UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Synthèse de Formes Fabricables à partir de Specifications Partielles

THÈSE

présentée et soutenue publiquement le 1er Février 2017

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Jean Hergel

Composition du jury

Rapporteurs :	Bernhard Thomaszewski Marco Attene
Examinateurs :	Joëlle Thollot Monique Teillaud
Encadrant :	Sylvain Lefebvre

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Table des matières

Introduction

Contributions

Introduction

Introduction

The development of Makerspaces and Fablabs is a good indicator that rapid manufacturing blossomed in the past few years. Most of these spaces are equipped with Fused Filament Fabrication (FFF) 3D printers and laser cutters [1], which are inexpensive and easy to operate technologies for fabrication. The popularization of those spaces led to the development of inexpensive and more user friendly 3D printers. It is now very simple to manufacture physical objects from virtual models, without any specific expertise. Despite their simplicity these machines produce functional physical objects in a variety of materials (plastics, nylon, wood, etc.). Those technologies are important to study because they are popular and accessible, and thus any improvement can bring benefits to a large user base.

Industrial 3D printers were initially targeted at specialists operating the machine, close to the final fabrication process. The increased accessibility of rapid prototyping now allows the initial designer to also be the one operating the machine, and let us envision that the designer might become anyone, from home users to enthusiasts and small entrepreneurs. Interestingly, the main barrier to entry for novel users is no longer the equipment or how to operate it, but rather how to create interesting and useful 3D models to print. Hence we need software that helps end users directly produce parts that can be fabricated, where the software takes into account the constraints of the process, the desired function of the object, and the structural constraints on the final product.

Making an object is not a simple task. It is a complex process that begin by the design of a shape, its function, its aesthetics, while taking into consideration all the various constraints. The process ends by the fabrication itself. Throughout this thesis, we will see novel techniques aimed at simplifying and improving the process of making objects on rapid fabrication technologies. Algorithms can improve the quality of a fabricated model without interfering with the design of the shape. Algorithms may also synthesize novel shapes from incomplete specifications or cooperate with the designer to model fabricable shapes.

The techniques I propose in this thesis explore a trade off along a line between the user and the machine (see Figure 1). The closer the cursor is to the user, the more the algorithm helps her to create the shape. The closer the cursor is from the printer, the less the algorithm changes the design and instead tries to adapt the process.

The increasing popularity of 3D printers led to a change of the community of people that are using the technology. It ranges from designers that are expert users of design tools to hobbyists who do not master these tools. The best software is not the same for

1. `http://fabfoundation.org/setting-up-a-fab-lab/`

different groups of the community. While a design made by an expert might be a work of art that should not be deformed, and fabricated as it is, a hobbyist might expect the software to help him realize his vision. Those different expectations give importance to the exploration of the different positions of the cursor.

The next chapter presents the state of the art of a wide range of methods from the optimization of the 3D printing process to the synthesis of shapes. After the state of the art, I detail my contributions.

The first approaches I introduce focus on the optimization of the 3D printing process. Chapter 1 presents a method to improve the quality of a model printed with multiple materials. I also discuss my contribution to the design of support structures, that are sacrificial scaffoldings added to the geometry of a design to ensure it prints correctly. Chapter 2 introduces techniques that handle fabrication constraints as part of the modelling process. A first approach enables partial 2D specifications of mechanisms to be automatically transformed into fabricable 3D mechanisms. A second approach helps reduce wastage on parametric models made by assembling laser cut planar parts. Moving the cursor even closer to the user, Chapter 3 introduces a technique helping users to model entire designs (in this case furniture) from functional specification (supporting objects located in space).

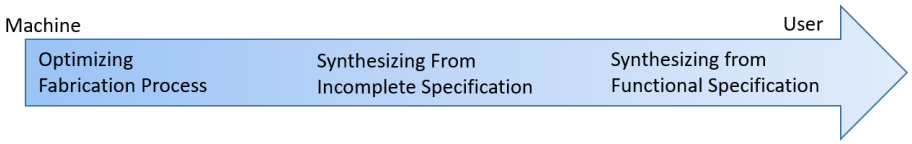


FIGURE 1 – The line between the user and the machine. The closer the cursor it is to the machine (left), the more precise the specification by the user has to be.

State of The Art

1 Overview

Digital manufacturing is everywhere in modern factories, where computer controlled machines turn virtual designs into actual objects. Objects are mass produced through complex chains of operations involving different machines and processes. However, while largely adopted for mass production, producing prototypes remained a manual activity for a much longer time, involving highly skilled and trained specialists handcrafting physical models. This motivated the research on rapid prototyping techniques. They allow fabricating unique prototypes at low cost, and have been designed to turn a virtual object into a physical prototype as automatically (and quickly) as possible.

Rapid prototyping relies mostly (but not exclusively) on additive manufacturing techniques which is the process of making an object by stacking successive layers of material. There are different technologies based on different principles. Here are the main ones :

- **Selective Laser Sintering (SLS)** is the principle of using a laser to fuse powder material. Powder is added to the bed layer after layer (typically with 0.1 mm thickness). The laser then locally solidifies contours. The next layers is added on top, progressively forming the object (see Figure 1).
- **Binder Jetting** uses a binder to bond powder material similar to plaster. The binder is deposited by a print head similar to a standard inkjet 2D printer, directly onto the powder bed.
- **Stereolithography Aparatus (SLA)** uses light to solidify a liquid photo-sensible resin with a projector or a laser and mirrors (Figure 1).
- **Fused Filament Fabrication (FFF)** uses melted plastic to build the object (Figure 4). As I will be focusing on this process later on, it is explained in more detail in Section 2.

While initially limited to prototyping, the wider range of available materials and the improvements in processes make it now possible to use additive manufacturing to produce functional parts. Among those technologies, fused filament fabrication blossomed the last decades. It is used in fablabs and makerspace to repair or fabricate customized objects, where mass production is not an option because it is too expensive.

Other fabrication techniques such as laser cutting or CNC milling reached maturity and became simpler to operate. Free software and firmwares are available (*Grbl, LinuxCNC* are open source solutions for example) for those technologies, that are also present in fablabs and makerspaces.

Makerspaces, fablabs and start-ups massively adopted those technologies. This led to

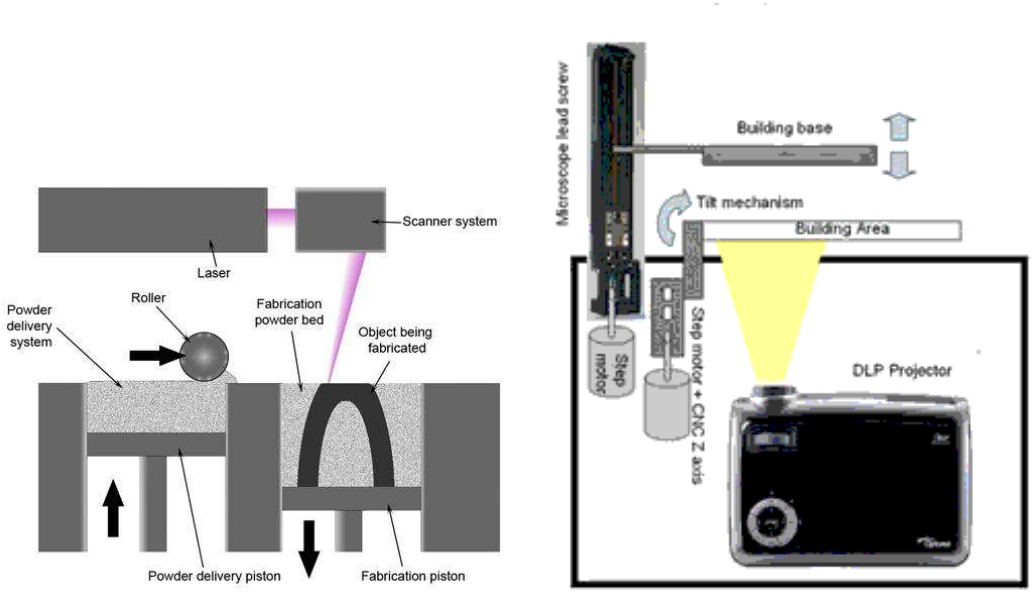


FIGURE 1 – **Left)**a selective laser sintering machine works by depositing bowder that is locally solidified by a laser. Image from Wikipedia(`www.wikipedia.fr`).**Right :**a Stereolithography apparatus machine works with a projector that is solidyfing a liquid photosensible.

a profound change of the typical user profile. At first, the processes were used by expert technicians with extensive training. Nowadays, these machines are used by hobbyists, teachers, small entrepreneurs who want to produce unique and customizable products [73]. This evolution creates a need for new software tools that are usable by a wider range of users. My work is part of this trend, focusing on algorithms that can help exploiting available technologies of rapid manufacturing without significant training in CAD software and processes.

Figure 2 shows a set of 3D printed models from the website *thingiverse* which is a website where people share models to fabricate. As can be seen a wide range of objects are being shared, from chess set pieces, to a functional 3D mechanical clock, and a small music box. All these objects have been created to be 3D printed on inexpensive filament 3D printers. Such sharing platforms are a good example of the profound evolution triggered by the massive availability of 3D printers. Figure 3 shows a set of laser cut models. There is an arcade, a dragonfly sculpture and a bottle holder that have been laser cut and assembled. Note how all these models are neatly assembled from planar cut-outs.

In Section 2 and Section 3 I discuss challenges in fabricating shapes with both FFF and planar cutting. There is a strong link between the geometry of a design and how challenging it is to fabricate on a specific technology. For instance, a given model might contain features that cannot be printed because they are too thin, in which case the fabrication software has to decide (perhaps arbitrarily) either to remove or thicken these features. In some cases, the software will simply reject the design as not fabricable. However, small changes to the design – either by the user or by an algorithm – can greatly simplify the

FIGURE 2 – A chess game *(thingiverse thing 1575432)*, a mechanical clock *(thingiverse thing 328569)*, a music box *(thingiverse thing 53235)*. All those model have been 3D printed

FIGURE 3 – An arcade *(thingiverse thing 1428410)*, a bee *(thingiverse thing 297758)*, and a pack of bottle laser cut *(thingiverse thing 16290)*. All those laser cut models require an assembly.

fabrication process. This avoids arbitrary decision by the machine software, and in some cases is the only way to produce the model.

Research has been conducted on both fronts : Some techniques have been developed to print despite problems in the model, while others have been developed to help users design shapes that can be fabricated more reliably. I present next both methodologies for each of the challenges on filament 3D printing and planar cutting. In Section 4 and 5 I describe techniques that go even further and automatically model objects from partial user inputs. For example, I describe topology optimization, which is a mathematical technique that optimizes a shape from a set of loads placed in space. These techniques are not necessarily dedicated to specific manufacturing tools. They are focusing on helping users model or synthesize objects that have a specific purpose.

2 Modelling and printing with Fused Filament Fabrication

Fused Filament Fabrication (FFF) is the process of making an object with melted plastic. Plastic is pushed by a step-by-step motor in a hot tool-head that is moving on a build plate and depositing plastic to build the object layer after layer (See Figure 4). This

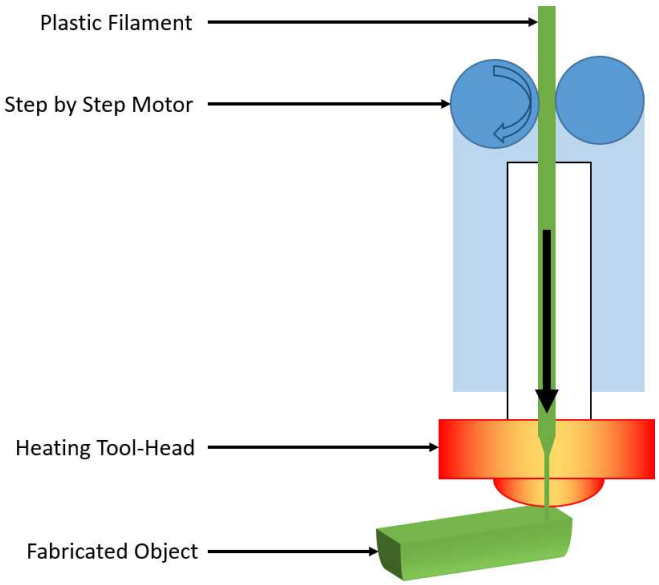


FIGURE 4 – Fused Filament Fabrication

is an additive manufacturing technology : the matter is added layer after layer to build an object. The layer thickness is typically between 0.1 and 0.3 millimetre. Usually, printers are equipped with a single print-head but some models have two or more. Having multiple print-heads allows printing multi-material models, and is often used to print multi-color models or to print temporary structures in a dissoluble material.

A software tool is needed to compute the paths along which the extruder deposits plastic. This software is called a *slicer* since its main operation is to slice the 3D representation of an object into thin layers (See Figure 5). To compute those layers, the slicer computes the intersection between a plane swept along the Z axis and the boundary of the model [58]. The intersection between the plane and the surface is a set of closed

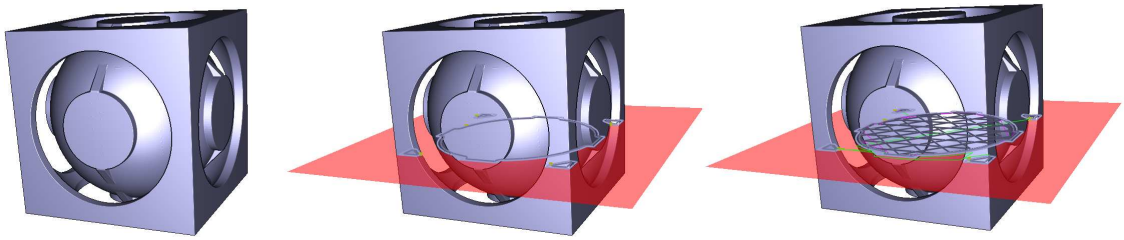


FIGURE 5 – This figure shows the slicing process. The slicer compute the intersection between an XY plan and the model. From the intersection, the software compute all paths that will be followed by the extruder. **Left :** The Square ball model **Center :**The intersection between a plan and the surface of the model**Right :** The paths at the corresponding height

2D polygons (assuming the input shape defines a solid). From those polygons the slicer computes the deposition paths. There are four main types of paths :

— **The perimeters** which represent the visible contour of the model. They are obtained by offsetting the polygons by half the diameter of the extruder.
— **The shells** that are computed by offseting the perimeters inwards. The more shells there are, the sturdier the external surface hull is.
— **The infills** that are inside the shell and fill the printed model. Infills can be dense or sparse, to avoid spending time and material printing inner parts of the model.
— **The travels paths** that are the motions done by the extruder between two other paths. An important difference with other paths is that travel paths are free variables : they do not define the actual geometry of the printed object. However, as I detail in Chapter 1 these paths may have a detrimental impact on surface quality if not carefully computed, in particular when printing with multiple materials.

There is a wide body of research regarding how to slice and to generate tool paths for additive manufacturing. For a review of the main aspects the reader can refer to the survey by Pandey [80] the SIGGRAPH course by Dinh and colleagues [28].

2.1 Challenges of Fused Filament Fabrication

Existing previous works tackle several challenges, for instance improving the quality of surface of 3D printed objects, ensuring that a model can print, decreasing the print time, improving the mechanical robustness of the models and minimizing the material wastage.

Those challenges are described in detail below as well as the techniques that have been used to solve them. Many of these challenges interact. For example, decreasing the print time might lead to a decrease in the quality of the printed parts ; or reducing the amount of material used to fabricate a model might decrease its mechanical robustness.

As I explained in the overview, some techniques have been developed to print the model as-is. They impact only the way the object is fabricated and focus on the *process stage* : the steps between the input design and the actual physical device. Other techniques impact the *design stage* by providing user interfaces that help the user to model fabricable objects or by optimizing the design to make it print more reliably. For each challenge, I introduce both methodologies whenever applicable.

2.1.1 Surface Imperfection

Even if the model is printable on a filament printer the process might not go well and several defects might appear on the surface of the printed parts (See Figure 6). They mostly come from the fact that the melted plastic is oozing from the extruder by gravity (See Figure 6 Right). The plastic that oozed ends up being deposited on the surface of the printed object. Oozing leads to another defect : holes (See Figure 6 Centre). The plastic that oozed is now missing, and the software that planned the motions is unaware of this. The extruder needs to be refilled to avoid holes in the printed part. Slicers reduce oozing by relying on retractation : the plastic filament is pulled backwards by the motor just before travel moves, thus depressurizing the melted plastic chamber inside the nozzle. While plastic will not ooze immediately, oozing still restarts after a few seconds. Due to

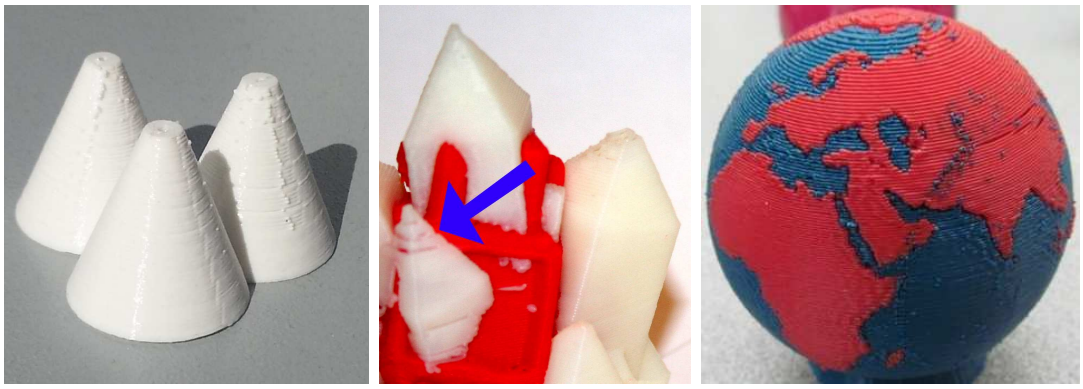


FIGURE 6 – The different kinds of defects. The holes (center) and ooze (right) appears more likely in multi material print. Zippers (Left : appears even with single color prints.

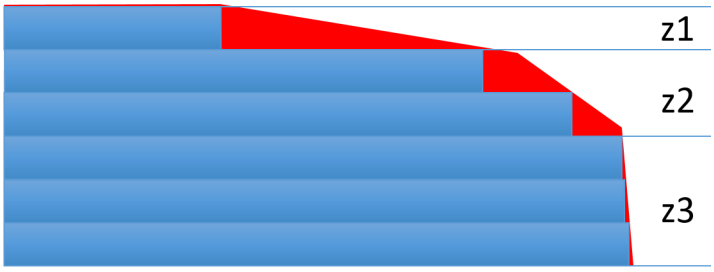


FIGURE 7 – The volumetric error (in red) depends of the gradient of the surface. The more horizontal it is, the bigger is the error. The zone shown in the figure presents the error in function of the gradient of the surface.

this, oozing and holes are more likely to appear in multi-material prints since extruders might be idle for long periods while others print.

Another defect are so-called *zippers*. The position of the endpoint of the perimeters are visible on the final model (Figure 6 Left), producing defects that look like zippers along the entire height of the printed part. This defect always appears but may be less visible with a well-calibrated 3D printer.

Chapter 1 presents a technique that limits those three defects.

Slicing the object into layers produces an approximation of the original surface. This produces a stair case defect, visible on the surface of most 3D printed objects. This defect reveals the approximation error with respect to the initial surface (see Figure 2.1.1). The quality of the approximation depends on the thickness of the layers used during the slicing process. For a same input object, the impact of layering can also be reduced by changing the object orientation, whenever possible (in particular this interacts with the need for supports, see Section 2.1.3). Thus, several approaches have been proposed to optimize for the object orientation such as to increase surface quality [80]. Build orientation has been studied a lot, the interested reader can refer to the survey by Taufik et al. [112].

A drawback of using thinner layers is that the print time may significantly increase. To avoid this, adaptive slicing methods have been proposed [113, 107, 29]. These techniques change the height of the layers depending on the model geometry. However they are not able to adapt to changes in complexity *within* the layer. To tackle this problem locally adaptive slicing has been introduced [116]. The key idea is to divide the model into region that are sliced independently [126]. The mains issue is that zippers appears along the surface, where the slices with different thicknesses meet. This smear can be avoided by slicing the interior with a larger thickness than the exterior [91].

Another way to further improve quality is to decompose the input into different parts, and change the build orientation of each part after decomposition [41].

2.1.2 Multi Material

Filament printers are sometimes equipped with multiple extruders. Those additional tools allow users to print with multiple materials, and are often used to have different colors within a same object.

In Chapter 1, I propose a method that improves part quality when printing with multiple extruders. Here is how the main available slicers deal with this problem. The input to the slicing process is a set of meshes. Each one describes the volume that has to be filled with a material by the corresponding extruder. Each mesh is sliced independently and the tool paths are merged, typically printing materials in sequence. *KisSlicer*[2] deals with oozing by creating a wipe tower on the side of the print and by wiping the extruder on this tower before switching tools. *MakerBot Desktop*[3] deals with oozing by adding a wall on each side of the print to wipe the extruder before printing. *RepRapPro*[4] lowers the temperature of the idling extruder to avoid oozing. However, this increases print time significantly because the printer is not printing while the temperature is decreasing. Inspired by dithering techniques, Reiner et al. [87] presented a method to produce objects that look like the colors in the extruders were mixed. For instance, using black and white filaments it can print objects with shades of gray.

MakerBot and *KisSLicer* avoid holes by printing the wall or the tower with the next printing extruder. This refills the extruder before the printing the actual object. With this technique, holes may appear in the wall or the cleaning tower – one danger is that they might not be robust enough to clean the extruder and avoid oozing. Thus, the size of the walls and towers has to be large enough to ensure it remains robust.

2.1.3 Overhangs

Newly extruded filament *has* to be deposited over an existing surface (the build plate or previously deposited filament) or it will fall by gravity and the print fails (Figure 8). That is why it is necessary to compute a support structure that will be printed at the same time as the model. This structure is removed and disposed of after fabrication.

2. `http://www.kisslicer.com/`
3. `https://www.makerbot.com/download-desktop/`
4. `https://reprappro.com/`

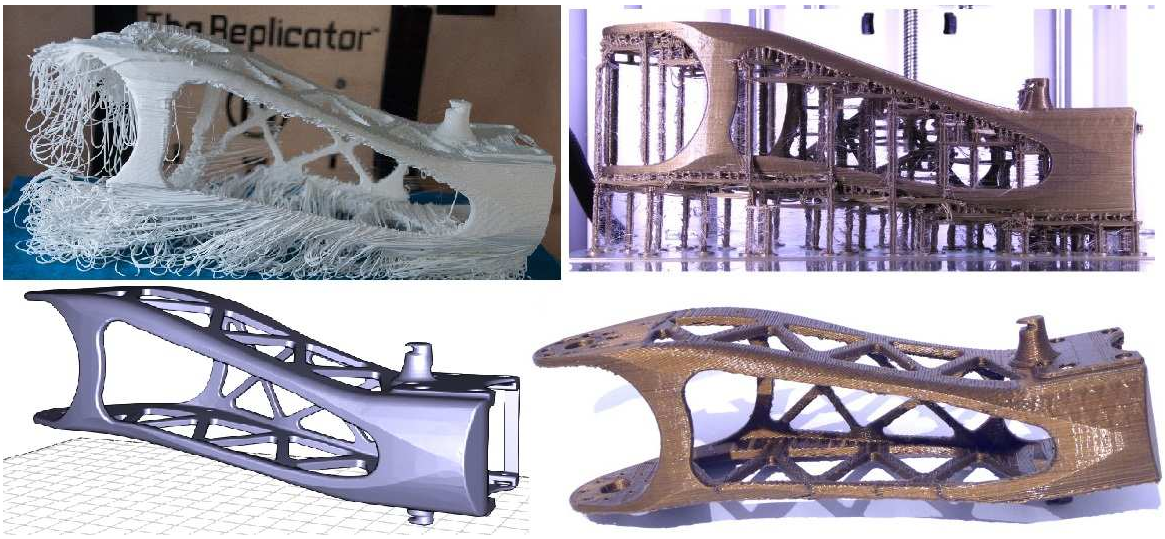


FIGURE 8 – **Bottom-Left :** The leg of puppy model. **Top-Left :**Without support structure the bottom part is not printed correctly. **Top-Right :** With support structure, the print is correct. **Bottom-Right :** After the removals of the support structure.

Some materials are dissoluble in special liquids (HIPS is dissoluble in D-Limonene, PVA is dissoluble in water) and available as filaments that can be used in 3D printers. They can be used to print the support structure with a second extruder mounted on the printer (in which case all the challenges mentioned in the previous paragraph apply).

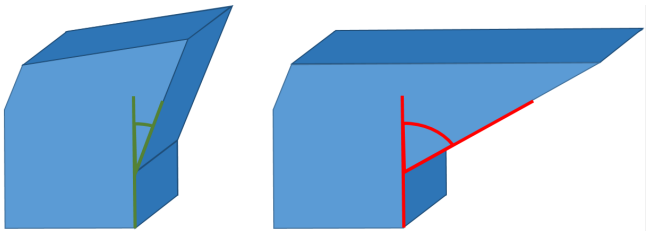


FIGURE 9 – and example of overhang. **Left** The angle (in green) between the Z axis and the face is smaller than the threshold, the dark blue face does not require support. **Right** The angle (in red) is bigger than the threshold, the face require support

Support structures are a major inefficiency of additive manufacturing processes, as they waste time, material and require manual finishing steps. Therefore, reducing the need for supports and minimizing their impact has been, and remains, a very active topic of research. This is a case where researchers have proposed techniques both at the process stage (design remains unchanged) and at the design stage (guiding users towards designs that require less supports).

Process Stage. First, one needs to compute the regions that require support, and then has to compute the shape of the structure itself.

The regions that require support are analyzed in several ways. It can by comparing the angle between the vertical and the faces of the model [1, 3]. If the angle is bigger than a threshold, then the face require support (see Figure 9). Detection can be done by a boolean difference between two successive slices [4, 16], using the width of the difference to determine if a region is self-supported. In our work we proposed to compute it from the print paths and the slices. For each point of each print path, the algorithm considers whether the slice below covers the point to be supported (checking which percentage area of a disk centered on the point and having the extruder width is covered). Note that other techniques have been proposed, for instance based on image processing [44].

Several techniques have been proposed to generate supports. Some form a dense support volume below the surfaces in overhang. The support volume is then usually printed with a weak infill pattern (*KISSlicer*, *Makerware*, [106]). The support is manually removed by breaking it apart from the object. Soluble material can also be used on multiple material printers [54]. Printing the support volume uses a significant amount of material and print time, but is very reliable : the support typically has a large area of contact with both the part and the print bed, ensuring the print stability in most cases. The volume is large enough to print without difficulty. A number of approaches modify the support volume to reduce its size. Huang et al. [45] use sloped walls instead of straight walls for the sides, shrinking the support volumes in their middle sections. Heide [37] also reduces the support volume by decreasing its size and complexity as the distance below the supported model increases. The size of the support structure depend of the size of the area that require supports. This area can be reduced by changing the orientation of the model on the build-plate [1]. Zhang et al.[140] optimizes the orientation to avoid support in zone of the model that have an impact on the perception of the model by an human.

Another family of approaches, increasingly popular, produce a sparse set of points that need to be supported, usually by down-sampling [31] the initial set of points. Vanek et al. produce a geometry based approach to support this set of point [121]. Schmidt and Umetani [93] introduce branching support structure.

Design Stage The support structure has a significant impact on the quality of surface of the model. When the support is removed, it leaves white dots on the printed parts. There exists method that deform a shape to minimize the quantity of support [43]. Reiner et al. [88] presented a design tool that allows the user to model 3D object that does not require support structure.

2.1.4 Print Time

The first point of 3d printing was to build prototypes quickly to let the designer iterate and modify an initial design idea. However the print time is often in hours, and reducing it would further fasten design iterations. Decreasing the print time by augmenting the speed of the extruder leads to defects that appear on the surface of the models. In addition, current printer firmwares take the maximal acceleration of the extruder into account during the fabrication stage, and this often constitutes the limiting factor. Therefore, the

print time depends mostly on the path generation and the position of the different parts on the print bed if there is more than one.

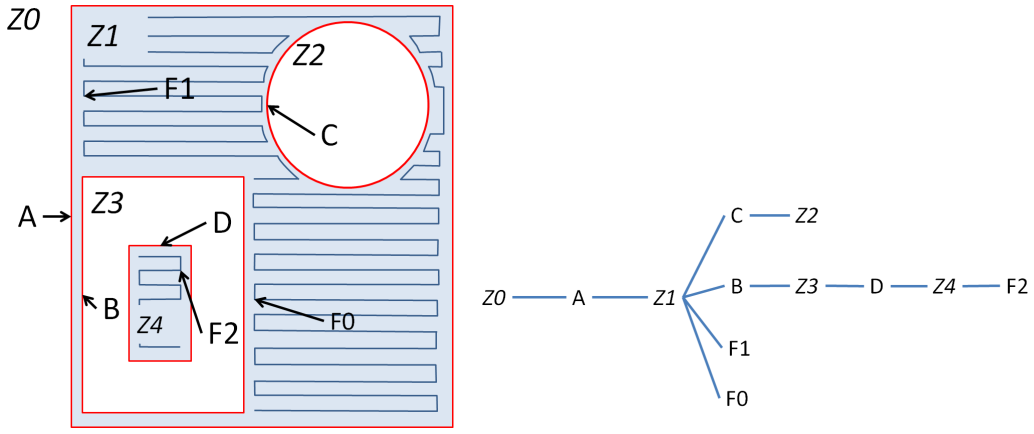


FIGURE 10 – Left : a slice with perimeters (in red) and infill paths (in blue). The perimeters are cyclic and delimit zone. Right : the tree structure shows the topology of the different paths

Process Stage Several approaches optimize the tool-paths to minimize the print time [14]. This leads to a Travelling Salesman problem (TSP) formulation [124] where each node of the graph is a printing path and the edge are the distance between each path. In general, print paths are closed paths and the formulation of the tool-path optimization with a TSP is limited to methods that are selecting the in/out position of the extruder for each path before the optimization. However, a TSP can be generalized (GTSP) [135] : the node of the graph are grouped into clusters that are mutually disjoints. The searched solution is the path that visits exactly one node of each clusters. This problem is still NP-Hard and therefore it is recommended to use a heuristic to optimize it. We can note that perimeters and shells are cycle and they are including infills or other shells (see Figure 10). This information helps to tailor a good heuristic that optimize the tool-path [21].

