability of the fabricated furniture nor the durability of the joints, which is a critical factor for the furniture to function in the real world use. This could be integrated as dynamic constraints following previous work on furniture design [UIM12], structural reinforcement [SVB$^+$12] and shape balancing [PWLSH13]. Another important future direction is to generalise the framework to handle other types of laser cut materials than wood, e.g., plastic plates that can be easily cut and more interestingly bend to have free-form shapes. Note that the packing problem will still be in 2D for such developable pieces. This can help produce interesting free-form shapes, while still making efficient use of materials.

For the general future research directions regarding computational fabrication, data-driven modelling methods could facilitate designing new shapes as the collections of fabricable models, either manually designed or 3D scanned from real objects, become larger. With this approach, a user of the system will be able to explore the design space of fabricable shapes more easily. Also, other design modalities, such as using natural language, are worth exploring. For instance, a system could generate a fabricable design of bookcase from a verbal description from the user. Furthermore, as the capability of 3D printers becomes greater, we can envision more interesting applications. If 3D printers are able to fabricate mechanical and/or electronic devices, for instance, cars or robots, as already assembled entities, i.e., all the mechanisms and circuits are printed with the appearance, or geometry, of the device, we can think of a system that generates optimal designs that satisfy user-specified constraints. For example, the user could ask the system to produce robot designs that can run as fast as possible. It will be interesting to see how the system optimise the layouts of the circuits, locations/shapes of joints and the appearance of a robot altogether to achieve the goal. Moreover, as some buildings can be 3D printed already, we can imagine 3D printing a whole building including lifts, water pipes, electricity lines, etc. Then, we could develop a system that optimises the blueprint of the building with respect to the user's constraints, e.g., the purpose of the building (office, accommodation) or capacity (the number of people who use the building per day). Similarly, we could expand this to building a whole city, or a colony on Mars.

Appendix A

Publications

The work in this thesis appears in the following publications:

- Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating Works-like Prototypes of Mechanical Objects. *ACM Trans. Graph.*, 33(6):217:1–217:9, November 2014.
- Bongjin Koo, Jean Hergel, Sylvain Lefebvre, and Niloy J. Mitra. Towards Zero-Waste Furniture Design. *This work has been submitted to a journal.*

Bibliography

[BBJP12] Moritz Bächer, Bernd Bickel, Doug L. James, and Hanspeter Pfister. Fabricating Articulated Characters from Skinned Meshes. *ACM Trans. Graph.*, 31(4):47:1–47:9, July 2012. 17, 23

[BBS08] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. ILoveSketch: As-natural-as-possible Sketching System for Creating 3d Curve Models. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 151–160, New York, NY, USA, 2008. ACM. 19

[BFM03] BFM Ltd. Wood waste recycling in furniture manufacturing – a good practice guide. Technical report, British Furniture Manufacturers, 2003. 5, 52

[BR98] Beat Brüderlin and Dieter Roller, editors. *Geometric Constraint Solving and Applications*, volume VIII. Springer, 1998. 21

[BWSK12] Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An Algebraic Model for Parameterized Shape Editing. *ACM Trans. Graph.*, 31(4):78:1–78:10, July 2012. 21, 70

[CCA$^+$12] Jacques Calì, Dan A. Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 3d-printing of Non-assembly, Articulated Models. *ACM Trans. Graph.*, 31(6):130:1–130:8, November 2012. 17, 23

[CDS98] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, January 1998. 71

[Cha83] Bernard Chazelle. The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Trans. Comput.*, 32(8):697–707, August 1983. 60

[CLM+13] Duygu Ceylan, Wilmot Li, Niloy J. Mitra, Maneesh Agrawala, and Mark Pauly. Designing and Fabricating Mechanical Automata from Mocap Sequences. *ACM Trans. Graph.*, 32(6):186:1–186:11, November 2013. 17, 23

[CPMS14] Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. Field-aligned Mesh Joinery. *ACM Trans. Graph.*, 33(1):11:1–11:12, February 2014. 17, 22

[CTN+13] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational Design of Mechanical Characters. *ACM Trans. Graph.*, 32(4):83:1–83:12, July 2013. 1, 17, 23

[CZL+15] Xuelin Chen, Hao Zhang, Jinjie Lin, Ruizhen Hu, Lin Lu, Qixing Huang, Bedrich Benes, Daniel Cohen-Or, and Baoquan Chen. Dapper: Decompose-and-pack for 3d Printing. *ACM Trans. Graph.*, 34(6):213:1–213:12, October 2015. 25, 52

[DHL14] Jérémie Dumas, Jean Hergel, and Sylvain Lefebvre. Bridging the Gap: Automated Steady Scaffoldings for 3d Printing. *ACM Trans. Graph.*, 33(4):98:1–98:10, July 2014. 17, 24, 52