If the path planning of concurrent tool has been investigated [20] it is less relevant in the case of filament printers where the different extruders are most often fixed to the same carriage (variants exists though).

The layering direction can be optimized to minimize the height of the part on the build plate. The motion speed of the printer is not the same for each axis [115]. Z axis moves slower, reducing the height reduces the build time.

The print time can be significantly reduced by hollowing and splitting the object hull in parts. I mention those techniques in section 2.1.5 or by printing the shape around an empty core [102, 35]

Design stage Mueller et al. [74] presented a method to compute and fabricate wire like low-res models that can be 3d printed faster than the designed model. This method has been extended to 5 degrees of freedom (DOF) printers [134] which allows a better shape

approximation. It has been extended to a 6 DOF robotic arm with an extruder [46]. The print time can also be decreased by substituting some part of the model with construction blocks (eg. Lego) [77]

2.1.5 Model Decomposition

Usually, the build volume of filament printers is around $200 * 200 * 200\ mm^3$. If the model is too large to be printed, it can be partitioned in smaller parts that will be assembled latter. In other cases the user might want to decompose the model before the fabrication. For example mechanical parts might be fused together during the process, which will prevent them from working properly.

All parts have to be printable and it should be possible to assemble the final model from the parts. Such a partition can be done by taking into account the size of the parts, the interface between the parts, the robustness and the aesthetics of the assembled models [62]. The model can be decomposed to save print time and to minimize support structure [122]. The model can be decomposed and packed to be printed in a single print then assembled. Attene [5] tackles the problem of shape segmentation and packaging to ship the model to a customer who will later assemble the parts. Dapper [18] considers the problem of decomposing the input into pyramidal shapes, packed into the print bed using a voxel representation of the parts. The decomposition of the model can be computed to save space during the shipping to a customer [139]. The decomposed model can be assembled with small connectors [62] or by creating interlocking parts [103].

2.1.6 Physical Properties after Fabrication

We need to be sure that the target model is fabricable and that it will endure its intended everyday use. Stava et al. [104] introduce an optimization technique that optimize a model to ensure the physical integrity of a 3D printed model. Zhou et al. [143] perform a modal analysis to extract the weak region of a 3D model. Langlois et al. [56] presents a method to compute failure probabilities with Finite Element Method (FEM).

The elastic models are deforming under gravity and the user has to predict the deformation and to edit the model such that the deformed model is the one that the user want to build. Chen et al. [19] presents a technique that compute automatically the inverse elastic shape.

There is a trade-off between the stress resistance of a 3D printed object and the quantity of material that is use inside the model. By creating honeycomb patterns inside the model, with a little impact on the strength of the 3D printed model [61].

2.1.7 Material Usage

The amount of material usage depends of the way the model interior is filled. Minimizing the amount of material inside the model minimizes also the fabrication cost (time and material) of the model. The techniques that tackle this challenge focus mostly the process stage.

Kisslicer, MakerBotDesktop, Cura[5] let the user choose the amount of plastic used inside the models. They use different infill patterns (e.g. straight line, Hilbert, honeycomb) and they do not truly consider ensure the printability of the model : when the inside is sparse, internal support might become necessary for the object tops. Wu et al. [133] consider the optimization of a rhombic infill that ensures that the inner structure is self-supported. Kumar et al. [55] uses hierarchical space filling curves to control the density of the infill pattern.

The minimization of material usage can also be done by hollowing the object to fasten print time and decrease the material usage. Hollowing is done by computing the offset surface of the model [32, 89]. The offset surface can be extracted from the distance field of the model.

The interior of a model can be filled with spare struss structure. Wang et al. [127] optimize those truss to preserve rigidity and use the reduce the number of beams. Zhang et al. [141] uses the medial axis as a skeleton from which grow a tree like structure. Medeiros et al. [67] generate an adaptive tessellation of the interior, and offset the edges of either the primal or the dual to produce an inner beam structure. It allows to generate a denser structure along the shape boundary than inside. Lu et al. [61] optimize for a Voronoi diagram inside the print, which faces form an infill pattern.

The complexity of the infill pattern might leads to an increase of the travel time. Yaman et al. [137] tackle this problem by printing the face of a Voronoi diagram. They use Euler loops to minimize in a graph where the edge are the face of a voronoi diagram to avoid travel move.

3 Planar cutting

Planar cutting is a subtractive technology. Usually, a panel made of wood, plastic or textile is cut into different planar parts that are assembled to build a 3D object. The cutting tool might be of different nature (laser, CNC, water jet, etc.). It requires to compute cutting paths like filament printers require to compute deposition paths for the extruder. The paths are not computed by exactly the same method however, as the tool in this case removes material, and issues such as oozing and zippers do not occur (see Figure 11).

Planar cutting is very common in many industries. It is used to fabricate wooden structure for boats, furniture, and to cut the fabric and the leather of our clothes. A complete overview would extend well beyond this thesis. This section thus focuses on assembly of 3D shapes from 2D cutouts of rigid material like woods or acrylic, in the context of personal and hobbyist fabrication.

Laser cut objects almost always require an assembly. Industrial use the technology to fabricate planar parts, present in boats or in textile industry. Clothes need to be sewn, and the wooden parts need to be assembled.

5. `https://ultimaker.com/en/products/cura-software`

FIGURE 11 – **Bottom)**The model of a round table. **Top Left :** The cutting plan to cut the different part of this table **Top Right :** The leftover of material after the cut.

3.1 Modelling and fabrication with planar cutting

There are two main challenges in planar cutting. The first one is to design objects for planar cutting. The second one is to minimize the amount of wasted material. This section follows the same organization as Section 2. First, I present the challenges that are specific to this manufacturing technology, and for each challenge I present the techniques that have been used to solve them at process or design stage.

3.1.1 Modelling 3D shapes from 2D cuts

Modelling for planar cutting is not straightforward. This presents two major drawbacks : First the cut materials are thick (e.g. a few millimeters) and this thickness must be taken into account for the subsequent assembly. Second the cut parts are almost always planar, and this limits the design possibilities. Note however that it is possible to create a pattern with the laser that allows some deformation of the parts (See figure 12).

The planar cut parts can be created by the user through a modelling interface [66, 96, 95] or optimized to best capture a 3D model [40]. This optimization can be done with crossfield [23]. The method from Mueller et al. [76] allows the user to draw directly shapes on the laser cutter with different pen. Clothes are 2D shape, cut in leather or tailored from fabrics that are deformed afterwards to be assembled. Umetani et al. [118] developed a technique to edit garment and model clothes. The method allows designers to edit a cloth by editing either the garment, or the 3D representation of the cloth.

FIGURE 12 – Those boxes have been cut with a laser cutter. The pattern allows followed by the cutter allows some deformation that permit the user to closes the boxes. *(Things 17240 from Thingiverse, user bdahlem)*

The fabrication and assembly of a model with a laser cutter is usually faster than a 3D print. Beyer et al. [10] present a technique that allows the user to replace the 3D printing by an assembly of laser cut parts. However, the fidelity of the cut model is lower than the quality of the 3D printed model. Mueller et al. introduced [75] LaserOrigami that use a planar cutter to produces 3D objects faster and without assembly. It works by changing the focus of the laser of the cutter to heat and bend the model.

3.1.2 Minimizing Wasted Material

Planar cutting is extensively used in several industries. When cutting parts from a rectangular piece of material, left over regions are wasted. This has a huge impact on cost as not only material is wasted, but also additional clean-up and waste handling is necessary. Thus, minimizing wastage has been studied in computer science since decades.

Process Stage We need to pack the different cut parts in the (typically) rectangular cut area to waste the minimal quantity of material. This nesting problem is known NP-Hard [24]. The packing algorithm are mostly heuristics however Fishetti and Luzzi [33] proposed an exact solution based on Mixed Integer Programming. Some heuristic have been developed, in particular based on the *Bottom Left* placement rule. The idea is to place iteratively each pieces in the left-most possible position. More recently Jones [49] proposed an algorithm based on the decomposition of the polygon into circles. An initial solution can be improved with *compaction* and *separation* algorithms [60] that can be thought as if forces are applied on the parts to move them in the packing space. The interested reader can find a more detailed survey in the thesis by Antonio Martínez Sykora : "Nesting problem, exacts and heuristics algorithms" [109] or refer to the tutorial by Bennell and Oliveira [8].

If a user wants to use a limited number N of panel to build an object, she has to edit her model manually until the packing algorithm founds a solution for N panels. The chapter 2 of this thesis presents a technique that optimize a shape to minimize the waste.

Design Stage Saakes et al. [90] presents a method to reuse leftover of panels. The method is interactive and the user is placing the part that she wants to cut in a virtual version of the panel. The second chapter of this thesis presents a technique that performs

design changes (within user defined bounds) to minimize the wastage of material with laser cut designs.

4 Shape Synthesis

"Forms follow function" is a design philosophy that appeared in the 20th century [108]. This is based on the fact that the form of a model depends primarily on its function. Sometimes, especially with non-experts, the user has no particular wishes about the shape of the model she wants to fabricate, however she knows precisely its intended function. In this case, the system has to create the geometry automatically from the function, without any other information. Designers call this process *generative design*[6]. To solve this kind of problems and produce shapes automatically, scientists proposed several methodologies. I describe the most related to my work in this Section.

Interestingly, the automatic synthesis of shapes (geometry and surface details) is a common problem in the video game and movie industries, see for instance the work of Ma et al. [63] where variants of game levels are produced from basic building blocks, or the work of Weta Digital [129] that recreate 1933 New York City for the movie King Kong. However the goal is mainly about aesthetics and serving a story or gameplay. While it is time consuming to design a single shape, many scenes in video games or movies require a lot of different models of the same type (e.g. a forest with different trees).

4.1 Approaches for shape synthesis

Different techniques exists to represent shapes as parametric geometries, from which infinite variations can be explored. Below I mention some examples such as procedural modelling and by example synthesis.

4.1.1 Procedural Modelling

Grammar Procedural modelling techniques are techniques based on the generation of content from a set of rules. Among them, shape grammars [105] consists of a set of rule that define how a shape can be transformed. **A parametric, stochastic, conditional, context-free Grammar** [111] is a tuple :

$$G = \langle V, T, \Sigma, \omega, P \rangle$$

where V is the set of variable, T is the set of terminals, Σ is the set of formal parameters, $\omega \in ((V \cup T) \times \mathbb{R}^*)^+$ is the axiom and $P \subset (V \times \Sigma^* \times B \times \epsilon) \times ((V \cup T) \times \epsilon^*)^*$ is a finite set of production rules. In this definition ϵ is the set of all correctly constructed arithmetic expressions with parameters from Σ. B is the set of all possible expression involving quantities form ϵ. A production rule $\rho \in P$ begins with a *predecessor* and ends with a *successor*. A predecessor is defined by a variable, any number of formal parameters, a boolean condition on the formal parameters and a function $\mathbb{R}^* \rightarrow [0, 1]$ that specifies

6. https ://redshift.autodesk.com/generative-design/

the probability of using the production rule. A successor is a set of variables or terminals. Here is an example of a grammar :

$$V = \{X\},\ T = \{S, +, -, [,]\},\ \Sigma = \{l\},\ \omega = X(1)$$
$$P = \begin{cases} X(l) : l \leq 2 \xrightarrow{1-l/2} F(l)[-X(l+1)][+X(l+1)] \\ X(l) : l \leq 2 \xrightarrow{l/2} F(l) \end{cases}$$

To use the grammar and generate a shape, the idea is as follows. From the axiom, we apply a rule and build a new chain. The system applies rules until all symbols are terminals. The rules to apply are selected with some probability (written on the top of the arrow) and under some conditions (on the left of the arrow). An example of derivation is given fig 4.1.1. We still need to generate a graphical content from the chain that have been constructed with the grammar. We can imagine a moving pen that interprets the terminal symbol as motion [84].

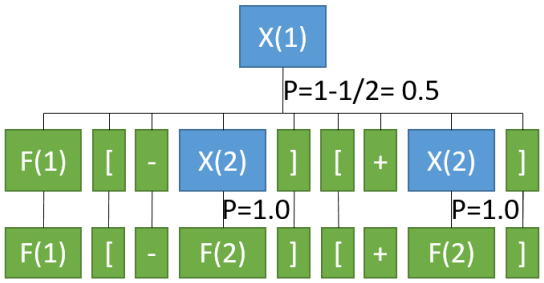


FIGURE 13 – By following the rule of the previous grammar, we can obtain this derivation, terminal are shown in green and non-terminal in blue. The probability of each step is written on the line.

Grammars were introduced into the graphics community by Prusinkiewicz [84]. Early techniques were tailored to represent models of plants [86]. Using only deterministic L-System was not enough to represent the diversity of shapes that could be generated. Thus, the concept has been extended to stochastic and parametric versions that are now widely adopted. Later techniques incorporate constraints from the surrounding environment [85]. Some techniques were dedicated to the generation of cities [81]. Wonka et al. introduced split grammars to produce architectural models [131]. Grammars have also been used to represent ornamental structures [130]. The derivation and the parameter of a L-system can be controlled with a probabilistic model to optimize an objective [111].

The question of defining the set of rule remains. Indeed, in most of the case the grammar is handcrafted and it requires a lot of effort and knowledge about the application domain and the desired shapes. It can be defined by the user through high level specification languages or, in most of the case it is defined beforehand. Recent works consider inverse procedural modelling to generate the grammar automatically from given input shapes. I mention some of these techniques in the following section, which considers by-example modelling techniques.

4.1.2 By Example Modelling

By example modelling of shapes consists in using a shape (or a set of shapes) to produce a variety of novel shapes. The early work of Funkhouser et al. [34] is a great example is this methodology : the work introduces a modelling tool that is able to cut and mix different shapes, and also gives the possibility to search for similar shape (sub)parts using similarities metrics.

Inverse Procedural modelling represents a set of techniques that generate a procedural model (grammar) from existing content. Stava et al. [123] shows a method to automatically build L-System from 2D shapes. The analysis relies on self-similarities detection techniques [71, 12]. This is a vast domain of research, interested reader can refer to the SIGGRAPH course by Aliaga and colleagues [2].

Part-based modelling relies on the decomposition and assembly of a large dataset of models. Whereas the segmentation is mandatory, it is often done manually by the users [34]. Kreavoy et al.[53] present a technique to segment the input mesh into a meaningful decomposition. The user can assemble the different parts through an interface that presents the most probable part to place next [17], but the synthesis can also be done automatically. Zheng et al. [142] present a method that uses symmetry information to mix different parts of different models together. Kalogerakis et al. [50] present a method to obtain probabilistic model by segmenting and analysing shapes from the same domain. While inverse procedural modeling relies on the study of one model to generate the grammar, their technique uses more models allowing to generate more varied content.

Texture synthesis The synthesis of textures from examples is also a typical problem in computer graphics. The purpose of those techniques is to generate large non-repetitive images of materials (wood, rocks, carpet) from small samples. The generated image has to be similar to the example such that a human observer thinks it represents the same material. The interested reader can refer to the survey by Wei and colleagues [128] on this extensively researched topic. Similar techniques have been proposed by Perlin and Hoffert [82] to synthesize volumes with a procedural method or to produce variations in geometry across a surface [11]. Merrel and Manocha [70] use the adjacency information of the edges, the faces and vertices of a 3D mesh to generate various shapes. More recent work focuses on microstructures generation to change the physical properties of a fabricated object [64].

Many approaches have been proposed to generate virtual content. Those techniques initially targeted video games and cinema industries, and need to be adapted if they are used to produce fabricable designs. Indeed, the fabrication constraints and the user intents needs to be integrated into the synthesis.

Integrating the fabrication aspect in methods for modelling by example has been studied. For example, Yang et al. developed [138] a technique that changes a shape such that it will look as if it has been fabricated with a different material than the original shape (e.g. reforming a chair in wood to show how it will look in wrought iron). Schulz

et al. [94] extended by-example modelling [34] with information about fabrication (in particular connectors) in order to produce fabricable models. Shugrina et al. [97] presented a technique to maintain the validity and ensure that a parametric design customized by a novice user is fabricable. AutoConnect [52] synthesizes a 3D printable connector between two objects.

4.1.3 Synthesizing from functional specification

There exist other synthesizing techniques that create a shape by optimizing an objective function.

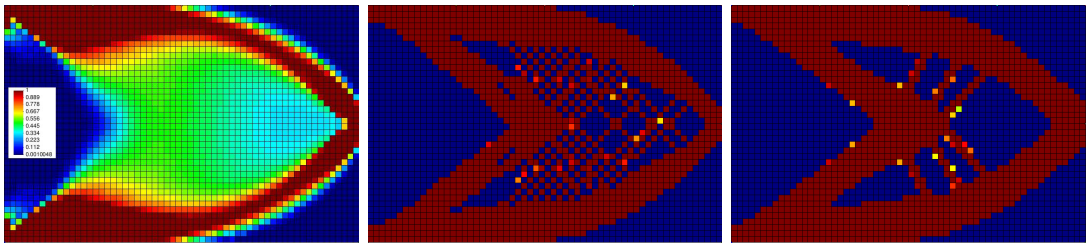


FIGURE 14 – Topology Optimization in 2D. The user determined anchor on the left top and bottom corner, and a force on the right edge. The colour shows the material density (0 : blue 1 : red) From left to right : without penalization, without filtering, with both.

Topology Optimization [7, 98] typically synthesizes a shape by maximizing its rigidity (other objectives are possible). Given the boundary condition and a set of loads it computes the shape with minimal compliance energy. The Solid Isotropic Material with Penalization (SIMP) approach formulates the problem as a material distribution problem. The domain is discretized into a regular grid of elastic elements with varying densities in $[p_{min}, 1]$. The density of each element is optimized with a gradient descent on the compliance energy. Unfortunately this gradient descent is not straightforward. The density takes value between $[p_{min}, 1]$, which have no real signification because we cannot fabricate composite material on most technologies (see Figure 4.1.3, left). To avoid this, p is replaced by p^3 in the optimization. Doing this produces a checkerboard pattern (see Figure 4.1.3, center). This pattern is problematic because it appears independently of the size of the element. However, it can be filtered by using the average gradient of the neighbourhood of each element (see Figure 4.1.3, right). The interested reader can refer to the review by Sigmund and Maute [99].

Topology optimization is a great tool but it is hard to combine it with a desired appearance. It is also time consuming because it requires the results of the FEM simulation to compute the gradients, at each iteration of the optimizer. Those limitations hinder the participation of the user in the design process. She only determines the boundary conditions of the problem and waits for the result. Topology optimization has been used a lot recently in graphics community [132, 22, 65].

It is not easy for a user to drive a system that fully automatically synthesizes shapes. The synthesis can be constrained, and additional objectives added, but the system often

proposes a single solution. This particular answer might not suit exactly the user intent. In this case, one alternative is to rely on shape enhancement techniques to refine a base shape provided by the user. I present sch approaches next.

5 Shape Enhancement

The boundary between shape enhancement and shape synthesis is not strict. I call "shape enhancement" the set of techniques that require partial information about a shape to work. If the user knows what parts of the shape looks like, she will not use shape synthesis techniques because they are not easily constrained by partial – but precise – information. In this case shape enhancement is better suited. The shape emerges from lower level partial specifications (e.g. a mesh, a drawing, a set of 2D curves) as opposed to synthesis technique that require high level specifications (e.g. loads in space, grammar describing a space of shapes, etc.). Lower level specification can also be used to guide a grammar : Nishida et al. [78] use sketches to determine derivation of a grammar, showing that low level specification can guide a synthesis system.

Automatically enhancing a design has been studied in computer graphics in a wide range of application to make designer life easier. A classical example is inpainting [9], that reconstructs a deteriorated image. There are also work on geometry cloning [110] that allows the user to clone the detail of a surface of a 3D model on another. We can also cite sketch-based modelling tools that use sketch to produce 3D models [27, 136]. The interested reader can refer to the survey by Olsen et al. [79].

A first approach to design enhancement is to start from an existing geometry and modify it through deformation or other geometric operations to achieve the user intent.

When the 3d model is given by the user, it can be deformed to match the real world constraints. A volume can be optimized to balance it, by carving inside and deforming its surface [83]. The optimized volume remains 3D printable. A volume can be deformed to optimize rotational properties and inertia [6] or the buoyancy of a fabricated part [125]. Zhou et al. [144] introduced boxelization which is a method to transform an object in a box by folding the different parts of the model. The object can be fabricated and transformed from a shape to the other.

User involvement Shape enhancement techniques allows more user involvement than automatic synthesis techniques, and that makes the process more controllable. The user can be involved in the design process to different degrees or at different times. In some techniques he is guided in the exploration of the design space.

The user might be involved at the beginning of the optimization. She specified entirely the input of the algorithm who generate the model [125]. She might be able to interact with the final model to deform it [83]. The solution can be generated in real time as she is modifying the initial model through an interface [25]. Ion et al. introduced a specialized 3D editor to allow user to create mechanical object with metamaterial [47].

Focus While applicable to many different domains, I noticed two cases where shape enhancement techniques are particularly relevant. The first is mechanism design : desi-

gning mechanism for additive manufacturing is a complex task that requires knowledge in mechanics, 3D printing and proficiency with modeling tools. Simplifying the design stage is important. The second domain is furniture design. Furniture is ubiquitous in our lives, and there are many case where one would like to design a custom furniture. Yet, designing durable, fabricable, low-cost furniture is a difficult modeling task.

5.1 Assisted design of mechanisms

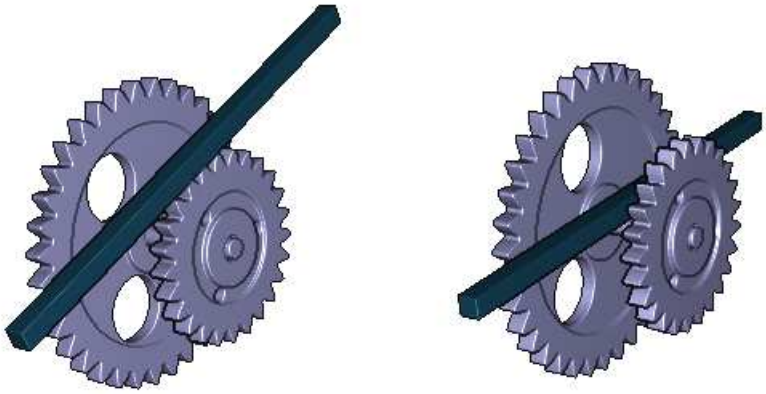


FIGURE 15 – Different Intersection types presented in [25] Left : The green bar is intersecting the small gear. Right : the green bar is intersecting the hinge between the two gears.

In computer graphics and fabrication, the design of mechanical models has been largely studied. Duygu et al. [15] propose a system that can embed an oscillation module in an animated character to reproduce the motion of a motion capture sequence. Calì et al. [13] propose a method that helps the user to add articulations on a Rigged 3D model in order to fabricate an articulated character. Skouras et al. [101] optimize a material distribution and the positions of actuators in a shape to reach deformations specified by the user. Zhu et al. [145] introduce a method that synthesize a mechanical toy that reproduce the motion of an animated input. Through a design system, a user is able to models craftable character with planar motion [68]. Thomaszewski et al. [114] present a system that allows the user to interactively build a moving character from an animated character. Ureta et al. [119] introduce an interactive system that helps the user to create joints between different moving parts.

5.1.1 Mechanical Layout

Computational design of mechanical character [25] presents a method that synthesize a mechanism from a user input. The user interactively animate an articulated character by drawing motion curves for different parts of the model. Then the system generates a

mechanism which motions follow the drawn curves. This generation is done by optimizing the parameters of parametrized small mechanisms that are given as input.

Once the elementary mechanisms are optimized, their relative positions is determined such as to prevent intersections between their respective components. There are two main ways intersections can occur between parts (Figure 15). If two parts are colliding, and if a third part intersects a pin between two other parts. The problem of placing the different parts in different layers is highly combinatorial and a brute force approach is not applicable. The layering problem presented in [25] is solved by a Constraint Satisfaction Problem (CSP). Since their formulation requires the number of layers in the mechanism, the solution they proposed optimizes the number of layer by trying to solve the CSP several times. While the problem as no solution, they increase the number of layer and solve again. The different mechanical parts are integrated in the model and are hidden in a gear box. This algorithm cooperates with the user to design the motion of the mechanism. It requires a bit of artistic bent which may appear as a limitation depending on the target audience.

5.2 Assisted design of furniture

Designing a furniture can be an hard task. Usually, the designer consider aesthetics and rely on a physical simulator to validate the physical properties (e.g. Finite Element Method). Some recent research integrated the physical analysis in the design process. Umetani et al. [117] propose a method where the user participates in the design process and the interface helps her to model feasible furniture by analyzing the physical validity of the design. This is defined by the durability of the joints between the different parts of the planks and the stability of the model. Lau et al. [57] convert a 3d model of furniture in fabricable part and connectors. The system decomposes a voxel representation of the 3D model. It parses this decomposition performing a lexical analysis to obtain a graph. The graph is augmented by information about the connection between the different parts. Koo et al. [51] compute a model from high end specifications provided by a designer. The designer places several boxes in space and chose the different interactions between those boxes. Then the system optimizes the parts and joints to generate a fabricable model. Saul et al. [92] present an interface that allows the user to control entirely the process of designing and building a chair. This tool allows users to sketch the planar parts that compose the chair. Li et al. [59] introduce a method to compute foldable furniture from a partitioned input shape.

6 Positioning of my contributions

In my contributions I explored along the cursor from the machine to the user. The next section lists my contributions, from the closest to the machine to the closest to the user.

Clean Color presents a technique that deals with the defects that appear with Fused Filament Fabrication. By changing the paths followed by the print head and optimizing

the part orientation we minimize the surface imperfections. This technique optimizes the process but it does not impact the design. It has been done with my advisor and has been published in Computer Graphics Forum, special issue of Eurographics 2014 [38]. It is presented in the Chapter 1.

2D Fabrication of 2D Mechanism computes a fabricable mechanism from a partial 2D blueprint. It computes the layout of the different parts from the simulation of the mechanism. It helps the user design pre-assembled 3D printed mechanisms by taking most of the fabrication constraints into account. It has been done with my advisor and has been published in Computer Graphics Forum, special issue of Eurographics 2015 [39]. It is presented in the Chapter 2.

Towards Zero Waste Furniture Design presents a technique to optimize parametric models of planar cut designs in order to minimize the amount of wasted material. Unlike previous work, the technique changes the model given by the user – within specified bounds. It is a the boundary between the design stage and the fabrication stage. It has been accepted with major revision at Transactions on Visualization and Computer Graphics. It has been done in collaboration with Bongjin Koo, Niloy Mitra and Sylvain Lefebvre. It is presented in the Chapter 2.

Synthesis of User-Friendly Shelf presents a technique that synthesizes complete shelves from user input. After the synthesis, the user is able to deform the furniture to achieve different purpose (e.g. minimizing material, improving aesthetics). It covers almost all the design stage, while still synthesizing a shape that is easy to manipulate for the user. This work will be submitted early 2017. It has been done in collaboration with Niloy Mitra and Sylvain Lefebvre. It is presented in the Chapter 3.

Contributions

1

Improving the 3D printing process

Introduction

Rapid digital manufacturing holds the promise to let artists and hobbyists create complex, intricate designs. This extends beyond the geometry to the material properties of an object, with the ability to use different materials in different areas, locally changing properties such as color, elasticity, opacity or even conductivity [7]

A major advantage of filament printers is that they can easily be equipped with multiple extruders to print using different materials. However, as I have discussed before in section 2 of the state of the art(see also Figure 6), there are several challenges hindering the fabrication of high quality parts using multiple filaments.

One could argue that high-end industrial printers would be better suited to this task. However, only few processes can change the material properties during printing. Most are limited to colour gradings, such as the ZCorp [8] and MCor [9] printers (the first uses ink to colour the powder before binding, the second creates objects by layering paper and prints a colour pattern on each sheet). The others, such as the Objet Geometries printers [10] or the MultiFab printer [100] print with different materials by projecting droplets of different resins. However, all of these technologies are expensive to acquire and operate, and require special training and facilities (in particular, resins and powders have to be handled with care).

In contrast, filament based printers are easier to handle and assemble/modify, inexpensive, wide spread, and do not require specific precautions beyond a well ventilated area. This could be one of the reasons why the technology blossomed during these past years.

I therefore considered how to improve the quality of multi-material prints on filament printers. I chose to not modify the hardware in any way in order to keep it simple and

7. https://www.proto-pasta.com/pages/conductive-pla
8. http://www.3dsystems.com/3d-printers/professional/projet-460plus
9. http://mcortechnologies.com/3d-printers/
10. http://www.stratasys.com/3d-printers/design-series/objet30-pro

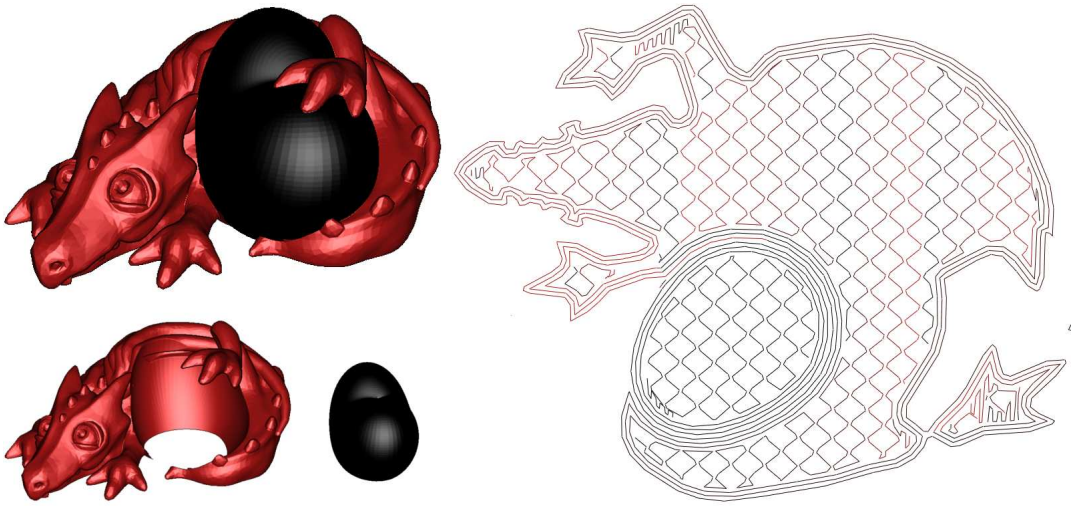


FIGURE 1.1 – A slice of the dragon heart model

affordable. Instead, I focused on novel algorithms for preparing the object and computing the travel paths. Our approach eliminates or strongly reduces most of the defects mentioned in Section 2.1.1 of the state of the art.

This chapter presents a technique that is closer to the printer than to the user : it is a process stage technique and strives to accurately reproduce the input design. For clarity in the remainder of the chapter I often refer to different materials has different 'colors'. It is to be understood, however, that each color could be a different material supplied as a filament.

This work has been done with my advisor and has been published in Computer Graphics Forum, special issue of Eurographics 2014 [38].

1.1 Overview

Our algorithm targets low–cost FDM printers of the RepRap family, equipped with multiple extruders. We focus our explanations on *dual printing* (two extruders), but our approach can be adapted to more extruders. The extruders are mounted on a single carriage at a fixed offset δ (2D vector). We denote by δ_i the offset of extruder i, with $\delta_0 = 0$ and $\delta_1 = \delta$.

The input to the slicing process are two meshes, each describing the volume in space to be filled with the corresponding color/material. An example is given Figure 1.1. We name the meshes $\mathcal{M}_0$, $\mathcal{M}_1$ and refer to the volume enclosed by each as respectively $\mathcal{V}_0$, $\mathcal{V}_1$. We assume non self-intersecting watertight meshes, with $\mathcal{V}_0 \cap \mathcal{V}_1 = \emptyset$.

To understand our approach, let us consider the major defect : strings of plastic deposited by oozing extruders. Strings are deposited in two circumstances, which we will refer to as *Case 1* and *Case 2* throughout the text :

— **Case 1** Travel moves (no extruder prints, all ooze). This is illustrated Figure 1.2, left.

— **Case 2** Print moves (one extruder prints, others ooze). This is illustrated Figure 1.3, left.

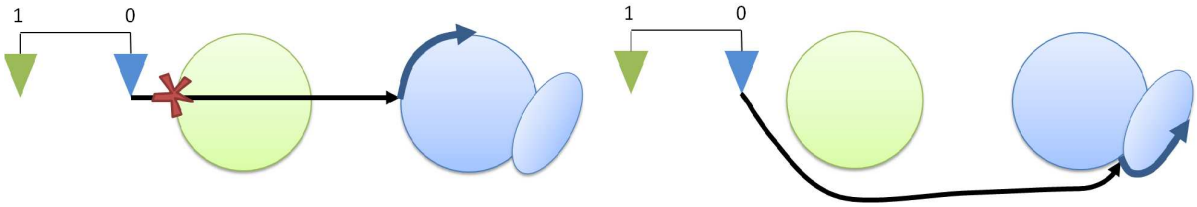


FIGURE 1.2 – Case 1. blue Extruder travels across the green region, depositing ooze.

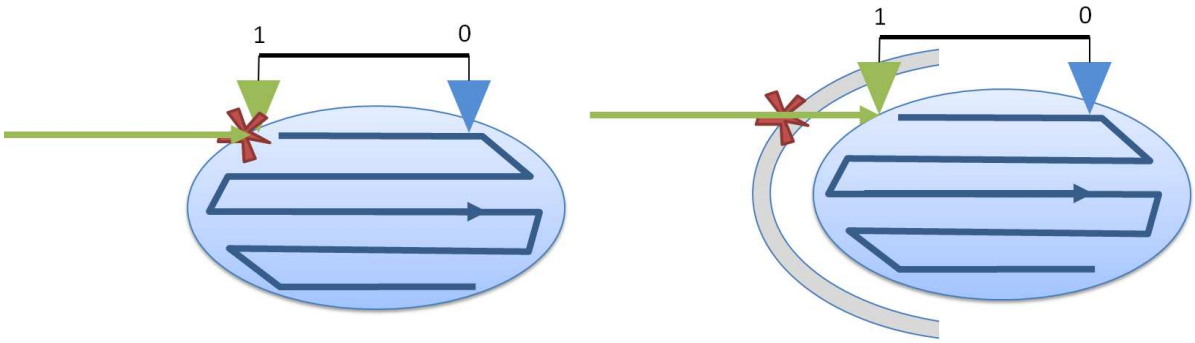


FIGURE 1.3 – Case 2. Blue extruder prints a segment, which makes the green one oozing over the blue region.

Case 1 is addressed by our path planner (Figure 1.2 right, Section 1.4). It navigates around the part while avoiding strings to deposit, finding travel paths where the idle extruder stays away from the print. If no such path can be found, our planner favors string deposition on matching colors. In last resort, it will always favor deposition in low visibility areas.

Case 2 is more constrained : the input printing paths cannot be changed. We address this in two ways. First, we minimize such cases through azimuth optimization (Section 1.2) and second, we create a rampart in close proximity of the shape, wiping extruders before they reach the surface (Figure 1.3 right, Section 1.3).

The planner and the rampart work in conjunction : the planner uses the rampart for circulation around the part. Each time an extruder crosses a rampart wall, it is wiped cleaned of any oozing string. In addition, we exploit the time during which the extruder circulates within the rampart to perform a *refill* : we run plastic through the extruder to refill the hot plastic chamber and keep it ready for printing. This prevents holes from appearing.

1.2 Optimizing print azimuth

For clarity let us consider the case of two extruders separated along the X print axis. We refer to them as the *left* and *right* extruders. Whenever the right extruder prints,

FIGURE 1.4 – The left extruder uses black and the right extruder red. While the red (right) extruder prints the body of the dragon, some black (left) plastic will interfere with the red region. This is accurately predicted by the intersection volume (green). The printed model exhibits severe color smears and strings throughout this volume. Additional defects are due to poor path planning.

the left extruder creates a ghost image of the print shifted by δ on the left. This ghost represents the locus of the points that may suffer color smears if the left extruder starts to ooze. Depending on the part size and azimuth, the ghost image will interfere (intersect) with what the right extruder is printing. Minimizing the size of this region will reduce the chance that smears occur. This is illustrated in Figure 1.4.

We perform the change of azimuth by a rotation around the Z-axis of angle Θ, noted in matrix form as R^Z_Θ in the following. We do not change the vertical orientation of the part (Section).

By reasoning throughout all the print layers, for a given orientation angle Θ the total size of the interference region due to the left extruder is the volume $I_L(\Theta) = \left(R^Z_\Theta \times \mathcal{V}_R\right) \cap \left(T^X_\delta \times R^Z_\Theta \times \mathcal{V}_R\right)$ where T^X_δ is a translation matrix of δ along the X-axis. Similarly, the interference region due to the right extruder is $I_R(\Theta) = \left(R^Z_\Theta \times \mathcal{V}_L\right) \cap \left(T^X_{-\delta} \times R^Z_\Theta \times \mathcal{V}_L\right)$. We therefore search Θ_{min} such that :

$$\Theta_{min} = argmin_\Theta \left(I_R(\Theta) + I_L(\Theta)\right)$$

In general computing the volume of the intersection between two meshes is a difficult problem. We rely on an approximate computation based on boolean mesh operations with dexels [120]. This integrates very well within our slicer which also relies on this principle to extract slices [58]. The best angle is selected after testing for all angles by 10 degree increments.

Figure 1.5 shows an example of optimized azimuth angle. A better azimuth angle strongly reduces the defects due to Case 2. However, they often cannot be entirely suppressed in particular on large parts. The rampart, described next, further reduces these defects.

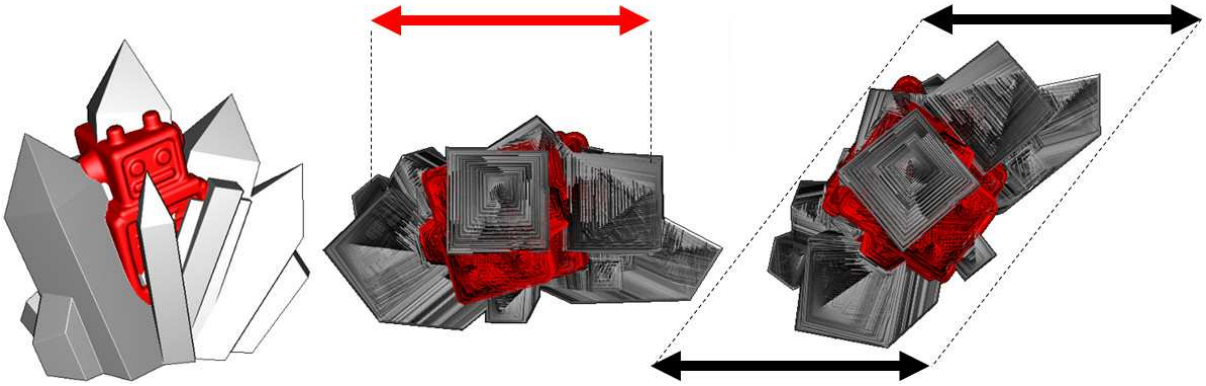


FIGURE 1.5 – Robot-ice (user Ultimaker on Youmagine) : this model is larger than the spacing between the extruders, and its default azimuth results in significant color smears. Our azimuth optimization finds an angle avoiding these entirely.

1.3 Rampart

The rampart is a disposable structure built around the part. Its primary purpose is to catch the oozing strings before they reach the part (Case 2, Figure 1.3). Its secondary purpose is to provide a space where extruders can be refilled with plastic.

We construct the rampart in close proximity of the part. This allows to wipe the extruders clean as close as possible to the surface, and also keeps travel time low by avoiding traveling to a distant wipe station.

Geometry : The outline of the rampart is obtained by rendering the part as seen from above, in black on a white background. We process this silhouette image to only keep the connected component containing the image border (inner holes are removed). The remainder is then offset towards the outside to obtain the contour of the first wall of the rampart. Another offset gives the second wall. We use 2 mm between the part and the first wall, and 4 mm for the inner spacing. These contours are added to each layer at all heights, creating a vertical extrusion of the rampart contours.

Printing : The walls of the rampart are built at the start of each layer. Having the walls reach the tips of the extruders ensures proper wiping. We print the rampart at high speed (120 mm/sec), using a different extruder for each wall. One subtlety is that while a rampart wall is printed, the other extruder may move over the part – risking ooze to deposit. We have to ensure that it will be properly wiped before reaching the part. Our approach is simple : we start printing the wall from a location which guarantees that the other extruder reaches the rampart before the print. On our dual–extruder printer, when the right extruder is used we start printing the rampart from its leftmost point, and vice versa.

1.4 Path planning for multiple colors

The azimuth optimization and the rampart reduce defects due to Case 2, where strings are deposited by the idle extruder while the other prints. We now consider Case 1, where

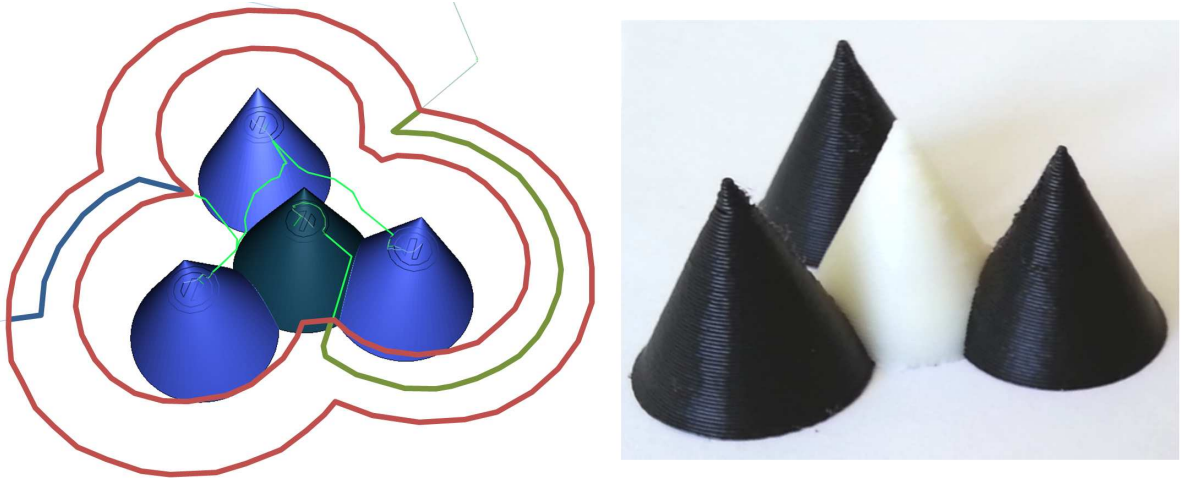


FIGURE 1.6 – *Left :* One slice near the top of the 4–cones model. The red outlines are the walls of the rampart. Lime-green paths are travel moves. The bold paths within the rampart walls show were the extruders are active to refill with plastic. *Right :* Printed model, using our technique.

FIGURE 1.7 – *Left :* The 4–cones model printing. The red arrow shows how much ooze exits the left (white) nozzle while the right (black) nozzle prints. *Right :* The rampart captured a significant amount of stringing. Result shown Figure 1.6.

FIGURE 1.8 – The ambient occlusion computed on the kitten model. Thingiverse, thing 12694, user MBCook

both extruders are idle and traveling.

The path planner chooses in which order the print paths are visited, the point where to start printing each, and the travel path for moving from a print path to the next. We design our planner under the worst case assumption that idle extruders are always oozing. Our strategy is therefore to always avoid having the idle extruders cross over the part. In cases where crossing over the part is mandatory, we hide defects in low visibility regions. In addition our path planner exploits the rampart, circulating within its double walls. This encourages wiping and reduces the quantity of oozing : even if the idle extruder crosses over the part, the defect will be minimal. Each time an extruder travels within the rampart we perform a refill of its hot plastic chamber, ensuring that it is always ready to print.

1.4.1 Overview

After slicing, we obtain a set of 2D slices, generated from the input 3D models. In our slicer each slice is a set of paths, tagged with an extruder id and a type. The path type can be either of perimeter, shell, or infill (Figure 1.1). Recall that most print paths are cyclic and therefore we can freely choose their start/end point when printing.

Conceptually our algorithm works in high resolution images representing the slice. We set the resolution to 0.05 mm per pixel, which for a nozzle of size 0.4 mm covers 8×8

pixels. This offers enough resolution for detailed prints. Our slicer generate paths by contour tracing the slice images ; the paths are therefore sampled at the pixel resolution.

In addition to the slices we also consider the visibility volume computed from the surface, using ambient occlusion (AO) – an example is shown in figure 1.8. This sparse 3D grid defines in every surface point a visibility coefficient. This determines whether a point along a perimeter is highly visible. We do not describe here the computation of the visibility volume and refer the reader to the survey on ambient occlusion techniques by Méndez-Feliu et al. [69]. In this volume, at any point in space we store a value between 0 and 1 ; 1 being the value for the most visible point.

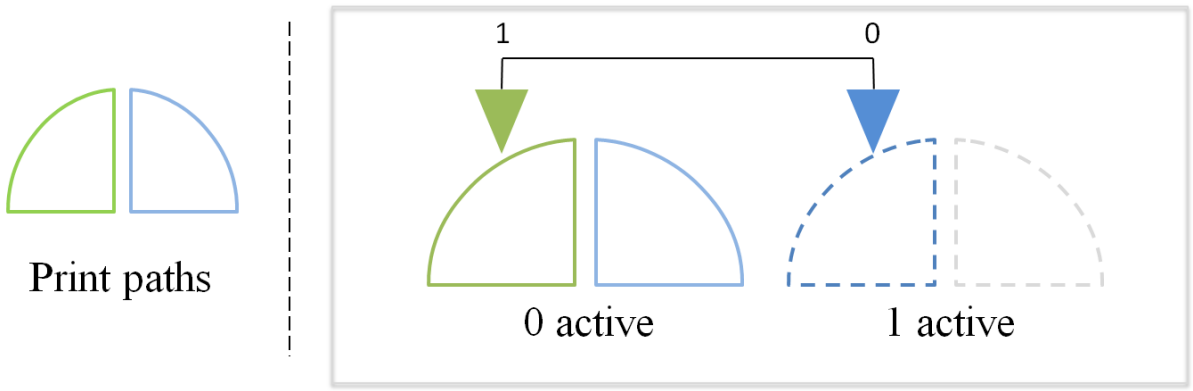


FIGURE 1.9 – *Left :* Print paths for both extruders. *Right :* During printing, we consider the motion of extruder 0 only. The plain and dashed blue lines reveal its trajectory. However, when printing in green extruder 1 is activated, depositing plastic at the correct location (plain green line).

The toolpath planning problem is a constrained form of TSP (See Section 2 of the state of the art). Solving directly with a TSP heuristic poses several difficulties. First, we need to consider a large number of nodes : paths have to be finely sampled to properly take into account visibility. Second, TSP is a global compromise between the order with which paths are printed and the navigation between them, while we want to favour quality over print time (within reasonable bounds) : navigation is of primary importance. Therefore we divide the problem in two distinct steps :

1. Ordering of paths, described Subsection 1.4.3 ;
2. Navigation between paths, described Subsection 1.4.4.

The ordering step decides the order in which print paths will be visited. The navigation is solved in a second step, maximizing the quality while taking into account the travel time. The benefit of resolving ordering first is to let us formulate the navigation optimization as a shortest path search in a graph, obtaining a high quality solution.

After these two steps we obtain a complete set of toolpaths, including travel paths that avoid or hide print defects.

1.4.2 Handling of multiple extruders

Our path planner always keeps track of the position of all extruders. However, all coordinates are rewritten using extruder 0 as the reference. The print paths for the other extruders are therefore translated into the frame of reference of extruder 0, as illustrated in Figure 1.9. When printing with extruder 1, the carriage will move along a trajectory defined for extruder 0, but extruder 1 will be extruding plastic at the correct location.

1.4.3 Ordering of perimeters

Our ordering step makes sure that paths are visited in reverse order of inclusion, from the inner-most towards the exterior of the slice. This ordering avoids an extruder to have to re-enter a region whose perimeter was just printed.

Since some regions can enclose multiple child regions, we also have some degrees of freedom in the ordering. We exploit these to further minimize the risk of defects, in particular holes due to missing plastic.

Inclusion tree. Most of the paths, with the exception of infills, are cycles which follow the hull of the surface. We organize the slice into zones, where each zone is a region separated by a print path. This is illustrated Figure 1.10, left.

The notion of zone is well defined since, by construction, paths do not intersect : they capture the contours of the intersection of a plane with a well defined interior-exterior mesh. Infill paths are generated by the slicer only within the inner parts. In case of thin-features paths may come in contact with each other, but do not cross.

Similarly to prior work, we build a tree representing the zones with the most exterior zone at the root (Figure 1.10, right). The tree contains two types of nodes : zone nodes and path nodes. Each parent zone node is separated from a child zone node by a path node, indicating which cycle separates both zones in the slice. The infill paths are attached to the zone representing the inside region they are filling. The tree is quickly built with a flood-filling algorithm in the sparse image of the slice, tracking which paths are neighbouring each zone.

Ordering children in a zone. Zones of the inclusion tree may have multiple child paths (Figure 1.10, right). We can freely choose in which order these children are printed.

Their ordering is not without impact on quality. In particular, the risk of holes can be reduced by printing long paths first : any lack of plastic is more likely to go unnoticed. Indeed, the inner–most paths are infills and shells (and therefore invisible). Note that lack of plastic is a rare event since our path planner encourages the use of the rampart, where refills occur.

We therefore order the children by decreasing path length. Once the children of all zones have been ordered, we traverse the tree in post–order, gathering all paths. For the example Figure 1.10, the final print order will be *F0,F1,F2,D,B,C,A*.

1.4.4 Navigation

After the order is determined we obtain a sequence of paths to be printed. We seek to generate travel paths in between. Each will navigate from one print path to the next, minimizing defects, using the rampart for circulation, and maintaining the print time low whenever possible.

We optimize for navigation by a Dijkstra algorithm in a *navigation graph* comprising the paths to be printed, as well as a number of additional navigation pathways. The Dijkstra algorithm traverses the graph from one print path to the next, constrained by the order computed previously. Note that it computes a single navigation path and therefore

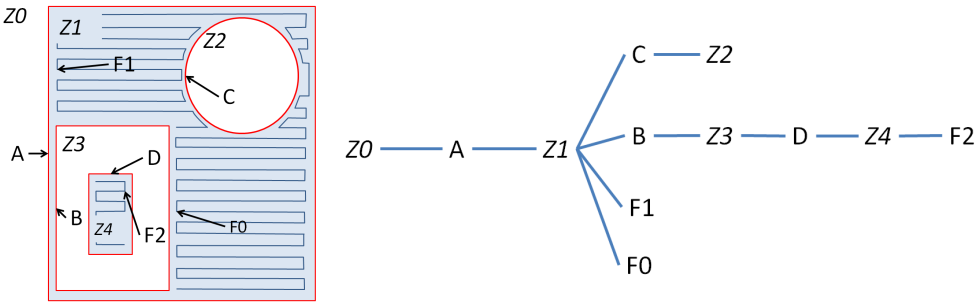


FIGURE 1.10 – *Left :* A slice showing print paths (A–D), infill paths (F0–F2) and zones (Z0-Z4). Each zone is a different region of the slice plane, with Z0 the outermost zone. *Right :* The inclusion tree captures inclusion relationships. Note how B, C, F0, F1 are children of the same zone Z1.

solves for the global navigation problem – as opposed to solving for each in-between travel path independently.

The optimized navigation path is only concerned with travel : it is enough to reach the start of a print path and resume from its end, or to reach any single point along a cyclic print path – the extruder resumes from the same location after printing the cycle.

Navigation graph. The navigation graph has a sparse set of nodes which are located at integer coordinates in a grid. Multiple nodes can occupy a same coordinate. The resolution of the grid matches the resolution of the slice, with a larger extent to take into account the motion of all extruders. A node in the graph represents the reference extruder positioned at this location. Nodes are connected through edges representing segments in the plane. The cost of edges takes into account the defects that can occur when traveling this edge, considering the position of all extruders.

The construction of the graph is described in the next paragraphs. An example graph is shown Figure 1.11.

Print paths. We insert in the graph the paths along which plastic has to be deposited : cyclic paths are rasterized into the grid, adding nodes and edges between neighbors. We only add the start/end points of open paths.

Navigation paths. It would be wasteful to add one node per grid coordinate in the graph for navigation : once optimized, travel paths use only a small set of optimal pathways to navigate around different regions of the part. To speed up computations we explicitly select a small number of pathways and connect them to form the complete navigation graph.

We form the navigation pathways by adding to the graph a set of nodes $\mathcal{N}$. We start by inserting the nodes corresponding to multiple versions of the print paths, each translated in the reference frame of one extruder (dashed lines in Figure 1.9).

We call these the *ghost paths*. Even though they are navigation paths they generally incur a large cost since they correspond to cases where an extruder passes over a mismat-

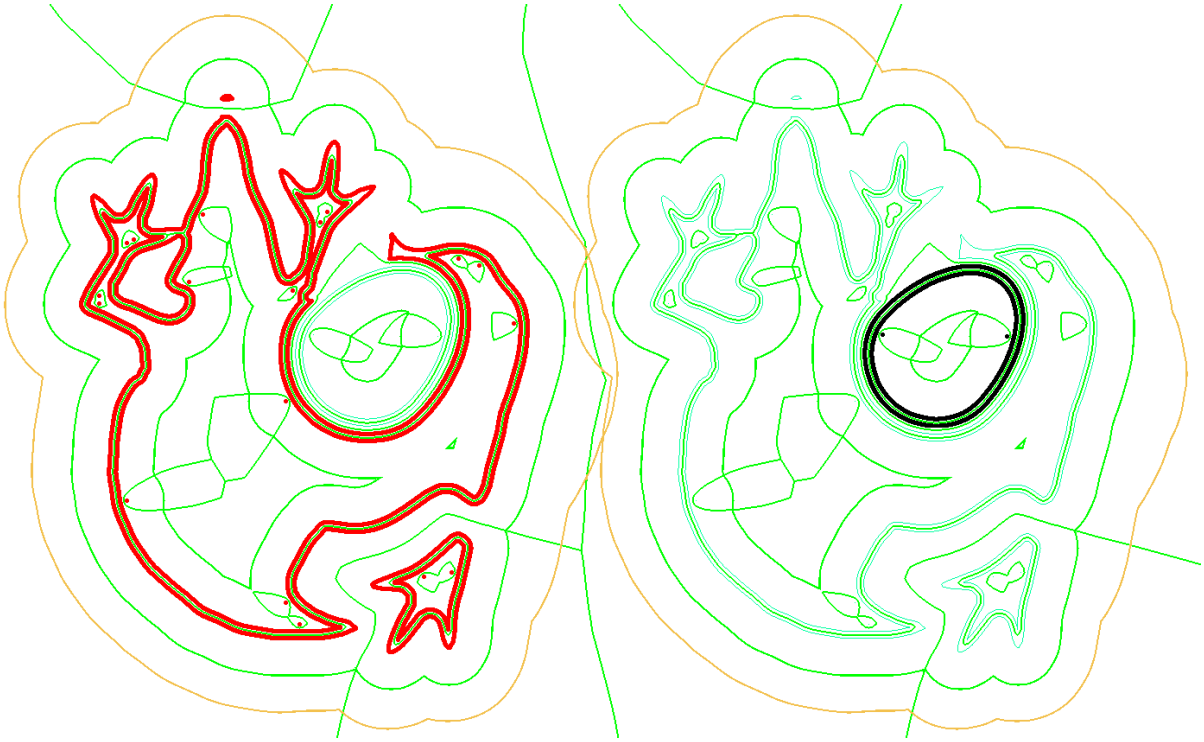


FIGURE 1.11 – The graph nodes of a slice of the dragon–heart model, for a dual left/right extruder printer. The right (red) extruder is the reference. Green paths are navigation pathways, the orange paths circulate within the rampart. The bold colored paths are the location of the reference extruder while printing ; their color indicates which extruder is activated. Note the red/black dots corresponding to the start/end points of (open) infill paths.

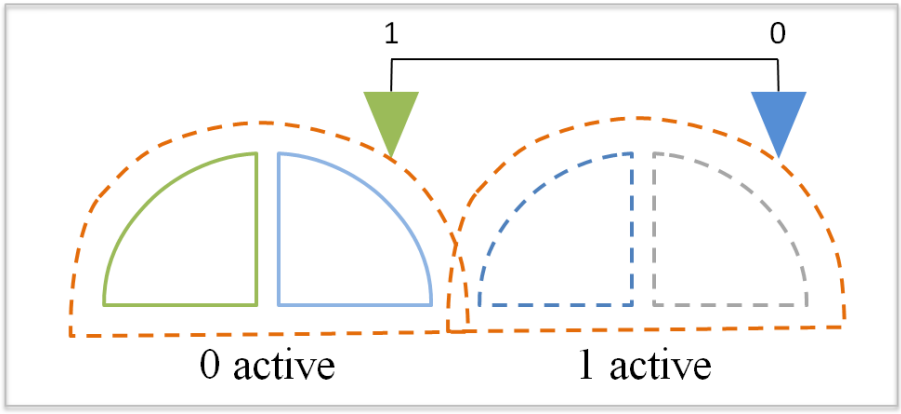


FIGURE 1.12 – The orange dashed lines are navigation paths added around the print and ghost paths. The ghosts are necessary to generate the navigation paths allowing extruder 0 to travel while extruder 1 avoids crossing the print.

ching color. They serve their main purpose at the next stage, when constructing additional navigation paths going around print and ghost paths as illustrated Figure 1.12.

We next compute a distance field in the grid [26], from all path nodes added so far (print and ghost). The distance field indicates for each empty cell at coordinate e in the graph grid, the vector $\mathbf{d}(e)$ to the closest cell occupied by a node $n = G(\mathbf{d}(e))$, with G the graph grid.

Using the distance field, we select a number of additional nodes : We form go-around pathways by selecting nodes at a fixed distance in the distance field (we use 4 times the nozzle width). We also form pathways between print paths, selecting the medial axis of the distance field – that is, selecting the nodes on either side of the medial axis which is located along integer cell edges. Finally, we add nodes for the path enclosed within the rampart, again in multiple versions translated in the frame of reference of each extruder. This provides a circulation around the part regardless of which extruder is active.
Figure 1.11 shows the nodes of a complete navigation graph.

Edges. We next add edges in the graph. Edges are always added in both directions ($a \to b$ and $b \to a$).

First, all nodes at neighboring integer coordinates are connected. Second, we connect the navigation nodes $\mathcal{N}$ to the rest of the graph, adding edges $(n, G(\mathbf{d}(n)))$, with $n \in \mathcal{N}$. Finally, the navigation nodes are connected together : For each node $n \in \mathcal{N}$ we gather the set of nodes $\mathcal{C}(n) = \{m \in \mathcal{N} | n \neq m, G(\mathbf{d}(n)) = G(\mathbf{d}(m))\}$. These are the navigation nodes having the same closest node as n. We then add all edges $\{(n, m) | m \in \mathcal{C}(n), \mathbf{d}(n) \cdot \mathbf{d}(m) > 0.5\}$, where "$\cdot$" denotes the scalar product. This connects n to the other navigation nodes that are on the same side of the node $G(\mathbf{d}(n))$, within a tolerance.

Edge costs. The cost of following an edge between nodes $p \to q$ in the graph is :

$$c(p \to q) = W_d * d(p, q) + T(p \to q)$$

with $d(p, q)$ the Euclidean distance between the grid coordinates of p and q normalized by the largest extent of the slice, and W_d a weight allowing to trade between travel time and quality (we use $W_d = 0.1$). The term T is computed as :

$$T(p \to q) = \sum_{i}^{E} \left(M_i(q + \delta_i) \cdot ao(q + \delta_i) + Z(p + \delta_i \to q + \delta_i) \right)$$

where $M_i(q + \delta_i)$ checks in the slice image whether at position $q + \delta_i$ the extruder number i is above a mismatching color. If that is the case, then $M_i(q + \delta_i) = 10$ and 0 otherwise. $ao(x)$ returns the value at coordinate x in the AO grid. In case no value is available at x we search for the closest value in a small neighborhood (nozzle width), and return 0 if none is found. The $ao(q+\delta_i)$ factor thus modulates the penalty in function of the visibility of the point.