[DL97] M. Daniel and M. Lucas. Towards Declarative Geometric Modelling in Mechanics. In P. Chedmail, J.-C. Bocquet, and D. Dornfeld, editors, *Integrated Design and Manufacturing in Mechanical Engineering*, pages 427–436. Springer Netherlands, 1997. 21

[DO09] G. Daian and B. Ozarska. Wood waste management practices and strategies to increase sustainability standards in the Australian wooden furniture manufacturing sector. *Journal of Cleaner Production*, 17(17):1594–1602, November 2009. 51

[ES09] Koos Eissen and Roselien Steur. *Sketching: Drawing Techniques for Product Designers*. BIS Publishers, Amsterdam, April 2009. 27

[FSY+15] Chi-Wing Fu, Peng Song, Xiaoqi Yan, Lee Wei Yang, Pradeep Kumar Jayaraman, and Daniel Cohen-Or. Computational Interlocking Furniture Assembly. *ACM Trans. Graph.*, 34(4):91:1–91:11, July 2015. 24, 52

[GJMB06] Gordon Brennan, John Brown, Mike Docherty, and Barrie Tullett. *Flat-Pack/PlaskaPaczka*. The Caseroom Press, 2006. 52

[GSMCO09] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: An Analyze-and-edit Approach to Shape Manipulation. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 33:1–33:10, New York, NY, USA, 2009. ACM. 21

[Hal12] Bjarki Hallgrimsson. *Prototyping and Modelmaking for Product Design*. Laurence King, London, September 2012. 27, 28

[HBA12] Kristian Hildebrand, Bernd Bickel, and Marc Alexa. Crdbrd: Shape Fabrication by Sliding Planar Slices. *Comp. Graph. Forum*, 31(2pt3):583–592, May 2012. 17, 22

[HJW15] Kailun Hu, Shuo Jin, and Charlie C. L. Wang. Support slimming for single material based additive manufacturing. *Computer-Aided Design*, 65:1–10, August 2015. 24

[HK12] Martin Habbecke and Leif Kobbelt. Linear Analysis of Nonlinear Constraints for Interactive Geometric Modeling. *Computer Graphics Forum*, 31(2pt3):641–650, 2012. 70

[Jyl10] Jukka Jylänki. A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing. *Retrieved from http://clb.demon.fi/files/RectangleBinPack.pdf*, 2010. 18, 25, 60

[KB74] Don Koberg and Jim Bagnall. *The Universal Traveler: A Soft-Systems Guide to: Creativity, Problem-Solving, and the Process of Reaching Goals*. W. Kaufmann, Los Altos, Calif, revised edition edition, 1974. 10

[KLP01] Tom Kelley, Jonathan Littman, and Tom Peters. *The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm*. Crown Business, New York, 1st edition, January 2001. 27

[KLY$^+$14] Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating Works-like Prototypes of Mechanical Objects. *ACM Trans. Graph.*, 33(6):217:1–217:9, November 2014. 7

[LBRM12] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning Models into 3d-printable Parts. *ACM Trans. Graph.*, 31(6):129:1–129:9, November 2012. 25, 52

[LJGH11] Xian-Ying Li, Tao Ju, Yan Gu, and Shi-Min Hu. A Geometric Study of V-style Pop-ups: Theories and Algorithms. In *ACM SIGGRAPH 2011*

Papers, SIGGRAPH '11, pages 98:1–98:10, New York, NY, USA, 2011. ACM. 23

[LSH$^+$10] Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. Popup: Automatic Paper Architectures from 3d Models. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 111:1–111:9, New York, NY, USA, 2010. ACM. 23

[LSZ$^+$14] Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. Build-to-last: Strength to Weight 3d Printed Objects. *ACM Trans. Graph.*, 33(4):97:1–97:10, July 2014. 17, 24

[MLB12] Stefanie Mueller, Pedro Lopes, and Patrick Baudisch. Interactive Construction: Interactive Fabrication of Functional Mechanical Devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 599–606, New York, NY, USA, 2012. ACM. 23

[MSM11] James McCrae, Karan Singh, and Niloy J. Mitra. Slices: A Shape-proxy Based on Planar Sections. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 168:1–168:12, New York, NY, USA, 2011. ACM. 17, 22

[NP82] J. Nievergelt and F. P. Preparata. Plane-sweep Algorithms for Intersecting Geometric Figures. *Commun. ACM*, 25(10):739–747, October 1982. 38

[Osb79] Alex F. Osborn. *Applied Imagination*. Scribner, New York, June 1979. 11

[PG98] D. T Pham and R. S Gault. A comparison of rapid prototyping technologies. *International Journal of Machine Tools and Manufacture*, 38(10–11):1257–1287, October 1998. 14