$Z(u \to v)$ takes into account edges entering or exiting print paths. These are the locations where zippers may occur and strings may deposit. Therefore, it is defined differently depending on the type of edge :

$$Z(u \to v) = \begin{cases} 0 & \text{if the edge } u \to v \text{ does not exist} \\ ao(u) & \text{if u on print path, v on navigation} \\ ao(v) & \text{if v on print path, u on navigation} \\ 0 & \text{otherwise} \end{cases}$$

Our edge cost favors travel paths avoiding strings to deposit : the term M_i strongly penalizes any crossing of an extruder above a print path of a different color. In all other cases the term $Z(u \to v)$ ensures that defects (zippers, strings) will be located in low visibility areas, as given by ambient occlusion.

Edges belonging the rampart pathway have a smaller cost (10 times smaller) to encourage their use.

Shortest path optimization. We solve navigation by searching for the shortest path from the start point of the slice, through all print paths, to a point of the last print path. We select the start point from the end point of the previous slice, or the origin for the first slice. The path is constrained to traverse the print paths in the order defined by the ordering step. Open paths are visited as soon as their first point is reached, and the shortest path search resumes directly from their last point. Cyclic paths can only be considered visited when all their nodes have received their shortest path cost.

We backtrack the overall shortest path from the node of the last perimeter having smallest cost. The shortest path goes through one node of each cyclic print paths and the first/last node of non–cyclic paths. The sequences of nodes between each print path node form travelling paths. We insert them into the initial sequence of paths to obtain the final sequence of paths. These are then used to produce the instructions for the printer (G-code).

1.4.5 Triggering refill

The path planner generates travel paths that tend to circulate within the walls of the rampart. After path planning, we follow each travel path and detect when it enters/exits

the rampart. We then activate the extruder which circulates the rampart, allowing plastic to flow. This is illustrated in Figure 1.6.

Our goal is to run a sufficient volume of plastic through the nozzle to properly refill the hot chamber. Since the travel path within the rampart is already determined, we can only vary the travel speed or the extrusion speed. We use a fixed extrusion speed ensuring reliability. We therefore adapt the travel speed. Given a path length L (mm), a volume to push v (mm^3), a filament extrusion speed e (mm/s), and a filament diameter f (mm), we compute the travel speed as :

$$s = \frac{L \cdot e \cdot \pi \cdot f^2}{4 \cdot V}$$

For mechanical safety we ensure the speed remains below a maximum $(120mm/s)$. This guarantees that enough plastic is pushed, regardless of the length of the segment within the rampart. We use $v = 2mm^3$ and $e = 2mm/s$.

While it is possible for several refill paths to overlap, we did not find that to be a source of concern during printing – however we disable refill on the first few layers where the nozzle is close to the print bed.

FIGURE 1.13 – Our result (right) has almost no defects compared to the same model printed with Skeinforge/ReplicatorG (left). Pictures focus on the worst regions of the surface. Both models use the optimized azimuth angle.

1.5 Results

We implement our method in our slicer [58]. We print on a Replicator 1 dual with Sailfish 7.4, using ABS plastic.

Our prints are all obtained with same values for all parameters. Our parameters are 30 mm/s for perimeters, 60 mm/s for other print paths, 120 mm/s travel, 20% infill and 2 shells, 0.5mm prime/deprime. All the results shown in this chapter are exactly as they came out of the printer : we do not clean them in any way.

Figure 1.20 shows how zippers are hidden in low visibility regions by the edge cost defined Section 1.4.4. This benefits both single and dual color prints.

Figure 1.16 summarizes the performance of our approach. Timings for processing include all steps (azimuth, path planning, AO). Timings are measured on an Intel I7 4770, 3.4 GHz equipped with a GeForce GTX 770. We process multiple slices in parallel (8 threads). Note that the relative overhead in filament length becomes smaller on larger objects or with denser infills (here we use only 20% infill).

The Weight of Plastic The weight of plastic depends of the shape of the support but also of the surface that require supports. To make a fair comparison of the weight of plastic, we need to apply the methods that generate the supports on the same surfaces. If the said surface are not the same for all algorithms, one might out performed the others if it detect less surface to support.

The Print Time Comparing the print time is tricky. We need to compare it for the same surface to support and sliced by the same software. However it is not always possible since the support generation is often embedded with the slicing process.

Some algorithm integrate the user in the loop for the generation of support and we have to compare them to fully automatic method.

1.5.1 Multiple color prints

We first test two challenging models. *Dragon–heart* has a significant imbalance between both extruders, implying that one is idle for long periods of time. The body is larger than the extruder spacing in all directions. The dragon is printed at 0.2 mm layer height. *Two–color–world* combines fine, detailed features with a circumference which is larger than the extruder spacing. We print it in black and natural plastic at 0.3 mm layer height. The later being translucent, any color smears end up being visible. Nevertheless, we reach high print quality with crisp color separation as shown Figure 1.19. This is to be compared with ReplicatorG/Skeinforge as shown in Figure 1.18.

Figure 1.13 compares dragon–heart printed with ReplicatorG/Skeinforge and our technique, both using our azimuth angle optimization.

Figure 1.17 shows the Robot–ice model printed in opaque red within a crust of translucent natural white ABS, at 0.3 mm layer height. As shown by the back–lit model our approach produces a clean output within the volume itself.

Figure 1.15 compares the result of progressively enabling each component of our technique (none, rampart only, full) versus the result of Makerware 2.4. The model fits within

the extruder spacing, so all defects are due to Case 1. For this comparison we match the settings of Makerware and use 150 mm/s for the travel speed, 40 mm/s for the perimeters and 90 mm/s for all other paths. Layer height is 0.3 mm. The print time for our full technique is 46 min versus 24 min without, using respectively 4.6 and 2.3 meters of filament. The overhead goes down on larger prints (see Figure 1.16). Makerware 2.4 prints in 39 min using 3.91 meters of filament. Some differences with Makerware are explained by the different approaches used for slicing, in particular regarding the thin petals ; however color smears are clearly visible in all but our result.

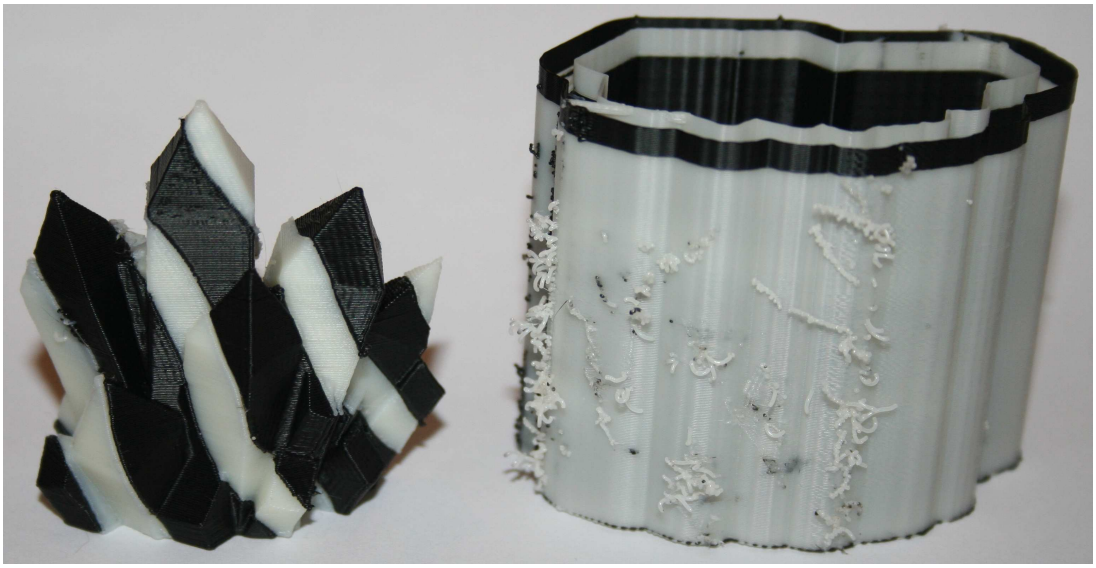


FIGURE 1.14 – A black and white crystal and its rampart. Note the significant amount of stringing captured on the rampart. Small defects are due to thin color slabs not adhering.

1.5.2 Limitations and future work

The rampart wastes a small amount of plastic – to be compared with the cost of failed or low–quality prints. Plastic is also easily recycled in filament (see for instance the *Filabot*). Printing the rampart may involve crossing over the print, risking color smears. This is visible at close inspection on the right side of the Robot–ice, Figure 1.17.

While we strongly minimize the amount of defects, some color smears may still be visible when one extruder prints for a long time and, due to the object geometry, misses the wipe. Vertical complexity also increases the rampart-surface distance which reduces its efficiency. A direction of future work is to further optimize the shape of the rampart to perform more wipes. Our ordering step could be improved to take print time into account. Our edge cost could also easily incorporate additional terms, for instance hiding zippers in high curvatures of the surface.

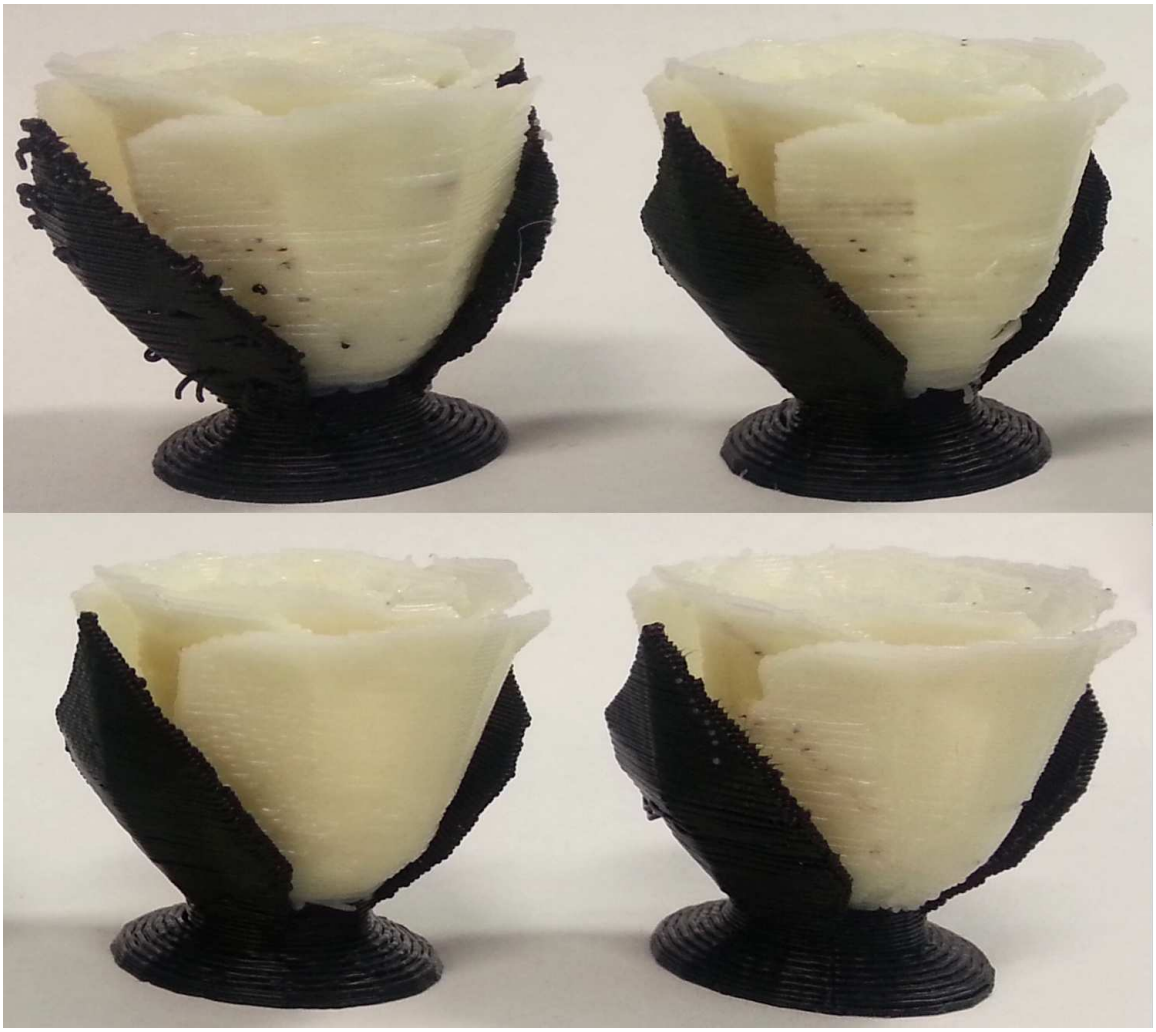


FIGURE 1.15 – Quality comparison. *Top left :* Print without our approach. *Top right :* Print with only the rampart (no navigation, no refill). *Bottom left :* Our final result. *Bottom right :* Print from Makerware 2.4. Rose by user Jillian (Thingiverse)

Model	Processing	Print time	Filament
World	4m22s	2h20m [1h32m]	12.3 [8.3]
Dragon	4m25s	1h51m [1h16m]	7.50 [4.3]
Robot	8m40s	1h53m [1h06m]	9.91 [4.9]

FIGURE 1.16 – Timings for processing, printing and length of plastic filament used (meters). Numbers in brackets are timings/lengths without using our technique.

1.6 Later improvement to the implementation

Since the publication of the work, we decided that the computation time was too important for its integration in the IceSL software developed within the team. In single color print, it does not require that much optimization to perform good results based on the geometry of the model. We decided to not use the information based on computation

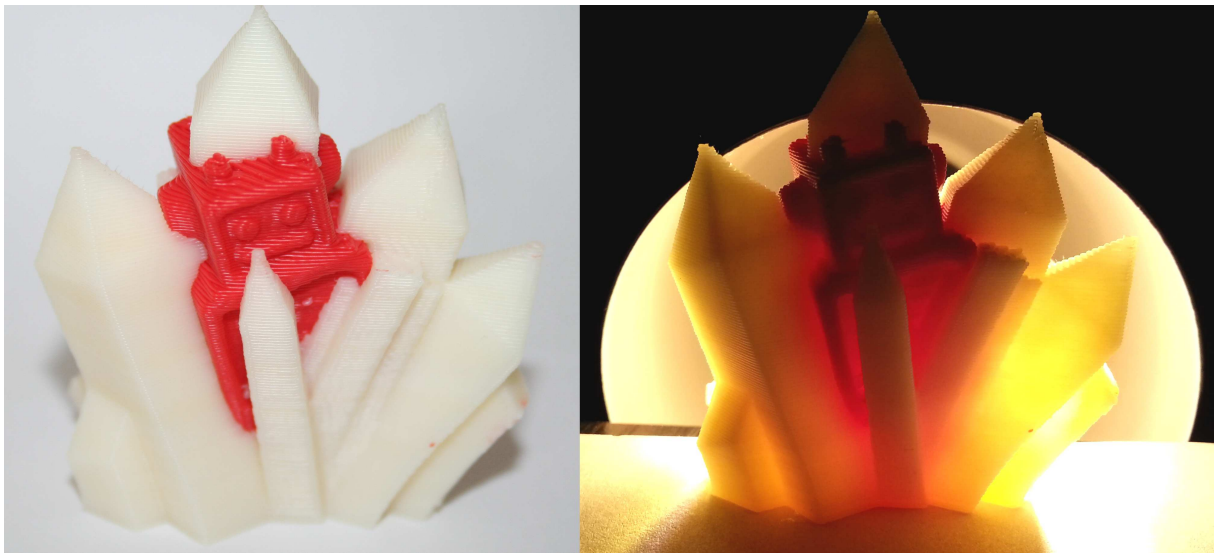


FIGURE 1.17 – Robot–ice printed with our technique. The backlit model reveals the absence of smears within the volume.

FIGURE 1.18 – *From left to right :* Two–color–world model, printed with ReplicatorG/Skeinforge, our technique without azimuth optimization and our complete technique.

of the ambient occlusion and to simplify the path planning in order to decrease both print and computation time.

Instead of exploiting ambient occlusion information to determine the starting point of each perimeter paths, we now rely on the curvature of the paths. Points on high curvature regions are favoured as starting points, with a higher priority given to the ones in concave regions (low visibility area). This hides zippers well for paths with salient angles – as before the zippers are not really hidden, but their visual impact on the surface of the model is reduced.

The rampart motivated future work by colleagues. Hornus et al. [42] developed a technique that generate rampart that are printable, and as close as possible to the object.

FIGURE 1.19 – Our result seen from different angles.

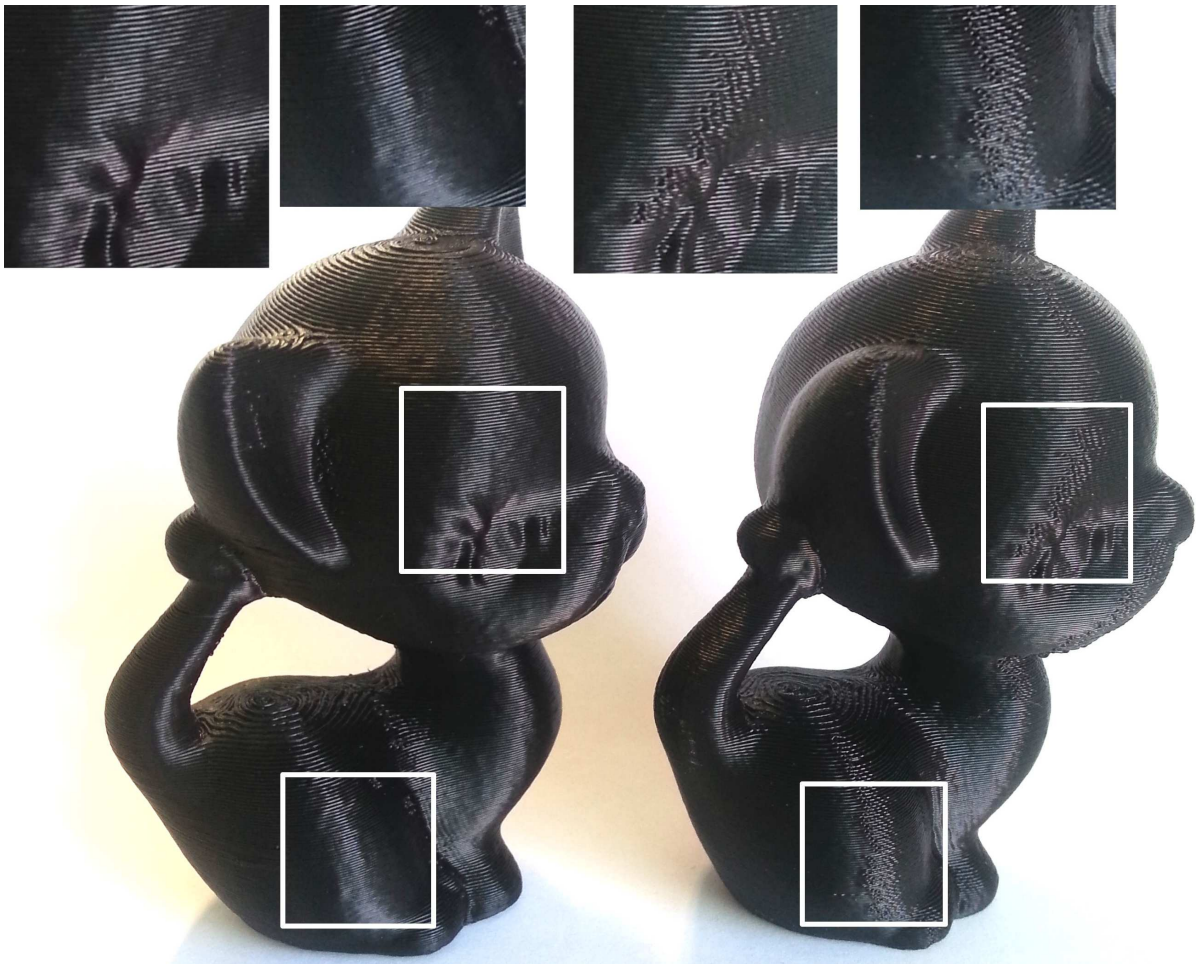


FIGURE 1.20 – *Left :* Our result, zippers are hidden in low visibility regions. *Right :* Skeinforge with jitter plugin.

1.7 Other contribution to the printing process

This section describe my contribution to other projects linked to the printing process. In particular, it presents an algorithm that detects points that need supports, and a

guideline to compare slicers. The first part is part of a collaboration with Jérémie Dumas and Sylvain Lefebvre. It has been published in Transaction of Graphics, special issue of SIGGRAPH 2014 [30].

1.7.1 Detection of support points

Before computing the geometry of the support structure, we need to determine a set of points that require support(See Figure 1.7.1). Our approach starts by slicing the model without any support, determining the full set of *print paths*. Each print path is a sequence of segments along which plastic has to be deposited. The print paths are of two types : the *perimeters* which follow the outer boundary of the surface and the *infills* which fill the interior of the volume.

Our algorithm walks along each print path and checks whether each segment endpoint is properly supported by the layer just below. The test considers the coverage of a disk having for diameter the size of the print nozzle (0.4mm in our setup). If more than 50% of the disk lies outside the object on the layer below, we consider the endpoint unsupported — this threshold was determined experimentally to ensure good surface quality. In practice, we implement the test using images of the layers with 0.05mm pixel resolution. The algorithm 1 describe this step. An simple example of how the algorithm works is given figure 1.7.1. We only check segment end points since the segments themselves form bridges if their extremities are supported. However, in order to ensure a good quality for bottom surfaces, we restrict the longest segment length. This is done by re-sampling any segment whose length exceeds a threshold (5mm in our implementation).

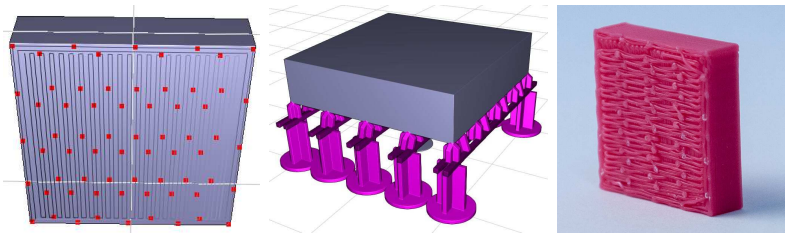


FIGURE 1.21 – Cube hanging in space. **Left)** Set of points requiring support. **Middle)** Generated scaffold. **Right)** Bottom surface quality.

This simple analysis does not take into account the fact that when a point is supported, a whole length of filament around it can be considered supported as well — the cooling filament having a non negligible rigidity. This is the case on perimeters. This effect turns into a surface support when filaments are deposited side by side, for instance when filling an area with a tight infill pattern (see [16] for a nomenclature of such cases). We therefore select only a subset of the detected points. Pseudo code is given by Algorithm 1. $\mathcal{C}(u, v)$ denotes the curvilinear distance between two points u and v on a same print path ; τ is the canceling distance (2mm in our implementation) ; `isUnsupported` tests the disk coverage test using the layer directly below the point. Figure 1.7.1 shows the result for a box hanging in empty space.

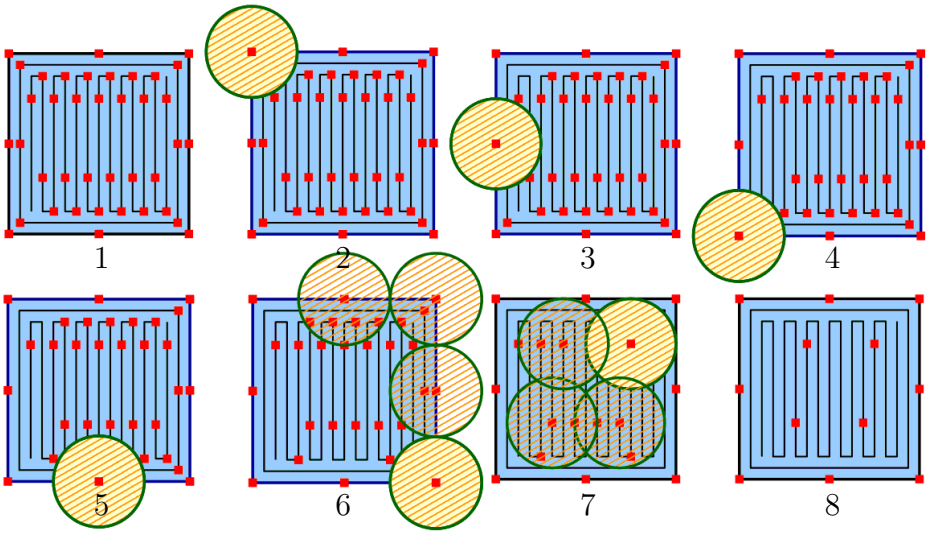


FIGURE 1.22 – The decimation of supporting point on the cube. It starts by removing required point on the perimeters then decimate points on infill segment

Algorithm 1: Select support points

Input: Array of paths $\mathcal{P}$, stored as array of points, ordered by Z $\nearrow$.
Output: Set of points $\mathcal{S}$ to be supported.

foreach *perimeter* `perim` $\in \mathcal{P}$ **do**
  foreach `u` $\in$ `perim` **do**
    if `isUnsupported(u)` **then**
      if $\nexists v \in \mathcal{S}$ *s.t.* $v \in$ `perim` *and* $\mathcal{C}(u, v) < \tau$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$
foreach *non perimeter* `path` $\in \mathcal{P}$ **do**
  foreach `u` $\in$ `path` **do**
    if `isUnsupported(u)` **then**
      if $\nexists v \in \mathcal{S}$ *s.t.* $z(v) \leq z(u)$ *and* $||u - v|| < \tau$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$
return $\mathcal{S}$

1.7.2 Comparing slicers

Either when we improve the quality of a surface or when we generate support structures for 3D printing, we need a way to compare our new techniques with previous work. The different techniques are not always available as publication or the code has not been published, making it hard to reproduce.

In the case of the technique for multi material printing, we wanted to compare the

visual quality of an object printed with our method with others, which was straightforward : it required pictures of the prints under different viewpoints. We also compared the computation time, and the amount of wasted material for auxiliary structures. The key difficulty was to match the parameters as much as possible despite the different slicing algorithms.

Our algorithm that generates supports structures was more difficult to compare with previous work. First of all, our algorithm is integrated into the slicing process, and the print paths are used to compute the set of support points. In contrast, previous support techniques use the angle with the geometry or image based computations to compute the area that need support (see section 2.1.3 of the state of the art). Therefore, we could not easily ensure that two different support structure generation algorithms would be given the exact same set of points as input.

To compare our results with *MeshMixer* [93], we adjusted two parameters in MeshMixer : the angle threshold and the density of point to support to match as closely as possible our support density. The *Makerbot desktop* support generator is integrated to the slicing software, and the slicing algorithm is not the same as ours. In order to compare the print times, we needed to carefully select the slicing parameters such that first print times would match *without* support, and only then would we slice with the support structures.

In general, comparing support structures is very difficult as their performance heavily depends on the choice of subsequent slicing algorithm and, of course, on the geometry to be supported itself. Ultimately, it could be interesting to let the user choose between different algorithms or interactively edit the initial produced support (which, for instance, *MeshMixer* and *Simplify3D* allow).

Conclusion

Our work makes it easier to get satisfactory results with multi-material prints. By carefully analysing the defects and the behavior of the extruders we proposed a technique removing or hiding most issues. Our algorithm finds a better azimuth angle for printing and uses a specialized path planner exploiting a disposable rampart in close proximity of the shape to protect the printed part and refill the extruders with plastic. Our approach also improves the quality of single color prints by hiding zippers in regions of low visibility. This is a side effect of our edge cost definition, which seeks to enter perimeters in hidden locations. We hope our work will help CG enthusiasts turn their designs into colourful, fascinating physical objects.

Some printers are equipped with one extruder with multiple filament for example the Cyclope[11] form E3D. Our method has to be adapted to this case. The refill is still important because the change of material is not immediate : it takes time to flush the previous material from the nozzle. Our refill strategy can be used to absorb this transition time. Zippers still appear and still have to be hidden. However, oozing will appear less because in this case the extruder idles only during travel which might be short enough to consider only retractation. However in this case retractation might lead to an other

11. `http://e3d-online.com/Cyclops`

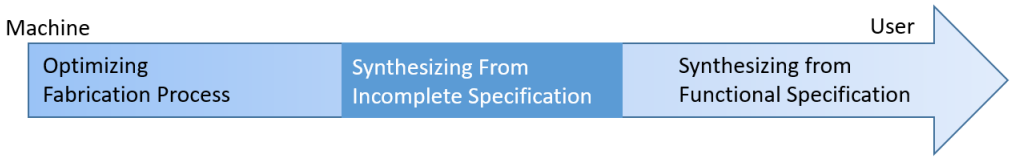
defects, because both filament are pressurized in the extruder and might be mixed during retractation.

The input is a complete specification of a 3D shape. If it needs external support , they are computed on software side, but it does not simplify the design process. It's limited to a particular technology. If we consider only what can be done software side there isn't so much thing that can be done. If we consider that the hardware improvement, this work might become less relevant, in particular if the 3D printing technologies evolve to fast. As I explained in previous section, we can also look for methods that are closer to the user and help her to design the shape. There are two ways to do so, either the algorithm can handle some parts of the design process by completing incomplete specification or by guiding the user in the exploration of the design space. The next chapter shows a method that does the first.

2

Synthesis from Partial Shape Specification

Introduction



It is hard to define a 3D model entirely. It can be done using modelling software that require extensive training to master, such as *Blender* or *3DSMax*, or by writing parametric models through Computer Aided Design(CAD) modeller such as *IceSL* or *OpenSCAD*. Taking into account fabrication constraints makes it even harder in particular if the user is not familiar with those constraints. Moreover she needs to iterate between the fabrication and the design, which is time consuming.

Thus, it seems natural to simplify the design process itself by developing algorithms that help the user to design shapes, or by optimizing shapes to make them fabricable. As we saw in the previous work, the design can be optimized to enforce the fabrication constraints. This chapter presents two contributions inspired by this methodology.

The set of constraints to deal with and the design process heavily depend upon the application domain. Thus, to explore this approach I identified cases where it is particularly interesting. The first case I considered is the aided design of mechanisms, and the second case I considered is for aided design of assemblies from planar cutouts. In both cases, The user needs an extensive knowledge of the fabrication constraints before starting to design. Both cases will be detailed later in this chapter, and they both result in formulation of difficult combinatorial problems, such as a constraint satisfaction problem and a Bin Packing problem, which are both known to be NP-hard.

In the first section I will describe a technique that assists the modeling of mechanisms. It takes into account the fabrication constraints and use a simplified version of the design – given by the user – to produce the final model. The final mechanism is meant to be fabricated pre-assembled (i.e. it is functional immediately after fabrication). This technique is not limited to a particular manufacturing technology, but we demonstrate its efficiency on low-cost filament printers. This work has been done with my advisor and has been published in Computer Graphics Forum, special issue of Eurographics 2015 [39].

The second section tackles the problem of reducing wastage when modeling planar cut assemblies. As discussed in the previous work, these designs are made of several parts cut out from a same master board. The material in between the cutouts is wasted. Given a parametric model of such a design, we minimize the amount of material wastage by optimizing both the packing and the parameters of the design. We presented our results with actual furniture cut with a CNC machine and smaller models cut with a laser cutter. This work has been accepted with major revision at Transactions on Visualization and Computer Graphics. It has been done in collaboration with Bongjin Koo, Niloy Mitra and Sylvain Lefebvre.

2.1 3D Fabrication of 2D Mechanism

Designing mechanism is a difficult task therefore significant research is dedicated to automated their generation or to simplify their design. Yet popular sandbox games allows gamers to design mechanism, either through a simple interface (*Physical Sandbox, Algodoo*) or as an element of gameplay (*Minecraft*). *Algodoo* is a sandbox game in witch the user can design and simulate 2D mechanism. The simple interface and graphics of the game make it appealing and fun. This game not only simplify the design process, but also make it enjoyable.

In this section, I seek to fabricate mechanism with low-end 3D printer, that were design with those kind of interface. While modelling mechanism for fabrication is an hard task, we developed a method that support part of the modelling for the user. Given underspecified mechanism(i.e. 2D scheme) we compute a 3D printable shape that make the same motion.

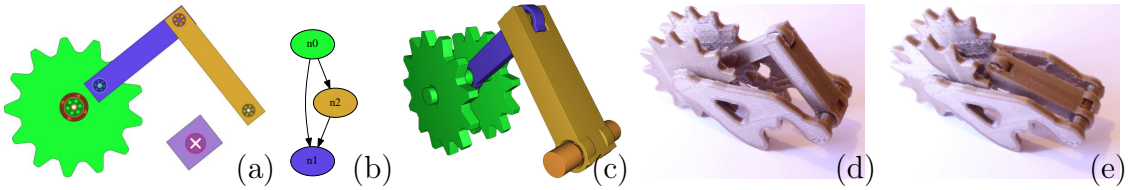


FIGURE 2.1 – Our algorithm takes as input a 2D design of a mechanism (a) and produces a 3D model (c) by computing a layout encouraging inclusion between parts, formulating the layout as a graph orientation problem (b). In the graph, edge orientation indicates inclusion relationships. The 3D model can be printed (c,d) and is functional. The chassis is automatically synthesized.

Contributions

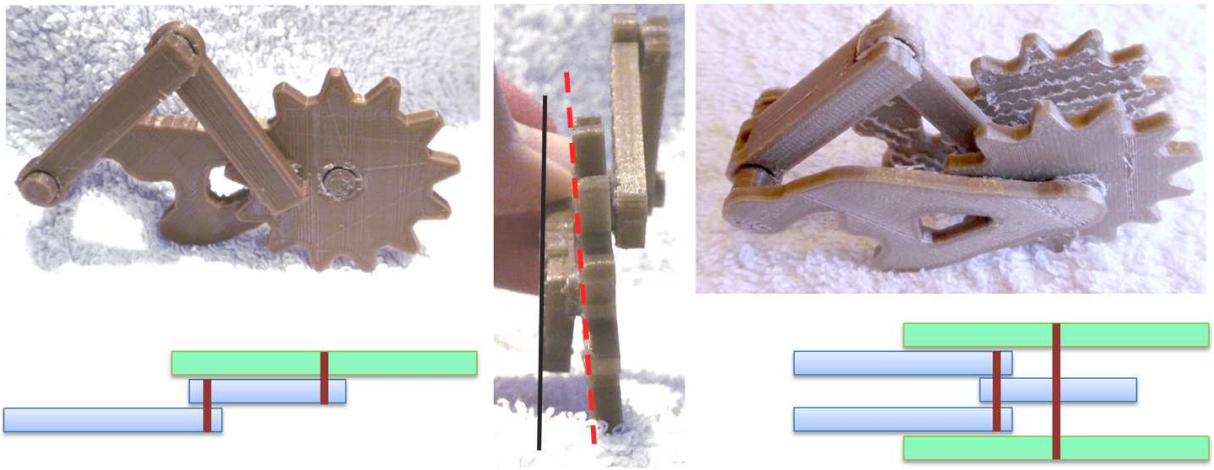


FIGURE 2.2 – Possible solutions for the example of Figure 2.1. **Left** : Print out of our result using layering only. **Middle** : Out-of-plane motion resulting from using layering. **Right** : Print out of our result using inclusion. It produces less out-of-plane motions as axles are connected on both extremities.

— We solve for the 3D layout of the mechanism by exploiting *inclusions* whenever possible.
— We define the 3D shapes of the final parts by constructive solid geometry (CSG), considering the positions of the parts throughout the mechanical simulation.
— We automatically synthesize a chassis for the mechanism using topology optimization. Our approach takes into account the forces generated by the mechanism on the chassis during the entire simulation.

Our approach makes no assumption regarding the shape of the mechanical parts : gears, cranks and cams are not tagged in the input. They function properly through the preservation of the set of contacts and interactions from the 2D mechanism into the fabricated mechanism.

Limitations Our algorithm does not detect unrealistic mechanisms (i.e. mechanisms that may generate excessive stresses, that may exhibit singularities, or mechanisms where parts can fall or detach). Some over-constrained 2D designs cannot be resolved by layering or inclusion (Section 2.1.2.6).

2.1.1 Overview

The input to our algorithm is a set of N *parts* $\mathcal{P} = \{P_i | i = 0..N-1\}$, each described by a 2D polygon. The polygon of a part may have holes, but has to form a single component. The parts may be connected through hinges and fixed joints. We denote the set of joints :

$$\mathcal{H} = \{(P_i, P_j) | P_i, P_j \text{ connected by an hinge or a fixed joint}\}$$

Our approach involves three steps : layout (Section 2.1.2), geometry synthesis (Section 2.1.3) and chassis synthesis (Section 2.1.4). These three steps are illustrated in Figure 2.3.

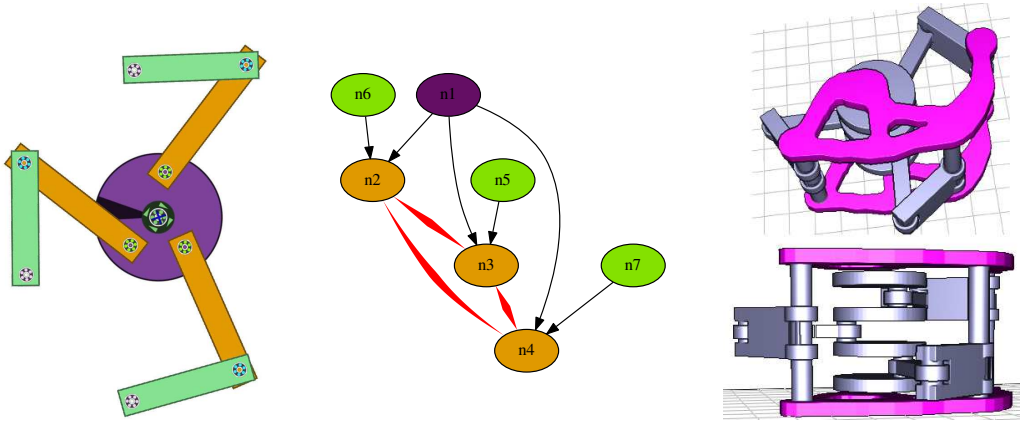


FIGURE 2.3 – **Left :** Input 2D mechanism. **Middle :** Graph used to construct the layout. Edge orientation indicates which part include which other, while the red edges show which parts have to be layered. **Right :** Two views of the generated mechanism. The synthesized chassis appears in pink. The crankshaft configuration automatically results from the inclusion and layering constraints.

The layout step determines the relative positions of parts and assigns them with a depth interval. We denote $I(P_i) = [l_i, h_i]$ the interval assigned to part P_i, with $l_i \leq h_i$ two integers. The inclusions between parts are determined by orienting edges in a graph capturing contact and collision constraints. The geometry synthesis step determines the exact 3D geometry of each part, using the layout and the motions resulting from the simulated mechanism. The last step synthesizes a *chassis* for the mechanism : a geometry for the main body holding everything together.

2.1.2 Mechanical layout

Our approach produces mechanisms favoring inclusion between parts. This is used in particular to resolve cases where the parts overlap in the 2D specification without interacting. The main advantage of this approach is to reduce mechanical jitter : the hinge axles are much stronger when supported on both their extremities. Inclusion alone cannot work on all mechanisms due to additional geometric constraints and interactions between parts. For these cases our approach resorts to layering, but only locally.

The layout process starts by analyzing the simulated mechanism, tracking overlaps and interactions between parts (Section 2.1.2.1). This information provides us with a set of observations that are used to make hypotheses regarding which parts can include which others (Section 2.1.2.3). Our algorithm greedily add inclusions, until contradictions are detected (such as having both $A \supset B$ and $B \supset A$). These contradictions are transformed into layering rules. The final optimization assigns depth values to the intervals by solving a constrained satisfaction problem (Section 2.1.2.4).

2.1.2.1 Analysis

Our approach starts by simulating the mechanism to construct the set of observations. It keeps track of *overlaps* – parts that overlap without colliding – and *interactions* – parts that are allowed to be in contact during the simulation. This is done by simulating the input 2D mechanism using the *Box2D* library.

We assume mechanisms to have a periodic motion, and run the simulation until a prior configuration is encountered, or a user selected maximum time is reached. The result is a set of T time frames. We denote by M_i^t the position matrix of part P_i at time $t \in [0, T[$, and denote by $M_i^t P_i$ the polygon of part P_i at time t.
We record all overlaps between parts in a set :

$$\mathcal{O} = \{(P_i, P_j) | \exists\ t \text{ such that } P_i, P_j \text{ overlap at time } t\}$$

We keep track of all interactions between parts during simulation. We denote the set of interacting parts as :

$$\mathcal{C} = \{(P_i, P_j) | \exists\ t \text{ such that } P_i, P_j \text{ are in contact at time } t\}$$

Note that for any two parts P_i, P_j we have $(P_i, P_j) \in \mathcal{O} \Rightarrow (P_i, P_j) \notin \mathcal{C}$ and $(P_i, P_j) \in \mathcal{C} \Rightarrow (P_i, P_j) \notin \mathcal{O}$. We also have $\mathcal{H} \subset \mathcal{O}$ since all parts sharing a joint also overlap.

From these sets we define the mechanism graph as $\mathcal{G} = (\mathcal{P}, \mathcal{H} \cup \mathcal{O} \cup \mathcal{C})$. The set of edges is the union of the joint set $\mathcal{H}$, the overlap set and the contact set. Each edge in the graph is tagged to track which set it belongs to (edges in both $\mathcal{H}$ and $\mathcal{O}$ are tagged as belonging to $\mathcal{H}$).

Additional observations are made about the mechanism. The first are *erasing* cases. A part P_j is said to *erase* a part P_i if the temporal sweep of its polygon covers P_i entirely *and* they are not connected by a joint. More formally :

$$P_j \text{ erases } P_i \text{ if } P_i \setminus \cup_{t \in [0,T[} \left((M_i^t)^{-1} M_j^t P_j\right) = \emptyset \text{ and } (P_i, P_j) \notin \mathcal{H}$$

In such cases, the part P_i cannot include the part P_j as it would have to be split in two independent parts to fit P_j within its depth. If P_i, P_j are connected by a joint, then P_i remains connected through the axle and is not erased by P_j.

The second are *detachment* cases. A part P_j is said to *detach* $P_i, P_k (k \neq j)$ if the temporal sweep of P_j overlaps with the joint between P_i, P_k. Note that the overlap test is considering the diameter of the joint axle (5 mm in practice). If P_j detaches P_i, P_k then we request that P_j includes both P_i and P_k to avoid having P_j cross the axle between P_i, P_k.

These cases are illustrated Figure 2.4.

2.1.2.2 Mechanical configurations

The mechanisms we generate can produce three types of mechanical configurations between two parts A and B. Each results in a rule regarding their intervals :

1. A includes B (denoted by $A \supset B$)

$$\Rightarrow I(A) \supset I(B)$$

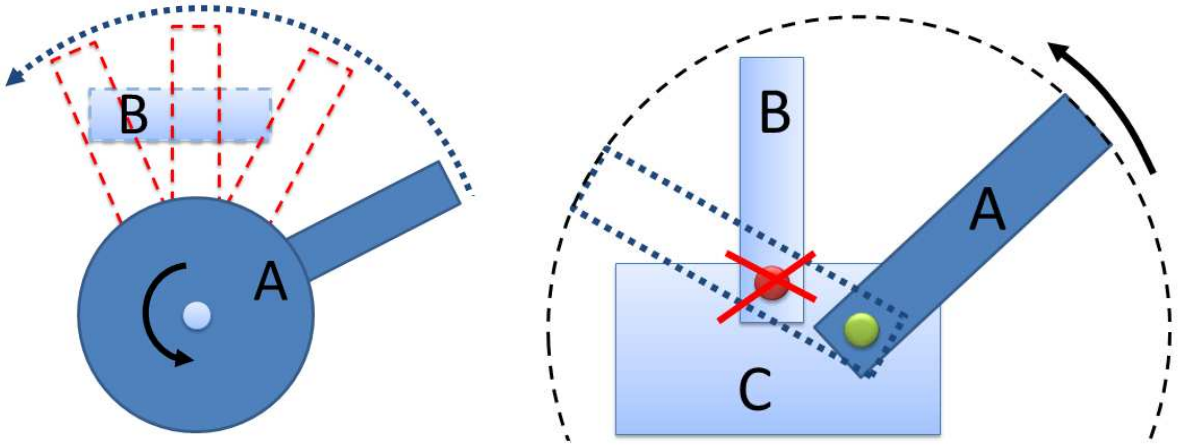


FIGURE 2.4 – **Left :** The bar A sweeps across the (fixed) bar B, covering it entirely during the simulation. A is said to *erase* B. In this case, B cannot possibly include A within its depth or it would be disconnected in two independent parts. **Right :** During simulation, the bar A sweeps across the hinge between B and C. A is said to *detach* B and C. In this case we request A to include B and C so that their axle is not cut by A. Note that these constraints will be combined and may conflict, in which case layering will be locally performed.

2. A interacts with B

$$\Rightarrow (I(A) \backslash \cup_{\forall P \subset A} I(P)) \cap (I(B) \backslash \cup_{\forall Q \subset B} I(Q)) \neq \emptyset$$

3. A layered with B

$$\Rightarrow I(A) \cap I(B) = \emptyset$$

These configurations are depicted in Figure 2.5. They are mutually exclusive : if A is layered with B they cannot interact as they share no interval in depth. If A is layered with B, then B cannot be included within A (and vice-versa) as they also share no interval. If A includes B, then the volume of B is carved out from A (see Figure 2.5 top-left) and therefore they cannot interact.

Whenever possible our algorithm favors inclusion over the other two configurations.

2.1.2.3 Generating inclusions

The goal of this step of the algorithm is to generate as many inclusion configurations as possible. To this end, we exploit the observations made during analysis and greedily attempt to include parts into one another. Whenever inclusion cannot be used we fallback to layering. At this stage we only determine relationships between the parts intervals (e.g. $I(A) \supset I(B)$). The exact values of the depth intervals will be computed at the next stage (Section 2.1.2.4).

Inclusion relationships are determined by orienting the $\mathcal{H}$ and $\mathcal{O}$ edges of the mechanism graph $\mathcal{G}$ (Section 2.1.2.1). An edge $(P_i \rightarrow P_j)$ implies that $I(P_i) \supset I(P_j)$. The goal is to find an acyclic orientation : an inclusion cycle would produce an unsolvable case

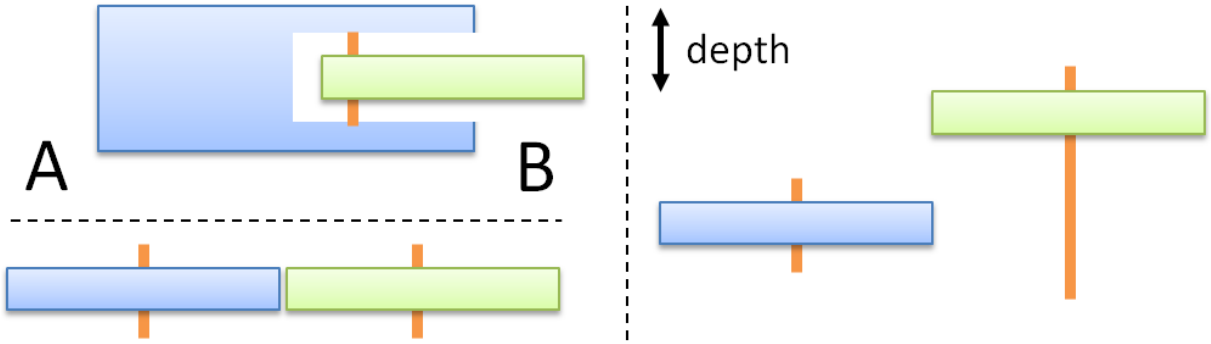


FIGURE 2.5 – Two parts in the three possible configurations, as seen from above. Inclusion (top-left), interaction (bottom left), layering (right). The configurations can be mixed, i.e. a part can include others that are in a layering configuration.

of circular inclusion, e.g. $I(P_i) \supset I(P_j) \supset I(P_i)$. When orienting the edges we take care not to contradict the interaction rules (recall that *A includes B* is incompatible with *A interacts with B*, see Section 2.1.2.2). This is done by verifying that no inclusion path is formed between two parts A and B if they interact in the input mechanism.

The orientation of some of the edges is constrained by the *erasing* and *detachment* cases :

$$\begin{array}{lcl} P_j \text{ erases } P_i & \Rightarrow & I(P_j) \supset I(P_i) \\ P_j \text{ detaches } (P_i, P_k) & \Rightarrow & I(P_j) \supset I(P_i) \text{ and } I(P_j) \supset I(P_k) \end{array}$$

As a result of these constraints, contradictions may appear during the orientation process, where both $I(P_i) \supset I(P_j)$ *and* $I(P_i) \subset I(P_j)$ are required. Such cases are resolved by switching to a layering configuration for P_i and P_j.

We orient the edges of $\mathcal{G}$ considering edges in $\mathcal{H}$ first and then edges in $\mathcal{O}$. In each subset, we start by the edges with a constraint. This order is important : we prefer to avoid layering on joints (set $\mathcal{H}$) since this is the case where fragile axles are generated. We therefore orient the edges in $\mathcal{H}$ first, so that contradictions are more likely to appear on the edges of $\mathcal{O}$ which are later considered.

For each edge in sequence, we first determine which orientations are possible. This involves checking for an existing constraint, verifying whether an orientation violates any interaction rule, and then checking whether an orientation would produce an inclusion cycle. If two orientations are possible we use a heuristic to select the orientation of lowest cost (see below). If no orientation is possible, we have encountered a contradiction.

Contradictions. We deal with contradictions by removing the problematic edge from the graph and by adding a layering configuration rule between the parts : $I(P_i) \cap I(P_j) = \emptyset$. This forces P_i and P_j to be in different depth intervals instead of being included into one another. However, this introduces an additional requirement on the graph orientation : the parts P_i and P_j should not have any common descendant in the oriented graph. Let us assume such a descendant P_k exists. We would have $I(P_i) \supset I(P_k)$ and $I(P_j) \supset I(P_k)$ which gives $(I(P_i) \cap I(P_j)) \supset I(P_k)$. This directly contradicts $I(P_i) \cap I(P_j) = \emptyset$. To account for this, every time a contradiction is detected we add the additional constraint that P_i, P_j

should not have a common descendant. The set of descendants is easily maintained during the graph orientation algorithm, and we reject any edge orientation that would violate such a requirement. When checking for common descendants we also follow interaction edges (edges in $\mathcal{C}$) to prevent the layered parts to include parts that have to interact, i.e. $P_u \subset P_i, P_v \subset P_j$ where P_v, P_u interact : this would result in a similar contradiction.

Resolving a contradiction by layering removes all detachment constraints between P_i, P_j : as both parts will be placed in non-intersecting depth intervals, they can no longer cross their respective axles.

Since the set of constraints is updated, and because some earlier choices may violate the new constraints, we restart the graph orientation every time a contradiction is resolved. When the process restarts edges are traversed the same order, skipping the edges removed due to contradictions.

Chassis. The chassis appears as a part in the graph. To guarantee that it includes all other parts, we orient the chassis edges prior to considering any other edge in the graph.

Figure 2.2 illustrates a case where the chassis is involved in a detachment constraint. In such cases, we request the detaching primitive to be the most included. That is, P_j detaches (P_i, C) implies $I(P_j) \subset I(P_i)$ and $I(P_j) \subset I(C)$, where C is the chassis. The axle between C and P_i will exist around P_j – even though P_j cuts it, it will exist in two parts attaching C and P_i on both sides. E.g. in Figure 2.2 the main wheel axle is cut by the inner arms. Nevertheless the wheel is properly connected on both sides to the chassis, and remains a single part through the axle with the arm. This is possible as long as P_j does not erase P_i, in which case layering would automatically be used between P_j and P_i. Indeed a contradiction would then appear when orienting the edge (P_i, P_j).

Orientation cost heuristic. Whenever we can freely choose the orientation of an edge, we apply the following cost heuristic. In absence of constraints our goal is to avoid thickening the parts too much. In other words, we want to keep the size of the depth intervals $|I(P_i)|$ small. The intervals are optimized at the next step (Section 2.1.2.4) and therefore we do not know their exact size during graph orientation. However, we can easily determine a lower bound. Consider a path in the oriented graph from part P_i to part P_k : $P_i \to ... \to P_k$ and denote L the length of this path. Since we have $P_i \supset ... \supset P_k$, it follows that $|I(P_i)| \geq L$. We therefore seek to minimize the length of the longest path from every node. When orienting an edge (P_i, P_j) we select the orientation minimizing $\mathcal{L}(\mathcal{G}_o, P_i) + \mathcal{L}(\mathcal{G}_o, P_j)$ where $\mathcal{L}(\mathcal{G}_o, P)$ computes the longest path from P to any other node in the oriented graph $\mathcal{G}_o$ (the graph with only the previously oriented edges, and the edge being tested).

2.1.2.4 Depth intervals

The previous step produces a number of rules relating the depth intervals of the parts. The goal of this section is to compute the depth values assigned to the lower and upper bounds of each interval $I(P_i) = [l_i, h_i]$. We assign integer depth values to the intervals, which are later mapped to physical thicknesses in the final object (see

Section 2.1.3). Intervals where $l_i = h_i$ correspond to parts having the minimal thickness (2 mm in practice).

After the analysis we obtain three types of interval rules : inclusion (e.g. $I(P_i) \supset I(P_j)$), layering (e.g. $I(P_i) \cap I(P_j) = \emptyset$) and interactions.

Inclusion. Inclusions are captured by the oriented edges in $\mathcal{G}$. They result in the following inequalities :

$$I(P_i) \supset I(P_j) \Rightarrow l_i < l_j \text{ and } h_i > h_j$$

Note the use of strict inequalities, which guarantees that the including part (P_i) has one layer on each side of the included part (P_j). This ensures that axles are supported on both sides.

Layering. Layering rules are produced when resolving contradictions on edge orientations. We distinguish layering due to the removal of an overlapping edge ($\in \mathcal{O}$) from the layering due to the removal of an hinge edge ($\in \mathcal{H}$).
On overlapping edges :

$$P_i \text{ layered by overlap with } P_j \Rightarrow l_i > h_j + 1 \text{ or } l_j > h_i + 1$$

These inequalities guarantee that P_i, P_j will have a spacing of at least one between them, ensuring an including parent piece will support their axles on both sides.
On hinge edges :

$$P_i \text{ layered by hinge with } P_j \Rightarrow l_i = h_j + 1 \text{ or } l_j = h_i + 1$$

This constrains both parts to appear next to each others through depth, ensuring that the axle between them is as short as possible.

Interactions. These rules are necessary to enforce that contacts exploited by the mechanisms (gears, racks) are properly captured by the 3D model. They result in the following equalities :

$$(P_i, P_j) \in \mathcal{C} \Rightarrow l_i = l_j \text{ or } l_i = h_j \text{ or } h_i = l_j \text{ or } h_i = h_j$$

This ensures that the parts will properly interact through their top or bottom layers. This rule is more restrictive than it could be, since in principle the parts interact as long as they share an interval where no included part exists (see Section 2.1.2.2). However, we found it sufficient in practice while reducing the combinatorial complexity for the CSP solver.

2.1.2.5 Solver

We directly translate these rules into an integer constraint problem that we solve using *Minion* [36]. We restrict the space of integer to $[0, 2 \cdot N]$, with N the number of parts.

Minion returns the solution minimizing the sum of part thicknesses, that is $\sum_{i=0}^{N} |I(P_i)|$. As Minion performs an exhaustive search within the solution space, we configure it to return the best solution found after at most 90 seconds.

Once the intervals are determined for each part, we are ready to generate the final geometry of the 3D parts.

2.1.2.6 Discussion, failure cases

At worse our algorithm eliminates all edges between the parts in the graph and includes everything in the chassis. The parts will then be layered within the chassis, which becomes a crankshaft hosting the layered mechanism.

However, this does not guarantee success as some mechanisms cannot be layered. Such a case is shown Figure 2.6. The only solution involves modifying the shape of the fixed joint between the two bars. This is not considered by our system nor, to the best of our knowledge, by any of the existing techniques. In such cases, the CSP solving for the intervals will admit no solution.

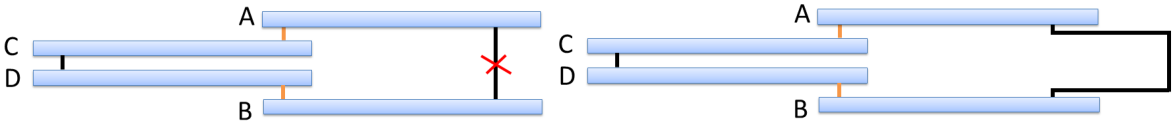


FIGURE 2.6 – **Left :** Four bars seen from the top. Black segments indicate a fixed joint, while orange indicates a hinge joint. C,D can rotate around A,B but will cut the fixed axle between A,B. This system cannot be resolved by assigning different layers to A,B,C,D. **Right :** The only solution is to change the shape of the axle, making room for C,D.

2.1.3 Part geometry synthesis

This step of the process takes as input the set of parts $P_i, i = 0...N$ and their corresponding depth intervals $I(P_i) = [l_i, h_i]$. The output is the 3D geometry of each part.
In order to produce the 3D geometry we take into account the following :

- Parts have to be carved to allow for passage of other, included parts.
- Parts that come in contact have to be modified to take a spacing tolerance into account (0.4 mm in practice).
- Parts that slide along others without being attached to the chassis have to be maintained at their selected depth.
- The geometry of hinge joints has to be produced, enabling the mechanism to print pre-assembled.

The base shape of a part is formed by the linear extrusion along the z axis (depth) of its 2D shape. We denote by B_i the base volume of part P_i. The position of the part in depth is computed such that pieces allotted in consecutive layers are separated by a small space. For an interval $[l_i, h_i]$, the extrusion takes places between $z_i^l = (t + s)l_i$ and $z_i^h = (t+s)(h_i+1)-s$ where t is the minimal thickness (2 mm) and s the spacing tolerance between consecutive parts (0.4 mm).

The final shape S_i is obtained from B_i by subtracting from the initial volume the time sweep of the pieces included in P_i, as well as the time sweep of the pieces in contact with P_i. All subtracted pieces are dilated by the contact tolerance spacing. The volume S_i is precisely defined as :

$$S_i = B_i \setminus \left(\cup_{j \in \mathcal{N}(i)} \cup_{t \in [0,T[} \left((M_i^t)^{-1} M_j^t B_j \right) \oplus \mathcal{B}_{0.4}^z \right)$$

where $\mathcal{N}(i) = \{j|(P_i, P_j) \in \mathcal{O} \cup \mathcal{C}\}$, $\oplus$ is the dilation operator and $\mathcal{B}^z_{0.4}$ is a cylinder of axis z, of height and diameter 0.4 mm. We compute the shape by a combination of 2D operations using the *Clipper* library [48], and a few 3D CSG operations.

Free assemblies. The mechanism might include parts, or sub–assemblies of parts that are not connected to the background by any hinge : they are only sliding along other parts. We call these *free assemblies*. They are detected as disconnected components in the graph with only $\mathcal{H}$ edges.

Free assemblies require special care : in 3D nothing prevents them from falling out of their assigned depth interval – recall however that we assume that they are properly locked inside the 2D mechanism. We address this issue by creating fins (protrusions) along the sliding parts, see Figure 2.7. These fins are added to the base shapes B_i of the parts, and are thus subtracted from the other parts that are in contact (with a spacing tolerance). They physically constrain the parts to remain aligned in depth.

This approach in currently limited to the case of free assemblies sliding against non-free assemblies : we do not support several free assemblies sliding against each others.

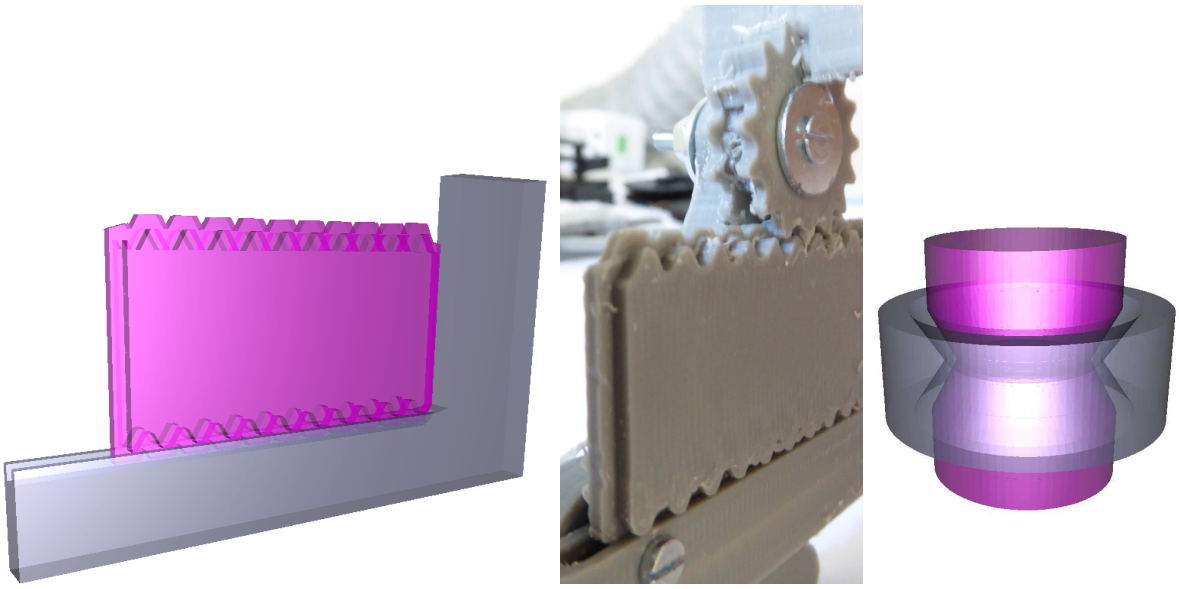


FIGURE 2.7 – **Left :** The part in pink slides along the L-shaped part. Our approach generates fins around the sliding part so that it is locked in depth inside surrounding parts. **Middle :** Printed result, note how the gear is also carved by the fin. **Right :** Interlocking shape of the axles allowing for pre-assembled printing on filament based printers.