[PWLSH13] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make It Stand: Balancing Shapes for 3d Fabrication. *ACM Trans. Graph.*, 32(4):81:1–81:10, July 2013. 17, 24, 52, 92

[SBSS12] Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. CrossShade: Shading Concept Sketches Using Cross-section Curves. *ACM Trans. Graph.*, 31(4):45:1–45:11, July 2012. 19

[SCMI13] Daniel Saakes, Thomas Cambazard, Jun Mitani, and Takeo Igarashi. PacCAM: Material Capture and Interactive 2d Packing for Efficient Material Usage on CNC Cutting Machines. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 441–446, New York, NY, USA, 2013. ACM. 18, 25

[SKSK09] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. Analytic Drawing of 3d Scaffolds. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 149:1–149:10, New York, NY, USA, 2009. ACM. 19

[SLZ^{+}13] Tianjia Shao, Wilmot Li, Kun Zhou, Weiwei Xu, Baining Guo, and Niloy J. Mitra. Interpreting Concept Sketches. *ACM Trans. Graph.*, 32(4):56:1–56:10, July 2013. 19, 20, 30

[SP13] Yuliy Schwartzburg and Mark Pauly. Fabrication-aware Design with Intersecting Planar Pieces. *Computer Graphics Forum*, 32(2pt3):317–326, 2013. 17, 22

[SSL^{+}14] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and Fabrication by Example. *ACM Trans. Graph.*, 33(4):62:1–62:11, July 2014. 23

[SSM15] Maria Shugrina, Ariel Shamir, and Wojciech Matusik. Fab Forms: Customizable Objects for Fabrication with Validity and Geometry Caching. *ACM Trans. Graph.*, 34(4):100:1–100:12, July 2015. 20, 52

[SVB^{+}12] Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. Stress Relief: Improving Structural Strength of 3d Printable Objects. *ACM Trans. Graph.*, 31(4):48:1–48:11, July 2012. 17, 24, 52, 92

[TGY^{+}09] Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. Exploratory Modeling with Collaborative Design Spaces. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 167:1–167:10, New York, NY, USA, 2009. ACM. 20

[UE07] Karl T. Ulrich and Steven D. Eppinger. *Product Design and Development*. McGraw-Hill, Boston, 4th edition, July 2007. 28

[UIM12] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided Exploration of Physically Valid Shapes for Furniture Design. *ACM Trans. Graph.*, 31(4):86:1–86:11, July 2012. 1, 20, 92

[VGB$^+$14] J. Vanek, J. A. Garcia Galicia, B. Benes, R. Měch, N. Carr, O. Stava, and G. S. Miller. PackMerger: A 3d Print Volume Optimizer. *Computer Graphics Forum*, 33(6):322–332, September 2014. 25, 52

[WWY$^+$13] Weiming Wang, Tuanfeng Y. Wang, Zhouwang Yang, Ligang Liu, Xin Tong, Weihua Tong, Jiansong Deng, Falai Chen, and Xiuping Liu. Cost-effective Printing of 3d Objects with Skin-frame Structures. *ACM Trans. Graph.*, 32(6):177:1–177:10, November 2013. 17, 24, 52

[XCS$^+$14] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. True2form: 3d Curve Networks from 2d Sketches via Selective Regularization. *ACM Trans. Graph.*, 33(4):131:1–131:13, July 2014. 19

[XWY$^+$09] Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel van de Panne, Falai Chen, and Baining Guo. Joint-aware Manipulation of Deformable Models. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 35:1–35:9, New York, NY, USA, 2009. ACM. 21

[XZCOC12] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Fit and Diverse: Set Evolution for Inspiring 3d Shape Galleries. *ACM Trans. Graph.*, 31(4):57:1–57:10, July 2012. 20

[YCL$^+$15] Miaojun Yao, Zhili Chen, Linjie Luo, Rui Wang, and Huamin Wang. Level-set-based Partitioning and Packing Optimization of a Printable Model. *ACM Trans. Graph.*, 34(6):214:1–214:11, October 2015. 25, 52

[Yva08] Pierre-Alain Yvars. Using constraint satisfaction for designing mechanical systems. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 2(3):161–167, August 2008. 21

[ZCC$^+$12] Youyi Zheng, Xiang Chen, Ming-Ming Cheng, Kun Zhou, Shi-Min Hu, and Niloy J. Mitra. Interactive Images: Cuboid Proxies for Smart Image Manipulation. *ACM Trans. Graph.*, 31(4):99:1–99:11, July 2012. 21

[ZFCO$^+$11] Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise Controllers for Structure-Preserving Shape Manipulation. *Computer Graphics Forum*, 30(2):563–572, 2011. 21

[ZXS+12] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided Mechanical Toy Modeling. *ACM Trans. Graph.*, 31(6):127:1–127:10, November 2012. 17, 23