Hinges. The geometry of the hinges is designed to allow for pre-assembled printing (see Figure 2.7, right). We add the hinges by CSG, carving the parts to let the axles through. If the part is too narrow the axle geometry will introduce a local bulge to ensure the axle fits properly – in some cases this can prevent the mechanism to function, e.g. if a surface along which two parts interact is modified. We do not currently address this issue.

After this stage the mechanism is almost ready to print, but still lacks a chassis.

2.1.4 Chassis synthesis

When designing mechanisms in 2D the user attaches primitives to the background 'wall'. When creating the 3D counterpart of the mechanism we have to synthesize a *chassis* acting as the background. We produce a 2D geometry that is used to sandwich the 3D mechanism in between two walls, as illustrated Figure 2.2. Alternatively the chassis could be extruded to the full mechanism depth and carved like any other part, see Section 2.1.3. This is unnecessary unless the chassis is a crankshaft, so to reduce material use and print time we prefer the sandwiching approach in practice.

The trade-offs in synthesizing the chassis are that we want it to be strong enough to support the efforts generated by the mechanism, but at the same time we would like to keep it small to reduce material use, print time and for aesthetics reasons (a thick chassis would hide the mechanism entirely). This problem is elegantly answered by topology optimization techniques [7].

2.1.4.1 Background on topology optimization

We cast chassis synthesis as a case of 2D topology optimization for minimizing the compliance energy. The optimization domain is a grid of square elastic elements where each element i, j takes a density $\rho_{i,j} \in [\rho_{min}, 1]$. We denote by ρ the vector of all element densities. Given a choice of densities and a set of fixed elements where the structure is anchored, the finite element method can be used to compute the planar deformation due to a set of forces $\mathbf{f} = f_1, ..., f_n$ located at the grid nodes (element corners). The displacement vector for all grid nodes $\mathbf{u}$ is obtained by solving $\mathbf{K}(\rho) \cdot \mathbf{u} = \mathbf{f}$ where $\mathbf{K}(\rho)$ is the global stiffness matrix assembled from the elements. The stiffness matrix of an element $\rho_{i,j}$ is given by $\rho_{i,j}^3 K_e$ where K_e is the 8×8 stiffness matrix of a square element in the target material.

The compliance energy is defined as $E(\rho) = \mathbf{f} \cdot \mathbf{u}$. Minimizing the compliance maximizes the rigidity of the system under the given forces. This energy is minimized by gradient descent under the constraint that $\sum_{i,j} \rho_{i,j} = A$, where A is the target area of the produced structure. Thanks to the cubic exponent in the per-element stiffness, the system tends to use only 0 or 1 for $\rho_{i,j}$.

2.1.4.2 Chassis optimization

The chassis has to resist to the forces exerted by the mechanism at all times. We therefore record the forces at the joints attached to the background for the entire simulation. This gives us a set of T force configurations, where T is the number of time frames. We optimize the chassis by maximizing the compliance over all time frames, that is $E(\rho) = \sum_{t \in T} \mathbf{f}_t \cdot \mathbf{u}_t$ where $\mathbf{K}(\rho)\mathbf{u}_t = \mathbf{f}_t$. This is made efficient by pre-factoring the stiffness matrix $\mathbf{K}$ and then solving for $\mathbf{u}_t$ for all time frames.

We select a resolution of 2 mm per pixel, a Poisson ratio of 0.35 and an elastic modulus of 2.3 GPa (ABS plastic material). The target area A is set to 10% and then reduced

until the structure is disconnected. We select the last value generating a fully connected structure. The final outline of the shape is extracted by smoothing out the result with a box filter of size 3×3 and then contouring the isovalue 0.25. The process is summarized Figure 2.8.

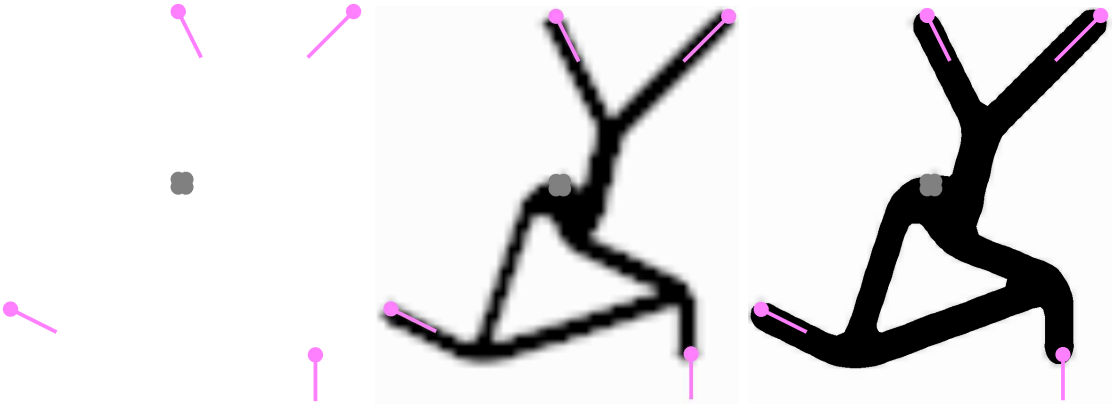


FIGURE 2.8 – Chassis synthesis by topology optimization. **Left :** four forces resulting from hinges at a given time frame (pink) and attachment point (gray). **Middle :** Optimized densities for these forces. **Right :** Final shape after smoothing and contour extraction.

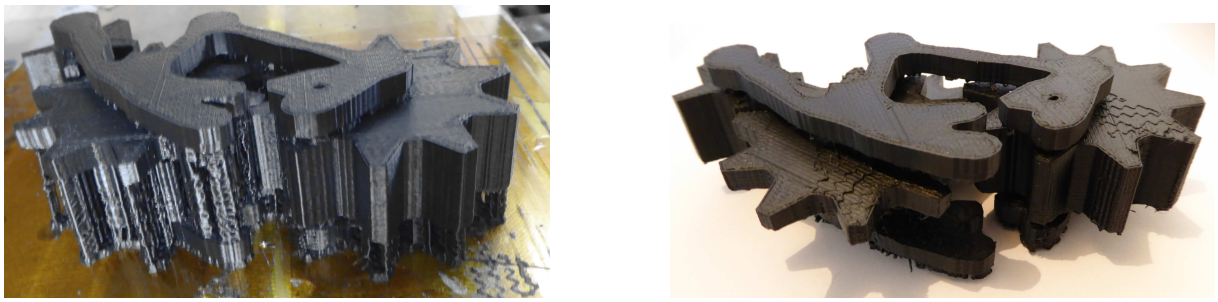


FIGURE 2.9 – **Left :** Model after printing, with weak infill support. **Right :** Cleaned model. The gears have rotated, note the small leftover from the infill pattern on top of the gears.

2.1.5 Results

We modeled all our results using *Algodoo*, with the only constraint that the mechanisms have to function properly in 2D and be in a valid position (interacting parts such as gears should not overlap). By default the models are rescaled so that their bounding box fits a 150×150 mm square.

All the 3D mechanisms are generated automatically from the scenes created in *Algodoo*, without any user intervention.

Figure 2.10 shows a number of printed results. From top-left to bottom right : the *Gear Puppet* model is made of six gears with custom shapes. The small brown box indicates

the base for the chassis. The *EG flag* model is a case of sliding part. The model has automatically generated fins along the sliding part that are carved from both the L-shaped part and the gear. This model broke during clean up and we used screws to repair it (see also Section 2.1.5.1). The *Scissor* model is a case where no chassis is needed. It is actioned by a hidden mechanism which is ignored by our system after simulation. This shows how our algorithm alternates inclusions as a result of the edge orientation cost heuristic. The *Wheel* model illustrates how part geometry is automatically created to allow for passage of included objects. The green box at the bottom of the design indicates the base of the chassis. The *Gear Train* model is case of cyclic interaction between gears, creating a layering between the two central gears. The *Three Leg* model is a case that results automatically in a crankshaft. It is also shown Figure 2.3.

The complexity of our printed results is limited by the capability of our printers. We show in Figure 2.11 and Figure 2.12 outputs of our algorithm on more complex designs. Our system is capable of keeping the overall design thickness small, while exploiting inclusions wherever possible.

Performance. Most results are computed in a few seconds. For instance, for the small model Figure 2.1 graph orientation takes 154 ms, Minion returns the CSP solution in 7 ms. For the larger model Figure 2.12 graph orientation takes 489 ms, Minion returns the CSP solution after 90 seconds as it explores for the best possible solution until the timeout. Running for longer does not return a better solution, but Minion has to finish exploring the space to guarantee the optimal is found. Chassis optimization takes 2.34 seconds.

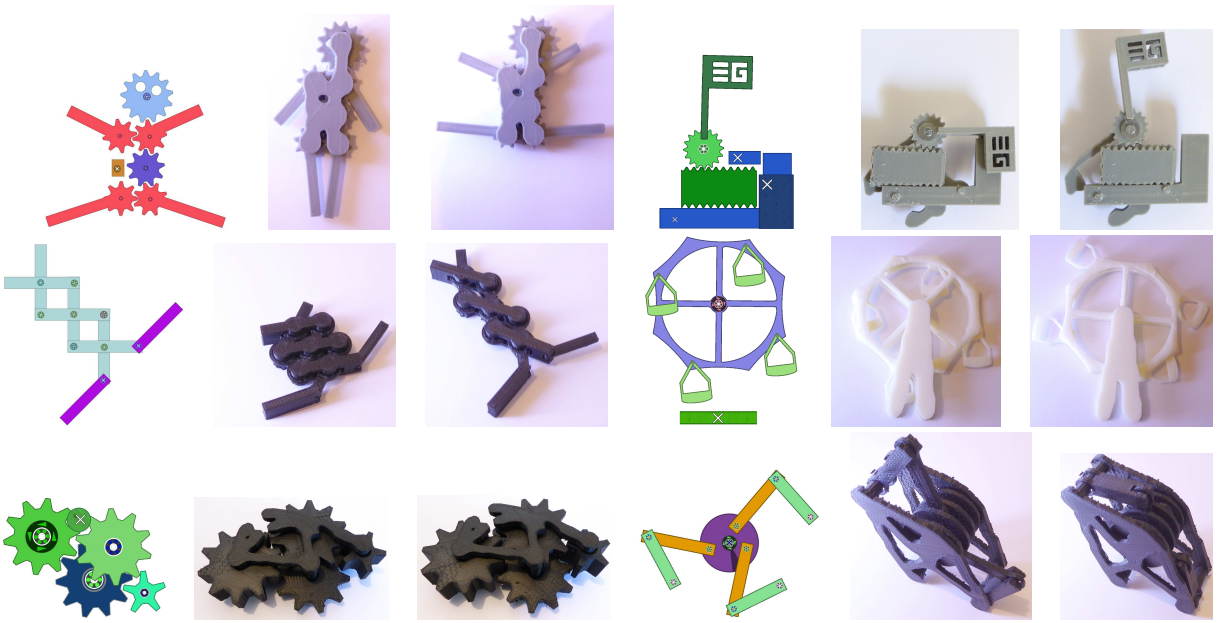


FIGURE 2.10 – A variety of results automatically generated from the input 2D design. Please also refer to the accompanying video clips to see them in motion. All these results are 3D printed on filament printers.

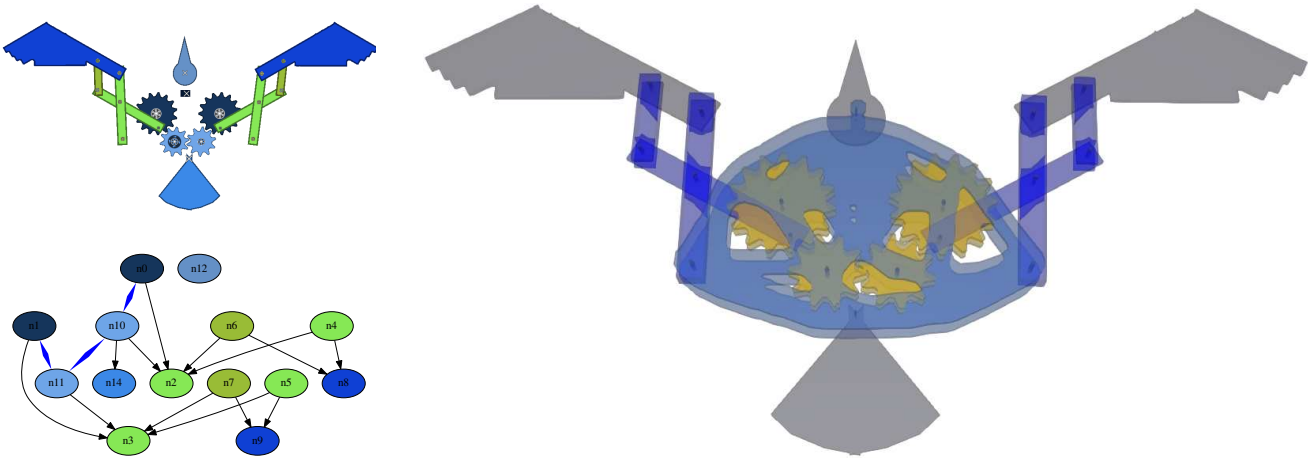


FIGURE 2.11 – *Eagle* model. The mechanism shown in transparency (bottom) is generated from the design (top). The mechanism graph (top left) shows inclusions and contact edges (bold blue). No contradictions were encountered : there is no layering. Note the double gears including the bars, as well as the space carved for the inner bars in the bars actioning the wings.

2.1.5.1 3D printing

We print all our objects on inexpensive filament printers in ABS and PLA plastic : a Replicator 1 from *Makerbot*, and Ultimakers 1 and 2 from *Ultimaker*.

All objects are printed in one piece, using support. We experimented both with dissoluble plastic and a weak filling pattern for support. We had better results using a weak infill pattern. See Figure 2.9 for an example of a print before and after cleanup.

Our mechanisms exploit the fact that 3D printers can produce pre-assembled articulated objects, so that we do not have to consider the assembly stage. However, this is not necessarily the best option of filament printers. In particular after printing some force has to be exerted to free the mechanism from inaccessible support. One issue we encountered is breaking of the plastic axles when applying force. The *EG flag* design, for instance, broke when we freed the sliding part and we had to use screws to reassemble it manually. An interesting direction of future work is to split the mechanism into pieces that are easy to assemble [62, 122].

Conclusion

We show that we can create a shape, here a mechanism, from partial specifications. The initial input is not fabricable, and we computed a 3D printable shape from it. We wanted to minimize the heights of the mechanisms under the constraints (collisions and contacts) that were discovered during the simulation. The algorithm knows the shape if different parts and compute the relative positions of the parts of the mechanisms.

The lack of the user in the loop that generates the final shape can be seen as a limitation. The mechanism does respect the fabrication constraints, but the result might

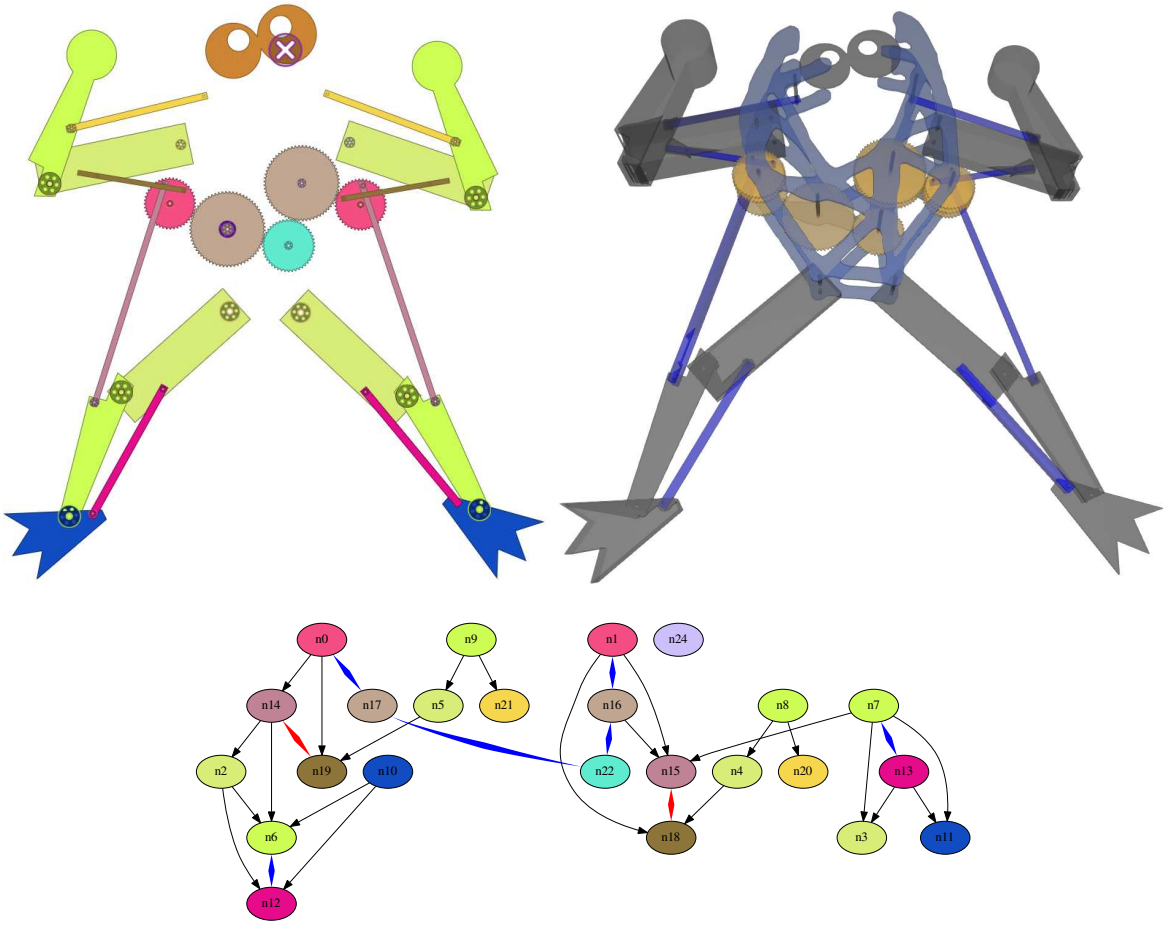


FIGURE 2.12 – *Frogger* model. The mechanism shown in transparency (top right) is generated from the design (top left). This model contains two crankshafts for steering the arms and legs. These were created by layering constraints (red bold edges in the graph). Inclusions are used in most places. Note the lightweight synthesized chassis. One of the large wheel is doubled to avoid the bottom bar of the neighboring crankshaft which overlaps during motion (edge between n15 and n16 in the graph).

be unexpected. Even if the shape respects the initial design and its simulation, the relative positions of the different parts are important for the global aesthetics of the mechanism. Using user intents to guide the system would be important. It would lead to a different system and methodology though, as this would be more 'modelling with the user' than 'modelling from user specifications'.

2.2 Towards zero-waste furniture design

There are software that allow to design parametric models for fabrication (*IceSL, OpenSCAD...*). We can find such parametric models on specialized websites like "Thingiverse". A parametric model defines a function that generates a shape from a set of parameters to a design space. The parameters of the function are tuned by the user such that the shape is good looking, and fits a function. In some cases, the exact values of the parameters are not important to the user, however they impact the fabrication process. In this case, there is an opportunity to optimize the parameters automatically.

In this work, we introduce the problem of *waste-minimizing planar cutout designs*, and investigate it in the context of flatpack furniture design using laser cut wooden parts. Specifically, we study the interplay between design exploration and cost-effective material usage.

FIGURE 2.13 – We introduce waste-minimizing furniture design to dynamically analyze 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%.

2.2.1 Design Workflow

Our goal is to propose design variations that minimize 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.

The user starts by choosing the desired material (i.e., thickness of wooden planks) and the number and dimensions of the master board(s). Our system considers rectangular master boards — in practice these can represent new boards or left over rectangular spaces in already used boards. The user starts by loading an initial part-based 3D object design, either created in a modelling system or as a parametric model. The parts can be rectangular or have curves boundaries. The user specify also a range for all the variable that can be modified by the system.

The algorithm 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) utilized. 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.

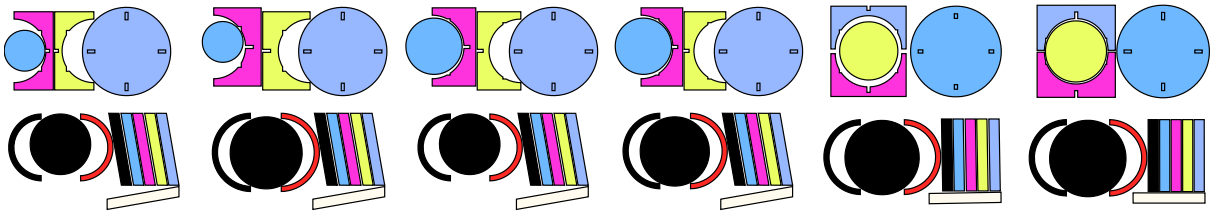


FIGURE 2.14 – Evolution of shape variation across a run of our algorithm on the coffee-table (top) and low-chair (bottom) models.

2.2.2 Overview

Our goal is to analyze aspects arising from material considerations, and investigate how design changes affect such considerations. Specifically, we ask how to adapt a furniture design so that it makes better utilization of material in the resultant design layout. Note that this is the inverse of the design rationalization problem, i.e., instead of taking a design as fixed and best fabricating it, we adapt the design so that the resultant rationalization makes better utilization of available material.

2.2.2.1 Parameterized 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 2.2.4 parametric designs modelled by CSG. We however expect a continuous behavior from $D(\mathbf{X})$, i.e., 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 optimization 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.

2.2.2.2 Material space

Since we focus on laser cut furniture, any 3D design given by a configuration vector $\mathbf{X}$ is realized as a layout (i.e., cutting plan) in the material space. Material space is

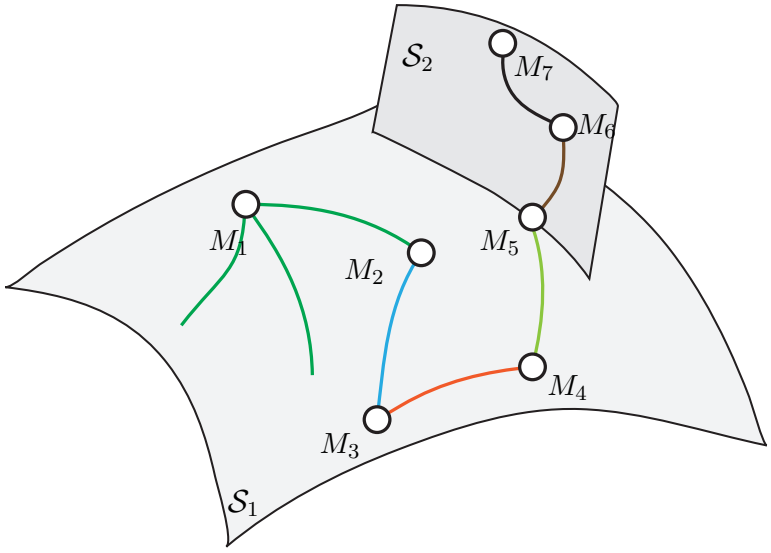


FIGURE 2.15 – Our algorithm discovers design variations in shape space. The user starts from a design M_1, and the algorithm seeks for wastage minimizing variations by interleaving between topologically different material layouts (indicated by changes in curved paths) or continuous changes to the layouts (indicated by same colored 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).

characterized 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 plank dimensions other than thickness. For a plank 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 optimization algorithm, alongside the design parameters $\mathbf{X}$ (see Section 2.2.3).

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 maximize utilization of the material space, the material space variables become tightly coupled with the design parameters. Our layout optimizer therefore jointly optimizes for material space variables and design parameters to minimize wastage (see Section 2.2.3)

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

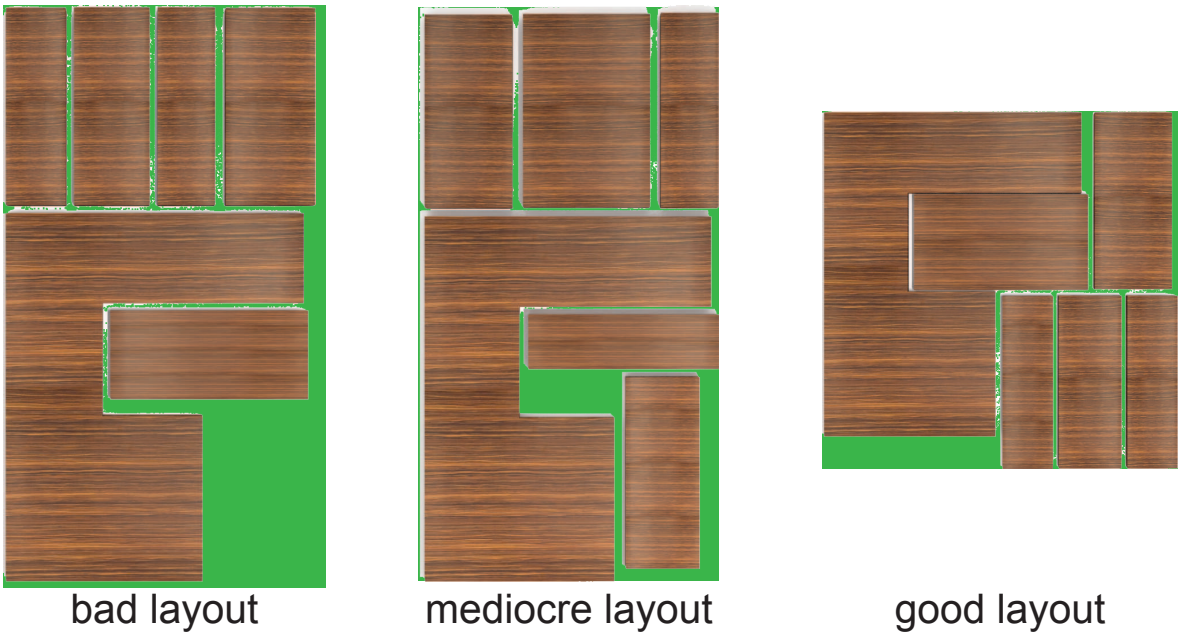


FIGURE 2.16 – Examples of stages of layout refinement, from bad to mediocre to good. A good layout is characterized by less area of material wasted (shown in green).

2.2.2.3 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 utilization of rectangular spaces, so that the user can use boards of exactly the right size and minimize wastage. The machine dimensions determine the maximum extent of a single board.

We measure wastage as the fraction of the space not utilized by the design in its material space bounding rectangle. Ideally, we want to achieve full utilization, 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 2.16).

2.2.3 Design Layout Optimization

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 optimization variable is the order in which the parts are sent to the docking algorithm. The second optimization 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.

Algorithm 2: MinWastage

Input: Design function D, starting design parameters $\mathbf{X}_s$
Output: Set of best layouts found $\mathcal{L}$

$O_s \leftarrow$ identity ordering ; `// 1,2,3,...`
$\mathcal{X} \leftarrow \{(\mathbf{X}_s, O_s)\}$;
for G *iterations* **do**
  foreach $(\mathbf{X}, O) \in \mathcal{X}$ **do**
    $\mathcal{O} \leftarrow$ ExploreOrderings$(\mathbf{X}, O)$;
    foreach $O \in \mathcal{O}$ **do**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{($ImproveDesign$(\mathbf{X}, O), O)\}$;
  $\mathcal{X} \leftarrow$ KeepBests(K,$\mathcal{X}$);
$\mathcal{L} \leftarrow \emptyset$;
foreach $(\mathbf{X}, O) \in \mathcal{X}$ **do**
  $\mathcal{L} \leftarrow \mathcal{L} \cup$ Docking$(D(\mathbf{X}), O)$;
return $(\mathcal{L})$;

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 2. The subroutine ImproveDesign is described in Section 2.2.3.1 while ExploreOrderings is described in Section 2.2.3.2. 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 2.15. The process returns the K best found layouts and designs and presents them to the user in thumbnails. She can then select her favorite design, and if desired update the constraints and restart the exploration from this point — which simply calls MinWastage again.

Algorithm 3: ImproveDesign

Input: Starting design parameters $\mathbf{X}$ and ordering O
Output: Modified design parameters $\mathbf{X}_b$ with reduced wastage

$L \leftarrow$ Docking(D$(\mathbf{X})$,O);
$\mathbf{X}_b \leftarrow \mathbf{X}, L_b \leftarrow L$;
$\mathbf{X}_c \leftarrow \mathbf{X}, L_c \leftarrow L$;
for N *iterations* **do**
  $\mathbf{X}_b, L_b \leftarrow$GrowParts$(\mathbf{X}_b, L_b, \mathbf{X}_c, L_c, O)$;
  $\mathbf{X}_c \leftarrow$ShrinkParts$(\mathbf{X}_b, L_b)$;
  $L_c \leftarrow$ Slide$(L_b, D(\mathbf{X}_c))$;
  `// Check for improvement over current.`
  if $W(L_c) < W(L_b)$ **then**
    $\mathbf{X}_b = \mathbf{X}_c$, $L_b = L_c$;
return $(\mathbf{X}_b)$;

Bitmaps During optimization we regularly call the parametrized design function $D(\mathbf{X})$ to obtain a new set of parts after changing parameters. The layout optimization represents parts internally as bitmaps : each part contour is rasterized 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 discretized into a regular grid of resolution τ.

2.2.3.1 Design optimization for wastage minimization

The design optimization improves the design parameters $\mathbf{X}$ to minimize wastage in the layout, keeping the docking ordering fixed. It appears as the subroutine ImproveDesign in Algorithm 2. The pseudo-code for this step is given in Algorithm 3. 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 2.2.3.1). 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 2.2.3.1). 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.

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 2.2.3.1) attempts to enlarge the parts 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 2.2.3.1). 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 analyze the layout to determine which parts have a higher probability to result in wastage reduction.

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 $\mathbf{s}(\mathbf{X})$ 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 $\Lambda = \mathbf{s}(\mathbf{X} + \Delta) - \mathbf{s}(\mathbf{X})$. In this process only the size s_i should change with others remain unchanged whenever possible, that is $\Lambda_{s_j, j \neq i} = 0$ 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 analyze 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 $\mathbf{\Lambda} = \sum_i \delta_i \cdot \frac{\partial \mathbf{s}(\mathbf{X})}{\partial x_i}$. 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-square sense, minimizing $||\mathbf{\Lambda} - (0, ..., \lambda, ..., 0)||^2$. If there are more parameters than part sizes, the problem is under-constrained and solved in the least-norm sense, minimizing $||\Delta||$. 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 $\mathbf{X}$ 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.

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 optimizing 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 favoring moves that result in the smallest layout bounding box. In case of a tie, we favor moves to the left/bottom versus displacements to the top/right. This is illustrated in Figure 2.17.

The pseudo-code is given in Algorithm 4. 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 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, favoring moves to the left

Algorithm 4: 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 masterboard`
      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);

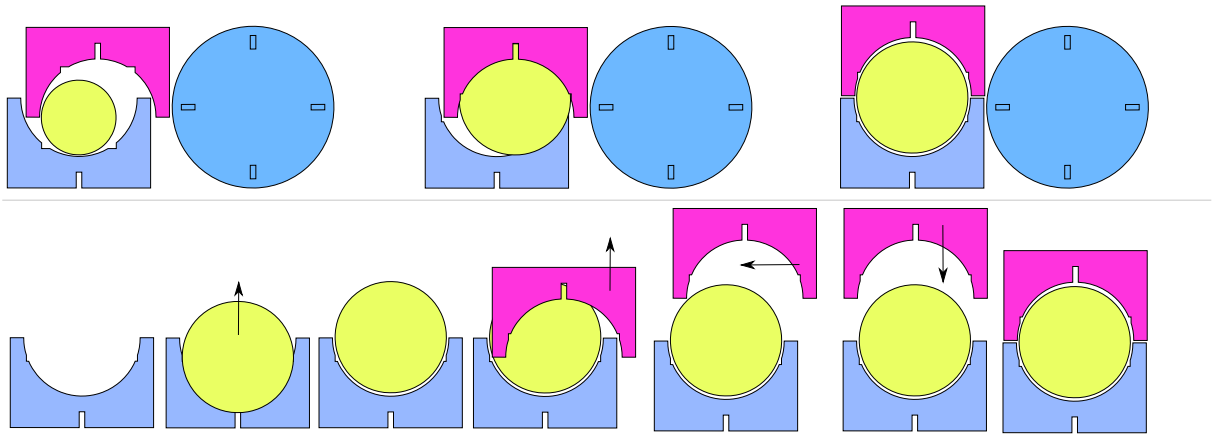


FIGURE 2.17 – Sliding a layout after a change of part sizes. **Top :** From left to right, initial layout, same after change revealing overlaps, layout after sliding. **Bottom :** Moves performed on the three first parts during sliding.

that collapse newly created empty spaces. Lines 4-7 similarly compute a vertical move. Lines 16-23 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 masterboard. 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.

Grow step The grow step is described in Algorithm 5. 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 continue the growth in cases were sliding fails to resolve overlaps by continuous changes. If wastage still not improves we stop the growth of this part size (line 15).

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 masterboard, and new opportunities for other parts to grow.

The overall approach is described in Algorithm 6. It first determines which parts to shrink by calling SELECTPARTSTOSHRINK and then computes a change of parameters using the approach described in Section 2.2.3.1.

Algorithm 5: 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.

$improvement \leftarrow true$;
while *improvement* **do**
  $improvement \leftarrow false$;
  foreach *part size* s_i *in random order* **do**
    $W_e \leftarrow 1$; `// max wastage`
    $\mathbf{X}_e \leftarrow \mathbf{X}_c$, $L_e \leftarrow L_c$;
    `// Grow a first time and then continue as long as it improves.`
    while *true* **do**
      $\mathbf{X}_e \leftarrow$ ChangePartSize($\mathbf{X_e}$,s_i,1) ; `// +1 pix.`
      $L_e \leftarrow$ Slide(L_e,$D(\mathbf{X_e})$);
      if $W(L_e) > W_e$ **then**
        $L_e \leftarrow$ Docking($D(X_e)$,O);
      if $W(L_e) < W_e$ **then**
        $W_e = W(L_e)$;
      else
        break;
    `// Check for improvement over current.`
    if $W_e < W(L_c)$ **then**
      $\mathbf{X}_c = \mathbf{X}_e$, $L_c = L_e$;
      $improvement \leftarrow true$;
  `// Check for improvement over global best.`
if $W(L_c) < W(L_b)$ **then**
  $\mathbf{X}_b = \mathbf{X}_c$, $L_b = L_c$;
return $(\mathbf{X}_b, L_b)$;

Algorithm 6: ShrinkParts

Input: Best design parameters $\mathbf{X}_b$ and layout L_b so far.
Output: Shrunk design parameters.

$\mathbf{X}_s \leftarrow \mathbf{X}$;
$\mathcal{S} \leftarrow$ SelectPartSizesToShrink(L_b);
$\Lambda \leftarrow (0, ..., 0)$;
foreach $s_i \in \mathcal{S}$ **do**
  $\Lambda_i \leftarrow -1$; `// -1 pixel`
$\mathbf{X}_s \leftarrow$ ChangePartSizes($\mathbf{X_s}$,Λ) ; `// -1 pixel`
return $(\mathbf{X}_s)$;

The core component is the SELECTPARTSIZESTOSHRINK subroutine, described in Algorithm 7. The selection starts by gathering all contacts between parts in the layout – this is done efficiently in the discretized layout grid. We first draw the part images into the grid and then check pairs of neighbors belonging to different parts. This produces the set of left/right and bottom/left 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 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.

2.2.3.2 Exploring orderings

The subroutine EXPLOREORDERINGS in Algorithm 2 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.

Algorithm 7: SELECTPARTSIZESTOSHRINK

Input: A layout L.
Output: Set of part sizes to shrink.

$\mathcal{K} \leftarrow \emptyset$;
foreach *axis* $a \in \{X, Y\}$ **do**
  $\mathcal{C} \leftarrow$GATHERCONTACTSALONGAXIS(a) ;
  $\mathcal{K} \leftarrow \mathcal{K} \cup$ FORMCONTACTCHAINS($\mathcal{C}$) ;
$\mathcal{S} \leftarrow \emptyset$;
while $\mathcal{K} \neq \emptyset$ **do**
  $s_i \leftarrow$ DRAWPARTSIZEWITHPROBABILITY($\mathcal{K}$);
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}$;
  $\mathcal{K} \leftarrow \mathcal{K} \setminus$ KILLEDCHAINS($\mathcal{K}$,s_i);
return $\mathcal{S}$;

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 placement of each part, according to a criterion that minimizes 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 $L_i \leftarrow L_{i-1} \lhd_{pos} p_i$ 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 pseudo code for the docking algorithm is given in Algorithm 8. 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 2.18. 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.

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 indeed better than others. If the concavity faces an already placed object, then further docking

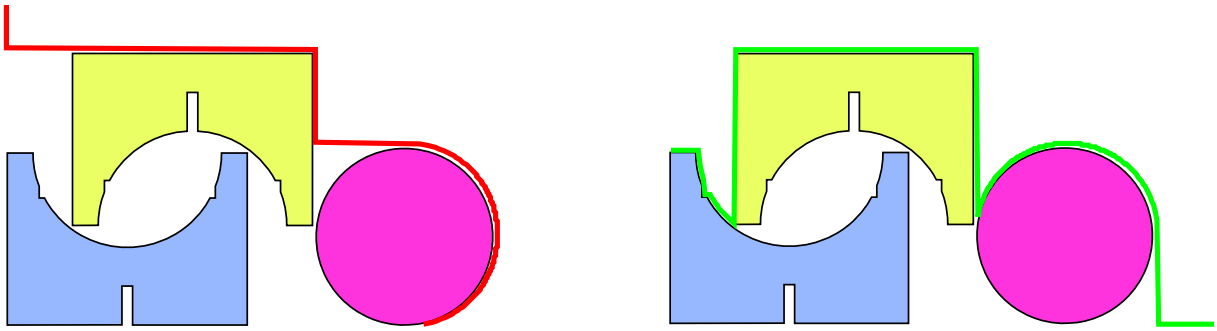


FIGURE 2.18 – Height-fields of the layout used to position the next part. **Left :** Height-field for dropping parts from the right (red curve). **Right :** Height-field for dropping parts from above (green curve). These height-fields are maintained every time a new part is added to the layout, and used for fast computation of the docking positions. Similar height-fields are pre-computed for the left/bottom of the parts.

Algorithm 8: DOCKING

Input: Set of parts P, order O, master board dimensions $W \times H$
Output: A layout L

foreach *part* $p_i \in P$ *following order in* O **do**
  $best \leftarrow \emptyset$;
  $bestscore \leftarrow 1$;
  foreach *drop location* (s, x, o) **do**
    $pos \leftarrow$ `ComputeDockingPosition`$(p_i, (s, x, o))$;
    $score \leftarrow D(L_{i-1}, p_i, pos)$;
    if $score < bestscore$ **then**
      $best \leftarrow pos$;
      $bestscore \leftarrow score$;
  $L_i \leftarrow L_{i-1} \lhd_{pos} p_i$;
return L_n;

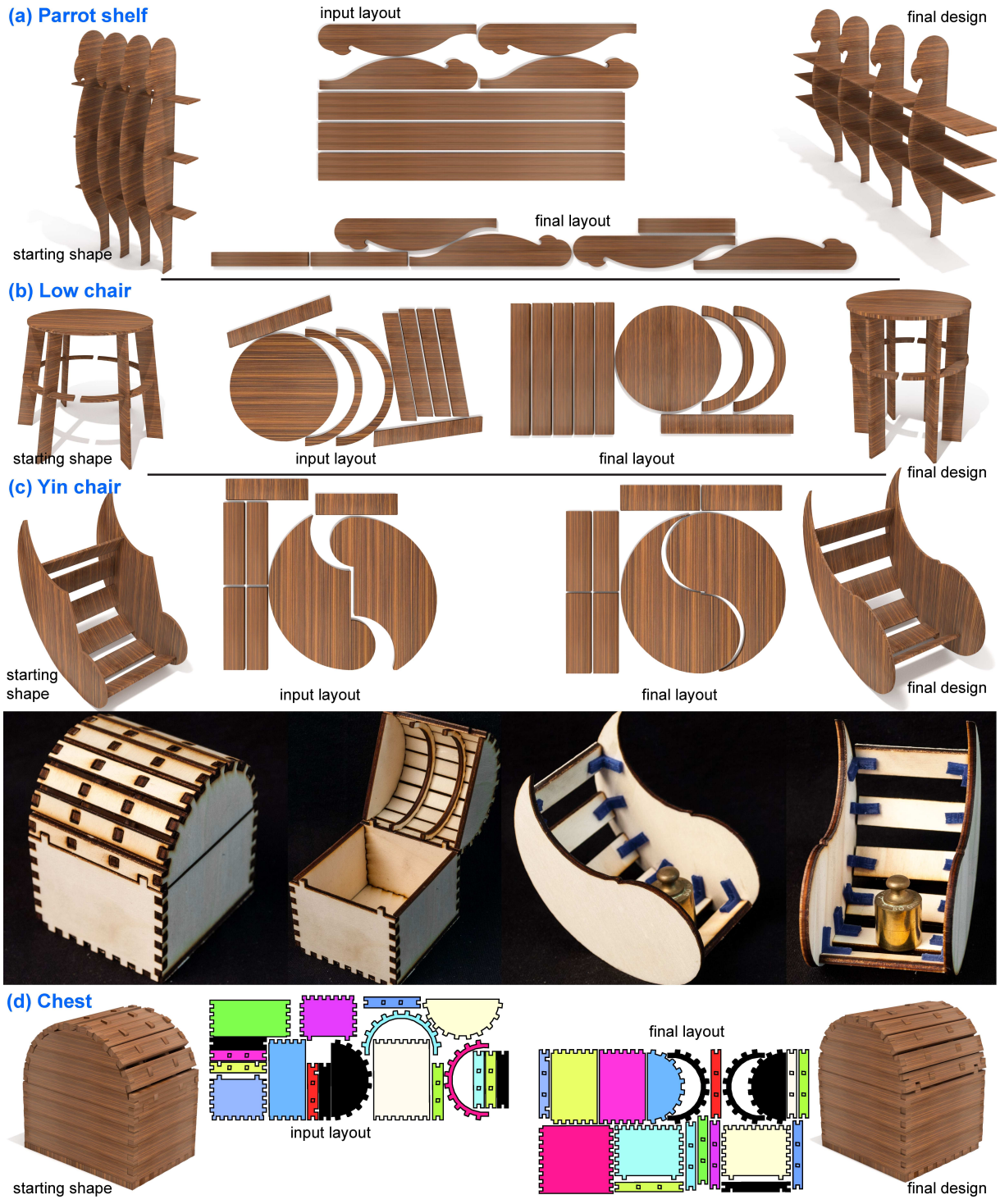


FIGURE 2.19 – Designs created using our system. Each design is shown with initial shape, starting layout, optimized layout, and final design.

within the concavity will never be possible. This is illustrated in Figure 2.20, 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 2.20.

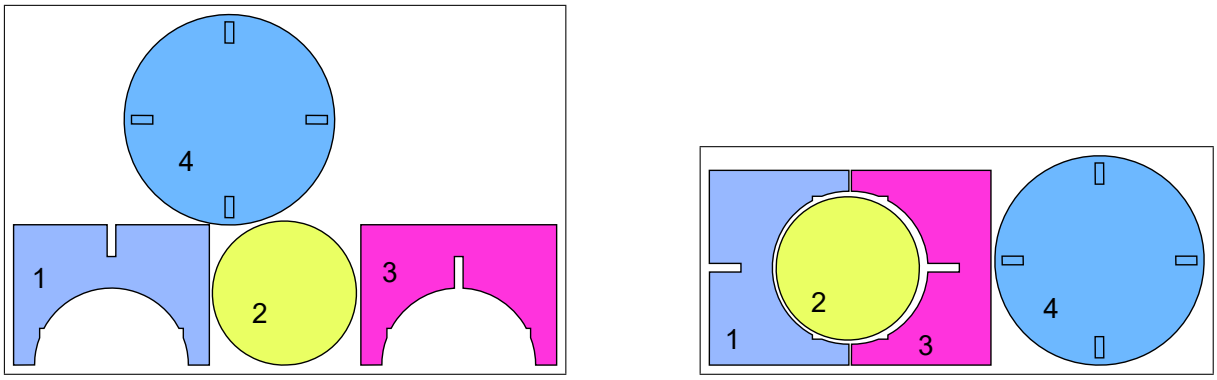


FIGURE 2.20 – Two layouts obtained with the same docking order. **Left :** Without taking enclosed area into account the first part is placed with the concavity against the bottom packing border. This prevents the second part to nest within and cascades into a series of poor placements. **Right :** Taking 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.

2.2.4 Results

We used our system for various design explorations. As the complexity of the designs grows beyond 4-6 planks, the utility of the system quickly becomes apparent. Note that the design constraints (see Figure 2.21), by coupling different object parts, make the optimization challenging by preventing independent adaptation of part sizes. By off-loading 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 utilization often increases significantly.

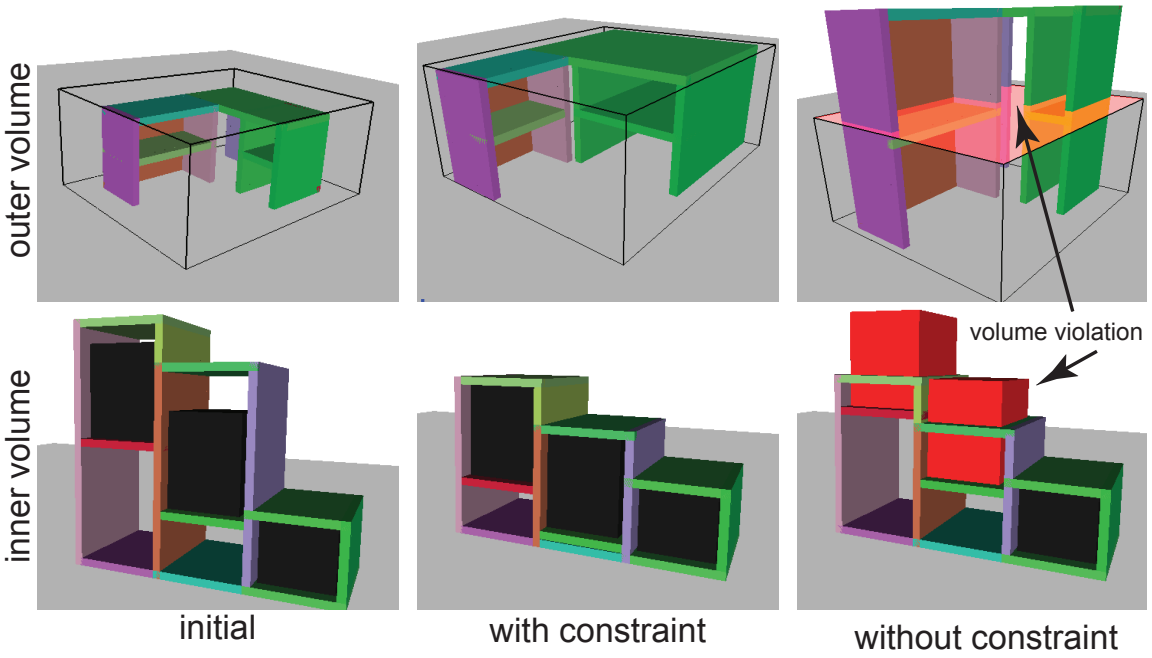


FIGURE 2.21 – We show effects of designing with (middle column) or without (right column) the respective constraints activated.

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

We directly output the cutting plan for the laser cutter (or CNC machine) from the design layout, adding connectors for planks sharing an edge, if needed. These are conveniently detected since planks 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 planks, or dowel-jointed for thicker materials (20 mm and 30 mm thickness).

Figures 2.19 and 2.22 show various results. Table 2.1 gives an overview of the complexity of each model, and the gains obtained by the layout optimizer. The system performs at interactive rates on a laptop taking from a few seconds to 3-4 minutes for the larger examples. Note that speed depends on how many exploration threads are pursued.

Figures 2.13 and 2.19 show results for objects with curved parts. Figure 2.14 shows some intermediate shapes as the design evolves for the coffee-table (Figure 2.13) and the low-chair (Figures 2.19-top) examples. Figure 2.23 shows alternate designs discovered by the algorithm for the Parrot shelf. While they have slightly lower usage they offer interesting variations that the user might prefer.

Figures 2.13 was fabricated using a CNC machine. The optimized design achieved

FIGURE 2.22 – Various material-driven design and fabrication examples. In each row, we show initial design (with material space layout inset), optimized design result (with material space layout inset), along with final cutout assembled model. Note that the design changes are often subtle, but still leads to significant improvement in material usage.

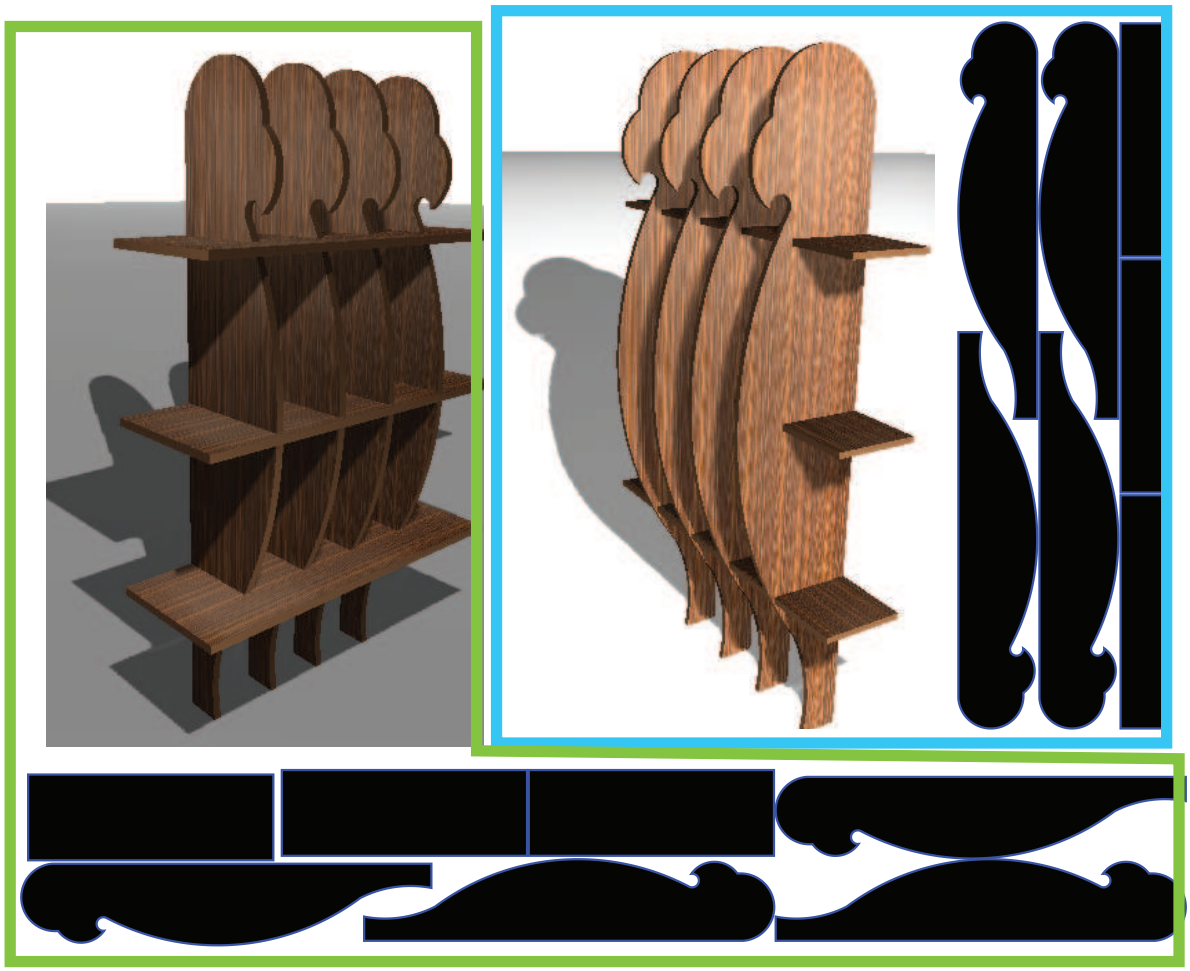


FIGURE 2.23 – Two different design suggestions (green has ratio 0.86, blue has ratio 0.85) for the parrot-shelf. Original design with another design suggestion is shown in Figure 2.19.

TABLE 2.1 – *Statistics for cut design showing the number of planks, number of constraints, material usage ratio before and after the design suggestions/optimization.*

	#planks	ratio before	ratio after
Figure 2.13	4	0.78	0.89
Figure 2.19a	7	0.66	0.92
Figure 2.19b	9	0.66	0.80
Figure 2.19c	8	0.76	0.83
Figure 2.19d	16	0.79	0.86
Figure 2.22a	6	0.85	0.96
Figure 2.22b	11	0.85	0.97
Figure 2.22c	8 3	0.74	0.97
Figure 2.22d	16	0.89	0.98

nearly 90% material usage, although one can achieve null wastage by deciding to pick a rectangular top – a decision that can be made after layout optimization as this opportunity is revealed. An allowable range was specified for the height and the bases were marked as symmetric as input design constraints. In the case of the parrot-shelf (Figure 2.19a), the user indicated minimum and maximum range for the horizontal shelves along with desired range for the shelf heights.

As described, parameteric designs are easily supported and optimized for in our framework. Figures 2.19b-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 optimize for.

Figure 2.22a shows a L-shaped work table. The user specified a target height for the design and a maximum work volume. Note that the legs of the table were also constrained to not change more than 25% of original dimensions to prevent unwanted design changes. Figure 2.22b shows a coupled shelf and table design where height of shelves and tabletop were similarly constrained. Figure 2.22c shows a stylized chair, where both the chair seat height and chair width were constrained not to change beyond a margin. Figure 2.22d 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.

Comparison We now evaluate the relative importance of the key algorithm steps. Figure 2.24a shows the importance of the docking criteria introduced in Section 2.2.3.2. 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 optimization is performed here) and plot the two conditions. The docking criteria consistently resulted in 10-15% better usage.

Figure 2.24b shows usage improvement over one exploration run on the coffee-table sequence. The legend explains which step (grow, shrink, etc.) is being performed. 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.

Figure 2.24c-d compare the importance of analyzing the material space layout to decide which plank to change and how. As baseline, we selected planks at random and perform either a grow or shrink sequence with equal probability. Note that our method consistently outperforms the alternative approach.

Limitations Currently, the algorithm can only make topological changes only for parameteric models. This will be an interesting future direction to pursue for constrained models. Our docking approach cannot nest parts into holes of other parts, a more advanced algorithm would be required. A more material-induced restriction arises when the starting layout does not leave much space to optimize over. This effectively means that the degree of freedom for the design is low. Adding more planks does reduce this problem (by providing additional freedom). However, beyond 25-30 planks, the exploration of the shape space becomes slow as there are too many paths to explore. One option is to limit exploration to only a subset of planks at a time, but then again, very desirable design

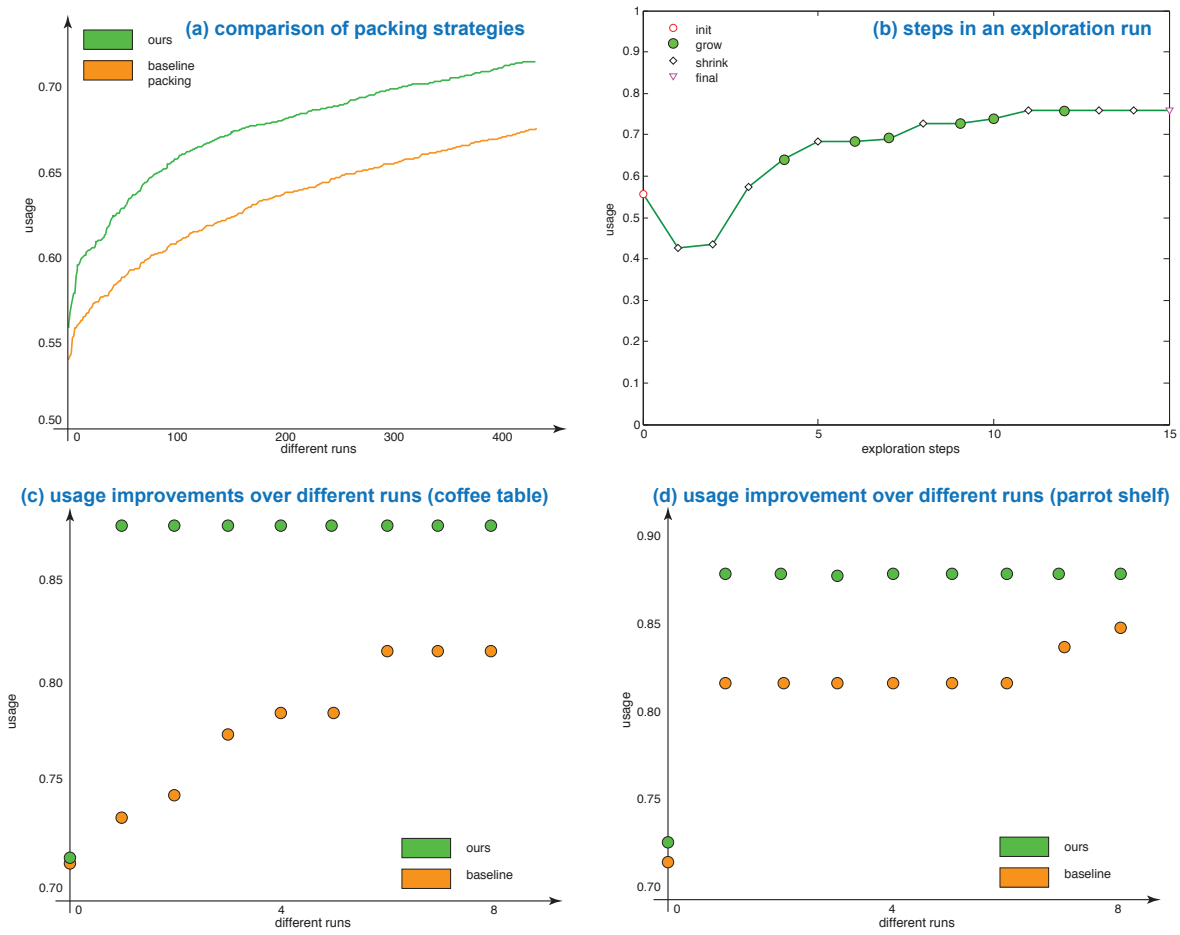


FIGURE 2.24 – Comparison of our algorithm against baseline alternatives. Higher is better. Please refer to the text for details.

configurations may be missed.

2.2.5 Conclusions and Future Work

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 analyzing 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 realized. 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, and hence difficult to account for. We used our system to generate a variety of shapes and demonstrated wastage reduction by 10% to 15%.

Interestingly, a first attempt was done with topology optimization, using the SIMP method to optimize planks. However, topology optimization is hard to tune for an inexperienced user. In this early attempt, the user needed to specify the rectangular planks that were optimized by the system. Topology optimization generated shape that are not simple to cut (see Figure 2.25), which is an additional problem.

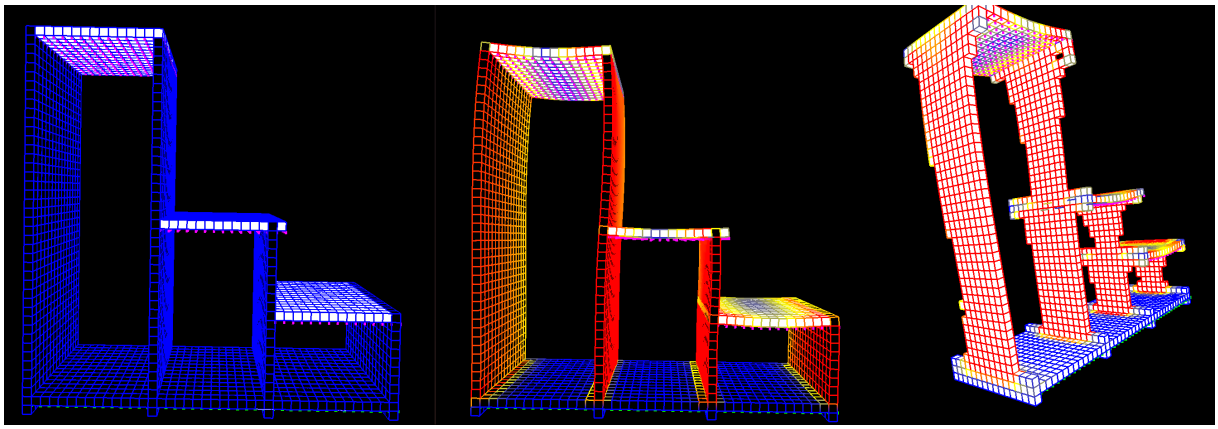


FIGURE 2.25 – **Left :** A furniture with three loaded horizontal planks. **Center :** The Finite Element Analysis on this shelf. **Right :** The results of the topology optimization. In this case, the shapes of the planks are constrained to be homeomorph with a cube.

Currently, we do not consider the stability of the produced furniture nor the durability of the joints. This could be integrated as dynamic constraints following previous work on structural reinforcement and shape balancing. Another important future direction is to generalize the framework to handle other types of laser cut materials, e.g., plastic plates that can be easily cut and more interestingly bend to have freeform shapes. Note that the packing problem will still be in 2D for such developable pieces. This can help produce interesting freeform shapes, while still making efficient use of materials.

Conclusion

Interestingly, during the work on waste minimization, we developed an interface to design simple furniture with rectangular planks. Once a complete UI was obtained, we confronted it to art school students(See figure 2.26). This has been an interesting experience : they had clear ideas about what they could do with this interface and they were particularly enthusiasts about the material optimization. However, the design tool was not easy to use for them, as it was too unusual in its interactions. They also lacked several features to further refine the designs. We thus decided to not further investigate the UI aspects and refocus on the optimization. This is however a very interesting direction of future work, but that requires a close collaboration with designers from the start.

The two methods that I presented in this chapter require at least some partial specification of the shape. It means that the user has to have a clear idea about what the shape looks like. However, this cannot always been expected. Therefore, I developed a synthesis system that automatically synthesize from functional specifications, in the context of furniture design. This system is described in the next chapter.

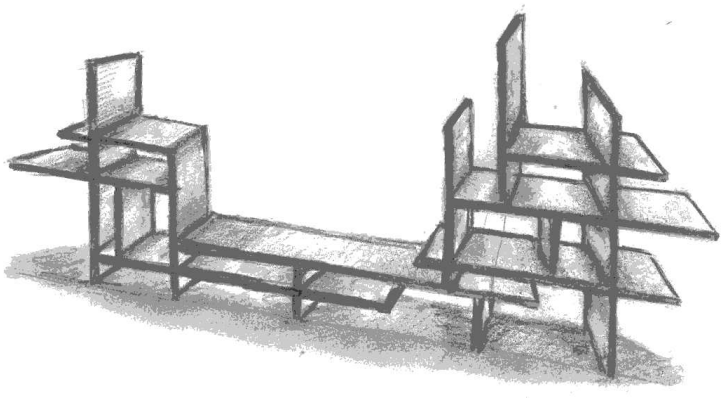
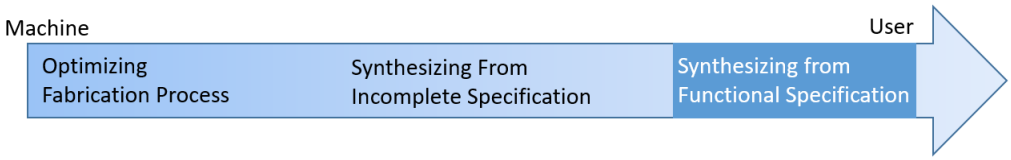


FIGURE 2.26 – One drawing from the art school student.

3

Synthesizing from Functional Specification

Introduction



The previous chapter details methods to optimize shapes – assemblies from planar cutouts and mechanisms – with respect to the fabrication process. In the case of assemblies from planar cutouts, our technique optimizes a parametric model to reduce the amount of wasted material.

In this chapter, we explore the synthesis of furniture from functional specifications; where the function is to support a set of loads arbitrarily positioned in space. Instead of defining the shape partially, the user only defines the function of the shape, and the system fully synthesizes it – we are at the extreme end of the spectrum of techniques I considered.

The optimization system presented in the chapter 2.2.5 required a parametric model of the design that the user wanted to fabricate. In contrast, the technique presented here operates similarly to topology optimization (see Section 4.1.3 of the state of the art), in the sense that it fully synthesizes a final shape that can support the specified loads. In this chapter, the shape is a shelf. It is more specialized than furniture; however shelves are everywhere. They can be easily manufactured at home with a saw and wooden planks. At the same time, besides the trivial cases, they are not simple to imagine and design. Modelling them to fit a particular environment, items to store and windows to avoid, *while* minimizing waste and work to fabricate them is a difficult task. Our system does exactly this. The shelves that are produced can have intricate, interesting layouts and

move beyond simple parallel shelves.

As all methods presented int Chapter 2, we formulate a combinatorial problem. It requires only a set of loads placed in space by the user, and it synthesizes a fabricable furniture that supports all the loads and is guaranteed to stand stably. Interestingly, this bares similarities to the problem of generating support structures in 3D printing [30] and some of the issues we face here are similar. Indeed the structure has to be self supported and at static equilibrium with or without the loads. Again, this work has been inspired by the Do It Yourself trend and we develop a technique that does not require expensive material or advanced manufacturing technologies.

This work has not been published yet. It is done in collaboration with Niloy Mitra and Sylvain Lefebvre.

3.1 Overview

The idea is the following : Knowing the position of user specified items in space, we want to synthesize a furniture that is at static equilibrium, is supporting all the items placed by the user, can be further edited, and use an as small as possible amount of wood. In addition, avoidance constraints can be specified to provide clearance for e.g. a window or door.

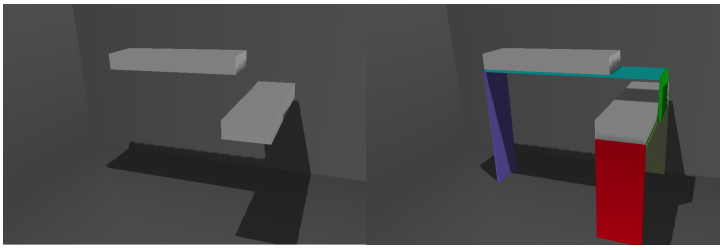


FIGURE 3.1 – **Left :** A Set of boxes in space placed by the user. **Right :** The synthesis is done by adding horizontal planks below each box and adding vertical planks to support the horizontal

The next section presents the user interaction. It also presents how the system maintains the topology of the furniture while the user is interacting to edit it. This is achieved by formulating a least norm problem on the design variables.

Then I explain how the synthesis is done, and in particular the **snapping** operation at its core. The snapping operation ensures that the static equilibrium is maintained. The synthesis algorithm has been tailored to use a small amount of planks.

3.2 User Interaction

In this section I describe the interaction allowed by the system. The user is able to place objects represented by their bounding boxes in space, to move and deform these bounding boxes (see Figure 3.2.1). The user also defines the size of the master boards he

will cut. As opposed to previous chapter, the master boards are planks, that will be cut in only one direction(see Figure 3.2.1).

3.2.1 User Interface

3D View The 3D view shows the user input as well as the furniture that has been synthesized by the system. In the 3D view, the user is able to select a plank and to move it or to change its size (see Figure 3.2.1). Once the furniture has been synthesized, she can move the boxes and preserve the same topology, or ask the system to synthesize the furniture again to obtain a variation of the current model. It can also leads to a topology change, since the user edit the input of the algorithm. The user is able to types of boxes : the first one represents the objects that she want to support, the second one are avoidance zones, that the synthesizer has to avoid.

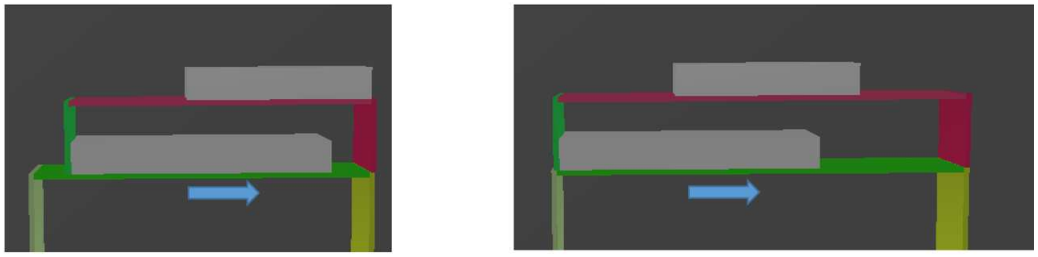


FIGURE 3.2 – **Left :** The user moves the center of the horizontal green plank. **Right :** The system maintains the connectivity between all the planks by changing the size and the position of all other planks.

Material Space We considered that the material space is composed by long wood planks that will be cut by the user. Indeed, those not require any special material to cut, The system shows the material space to the user.

3.2.2 Preserving topology during user edition

After synthesis, we could let the user edit individual planks. However, doing these changes without assistance would be difficult as changes would disconnect planks or result in self intersections. Instead, our algorithm helps the user maintain connectivity with a constraints based system described below. Throughout an edition session, the user is thus able to modify the furniture that has been synthesized, by smoothly deforming planks. The design globally adapts to these changes.

Notations A shelf **S** is represented by a set of planks **P** and a set of constraints **C**. A plank P_i is represented by its orientation $o_i \in \{Horizontal, Vertical\}$, its width direction $d_i \in \{X, Y\}$, its length l_i (thickness and width are fixed), and the position of their center

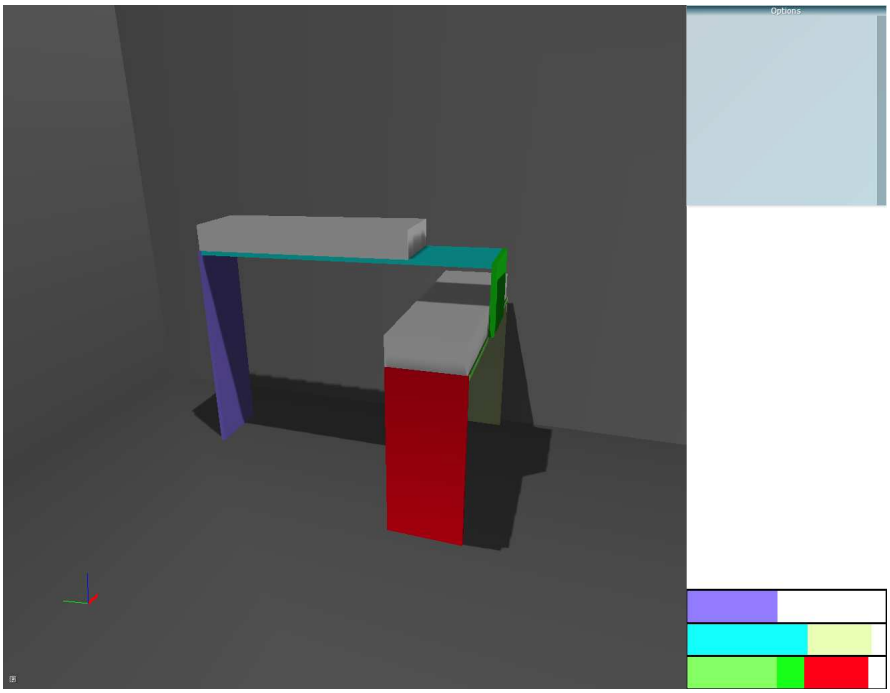


FIGURE 3.3 – This is the main view of the system. The user see the box that she places in space as well as the material space in the bottom right corner if the interface. The planks have the same color in the material space than on the 3D view.

in space p_i. We note $U(\mathbf{S})$ the vector constituted by the position and the length of all planks in $\mathbf{S}$.

Once the synthesis is done, the user is also able to deform the furniture, and the system maintains the topology. In order to maintain the feasibility during the user interaction, linear constraints are automatically defined during the synthesis of the furniture. Those constraints are under the form :

$$\sum_{i=0}^{N} A_i * p_i.x + B_i * p_i.y + C_i * p_i.z + D_i * l_i = K$$

Where A_i,D_i,C_i,D_i and K are constants defined during the synthesis and $N = Card(\mathbf{P})$. The user is able to smoothly edit the furniture one variable at a time (See Figure 3.2.1). We build the system by using all the constraints defined during the synthesis and a constraint based on the last user edition. If the user modifies the variable $p_i.x$ by δ, the constraints $p_i.x = p_i^0.x + \delta$ where p_i^0 is the previous value of p_i is added to the system.

Usually, the number of constraints (equations) in the system is greater than the number of variables. The said system is under determined and has an infinite number of solutions. However, to avoid non-intuitive moves in the solution, we minimize the distance between the furniture before the user interaction, and the furniture after the user interaction. This means that we want to minimize the following equation where $M(\mathbf{C})$ is the matrix defined by the constraints in $\mathbf{C}$:

$$||U(\mathbf{S}) - U(\mathbf{S}^0)||$$

Subject to :

$$M(\mathbf{C}) * U(\mathbf{S}) = k$$

However, by posing : $X(\mathbf{S}, \mathbf{S}^0) = U(\mathbf{S}) - U(\mathbf{S}^0)$ we can solve :

$$||X(\mathbf{S}, \mathbf{S}^0)||$$

Subject to :

$$M(\mathbf{C}) * X(\mathbf{S}) = K(\mathbf{C}) - M(\mathbf{C}) * U(\mathbf{S}^0)$$

This is a least norm problem and it has a direct solution. Thanks to the fact that it minimizes $||U(\mathbf{S}) - U(\mathbf{S}^0)||$, the changes done by the system are only on variables that appear in constraints that are impacted by the user edition.

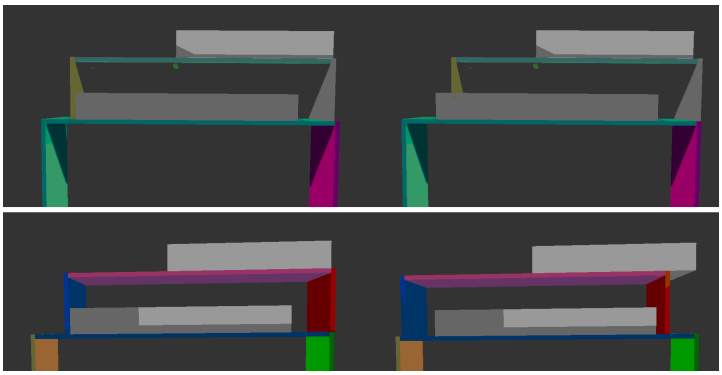


FIGURE 3.4 – Without feasibility constraints, the user edition can leads to an unwanted results, that still verify all other constraints. **Top :** The user move the center of the yellow plank, which collide with a box. **Bottom :** The user move the center of the pink plank, which doesn't support the top box

Feasibility constraints Some user edits may lead to planks intersecting with the boxes defined by the user (see Figure 3.4) – note however that this never happens after synthesis. Since the boxes represents the objects that will lay on the furniture, those collision have to be avoided. For example, the user is also able to add avoidance zones which are zones in witch there will not be any planks. Or, in order to maintain the equilibrium of the objects on the furniture, the planks that is supporting an object has to be bigger than the objects (see Figure 3.4).

Those constraints are not equality constraints and appear in the system only when it is necessary. They do not impact the topology of the furniture but they are necessary to ensure that the feasibility of the furniture. Our key idea is to add constraints dynamically in the system to prevent moves that impact design feasibility.

We observe that all the feasibility constraints can be enforced by dynamically adding or removing equality constraints. For instance, when a plank is about to hit an object a constraint can be inserted to maintain the plank 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 vector change $\mathbf{u}$ with $u_i = \delta$ on the variable that the user changes, $u_k = O$ everywhere else. That is, we parametrize the problem with a variable $\alpha \in [0, 1]$ and find the smallest value α_1 such that $U(\mathbf{S}) + \alpha_1\mathbf{u}$ verifies all design effectiveness constraints, but one or more are violated on $\alpha_1 + \epsilon$ for any $\epsilon > 0$. We then add an appropriate equality constraint to $\mathbf{C}$ such that all design effectiveness constraints are enforced. The algorithm then recurses until reaching $\alpha = 1$. 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 9.

The 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 plank collides an object of height H, it adds a constraint to keep the colliding plank at a distance of H of the plank supporting object. This is a very flexible approach allowing for a variety of feasibility constraints.

Algorithm 9: DynamicFeasibilitysConstraints

Input: Valid furniture $\mathbf{S}$, vector of changes $\mathbf{u}$, position $\alpha \leq 1$
Output: Valid furniture $\mathbf{S_c}$ such that $\mathbf{S_c}$ verify all effectiveness constraints.

$U(\mathbf{S}) \leftarrow$Solve$(\mathbf{C}, U(\mathbf{S}) + \mathbf{u})$;
if $U(\mathbf{S})$ *verify all effectiveness constraints* **then**
    `// Found a solution, return`
    return $\mathbf{S}$;
`// Search for first violation, bisection on` α
$\mathbf{l} \leftarrow \alpha$;
$\mathbf{r} \leftarrow 1.0$;
while $|l - r| > \epsilon$ **do**
    $\mathbf{m} \leftarrow (\frac{l+r}{2}) \cdot \mathbf{u}$;
    $U(\mathbf{S}) \leftarrow$Solve$(\mathbf{C}, U(\mathbf{S}) + m * \mathbf{u})$;
    if $U(\mathbf{S})$ *verify all feasibility constraints* **then**
        $l \leftarrow \frac{l+r}{2}$;
    else
        $r \leftarrow \frac{l+r}{2}$;
$\mathbf{C_2} \leftarrow$AddDynamicConstraints$(\mathbf{S}, \mathbf{C})$;
if $\mathbf{C_2} = \mathbf{C}$ **then**
    `// No constraint could be added: return best found so far`
    return $U(\mathbf{S})$;
$\mathbf{C} \leftarrow \mathbf{C_2}$;
`// recurse`
return DynamicEffectivenessConstraints$(\mathbf{S}, \mathbf{u}, l)$

3.3 Synthesis

As we saw in the previous section, the synthesizer does not not only synthesize the furniture but it also generates a set of constraints to maintain the feasibility of the shelf while the user is interacting. This section describe the snapping operation and the constraints that are produced by it, as well as the overall synthesis algorithm.

3.3.1 Synthesis algorithm

First, the system add a plank below each item specified by the user. The orientation depend of the size of the item along X and Y axis. The idea is to snap all horizontal planks together. If they have the same altitude, we snap them by their edges or add vertical planks to snap an above plank to a plank below, again this can length the horizontal planks. During this process several possibilities are considered, scored and the best is kept at each iteration. Vertical planks have to touch the ground or to have their bottom part on an horizontal planks. An overview of the algorithm is presented in Algorithm 1.

Algorithm 10: SYNTHESIS

Input: A set of boxes D
Output: A shelf S

```
1  while S is not valid do
2      Cost ← ∞
3      BestOperation ← NoOperation
4      foreach pair (Edge,Plank) in (E, P) do
5          CurCost ← Snapping(E, P)
6          if CurCost < Cost then
7              Cost ← CurCost
8              BestOperation ← Snapping(E, P)
9      foreach Edge E do
10         CurCostPillar(E))
11         if CurCost < Cost then
12             Cost ← CurCost
13             BestOperation ← Snapping(E, Ground)
14     if Cost < costMax then
15         apply BestOperation
```

Generating Variations The user can ask to the system to generate variation of the topology of the current shelf. It is done by adding randomness in the choice of the snapping operation, instead of taking the one of smallest cost. The system still chooses the snapping that are adding the less wood with higher probability.

3.3.2 Snapping operation

The algorithm is based on an operation called snapping, which snaps the edge of an horizontal plank to the edge of another horizontal plank, possibly introducing vertical planks to do so.

The snapping is tailored to ensure stability and to produce furnitures using few planks. It also generates the constraints that maintain the feasibility during the user manipulation. The snapping between an edge e an a plank p is possible under some precise conditions. The edge has to be higher than the plank (noting the edge e, this give $se.Z < p.z$). It is only possible to snap planks that are aligned as seen from above, or that are perpendicular to one another.

The cost of the snapping operation is the quantity of wood added by growing the plank and adding the vertical plank.

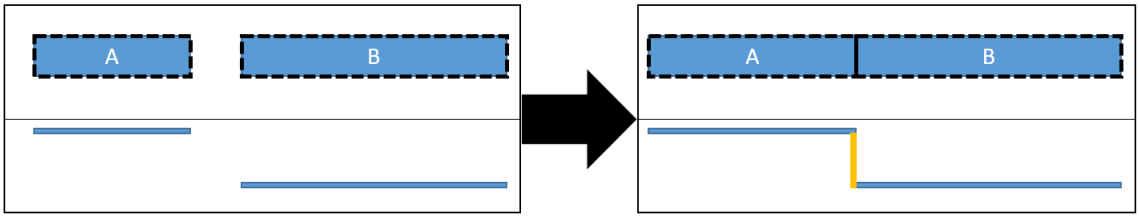


FIGURE 3.5 – Snapping of left edge of plank A on plank B, with two parallel planks. The operation extends the size of A, to the right Edge of B and add an Vertical plank(in yellow) between A and B. **Top)** Top View, **Bottom)** Side View.

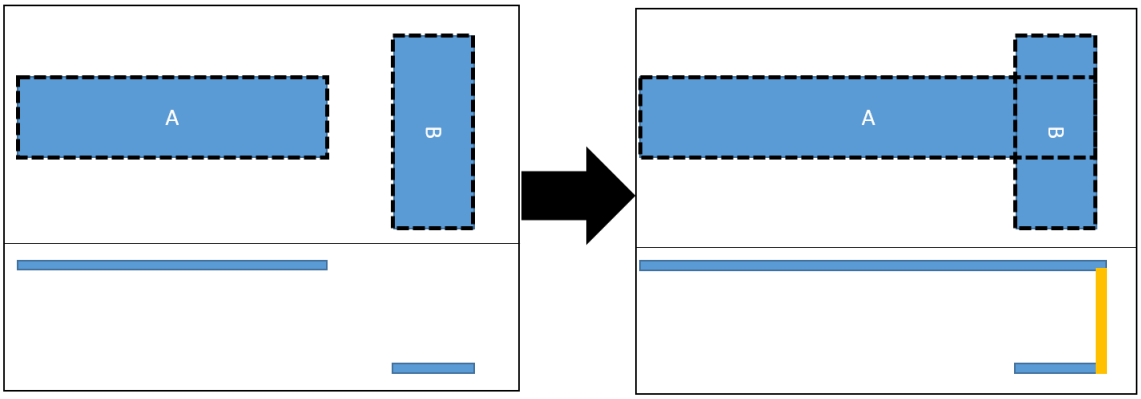


FIGURE 3.6 – Snapping of Left Edge of plank A on plank B, with two perpendicular planks planks. The operation extends the size of A, to the left Edge of B to ensure static equilibrium. **Top)** Top View, **Bottom)** Side View.

Avoiding collisions The snapping tries to avoid collisions by extending the planks along the edges if it is possible (See figure 3.7 and 3.7) . If it is not possible the snapping operation is denied by the algorithm (it is given an infinite cost).

Constraints The snapping operation generates constraints on the planks depending on their relative orientations after snapping. The constraints are added to maintain the connectivity during user edition. The constrained are detailed in section 3.2.2

Adding Vertical Planks If an edge is snapped to a plank that is not at the same altitude, the system add a vertical plank between this edge and the plank. The system add a constraints to maintain this snapping despite user edition. The system also try to snap the edge of a plank to the ground, and compute the corresponding cost.

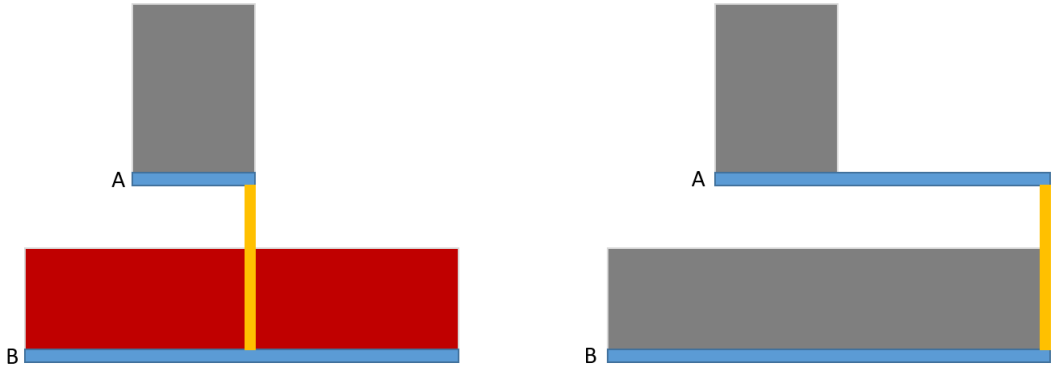


FIGURE 3.7 – Snapping of Left Edge of plank A on plank B. In this case, the vertical planks is intersecting with the object on plank B(Left). The algorithm extends both planks to avoid the collision.

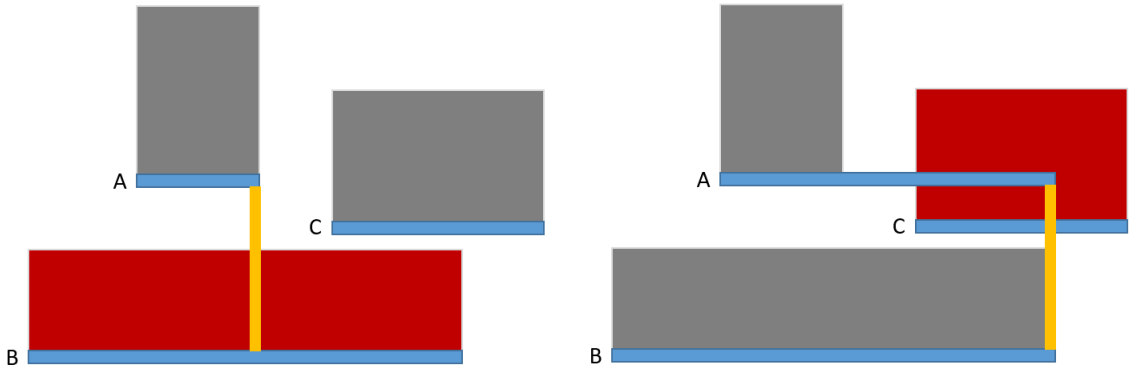


FIGURE 3.8 – Snapping of Left Edge of plank A on plank B. In this case, the vertical planks is intersecting with the object on plank B(Left). There is no way to snap the left edge of A without intersecting a box. This snapping is impossible and not considered during optimization

3.4 Results

We presents a variety of results to show the efficiency of every part of our algorithm. Figure 3.4 and 3.4 shows the effect of the avoidance zone placed by the user. Using avoidance zone allows her to cope with room with more complex geometry (e.g. a room with a pillar in the middle, Figure 3.4).

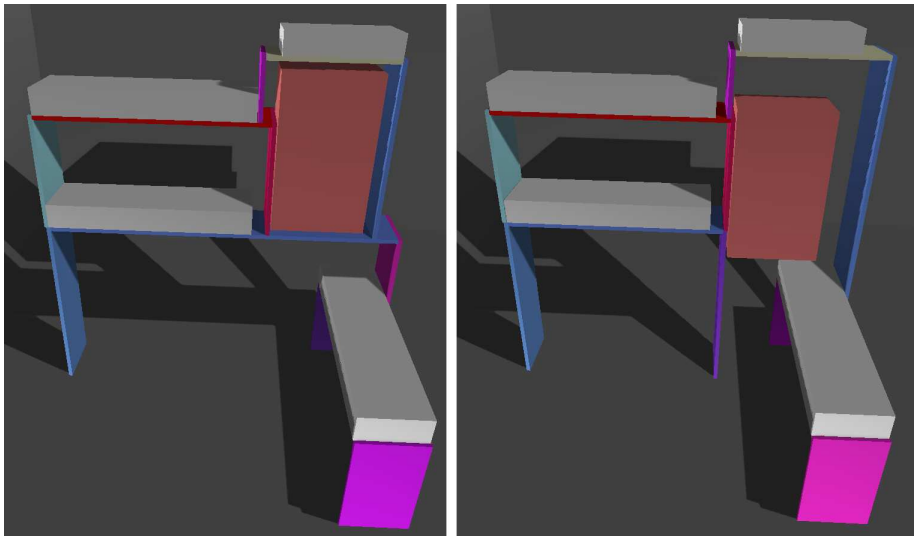


FIGURE 3.9 – The user moves the avoidance zone (red box) and ask for the system to synthesize again. In both cases, the system carefully select the snapping operation to avoid this zone.

Conclusion

We presented a technique that synthesizes a furniture from functional specifications. A shelf is synthesized by using a set of objects placed in space by the user. The user is allowed to edit the shelf after synthesis, and the system maintains its feasibility with a constraint system automatically built during synthesis.

It is important to let the user edit the furniture to give her back the control over the shape that she wants to fabricate. Designing from scratch is hard to do for a novice, but our system provides and initial complete answer that the interaction system helps easily edit through simple drag and drop.

Limitations and Future Work

This work is limited to the synthesis of shelves but they are omnipresent, and often have to be adapted to fit difficult spaces and/or items. We did not consider the synthesis of other type of furniture (e.g. table, chair...). We do not study the aesthetics impact that

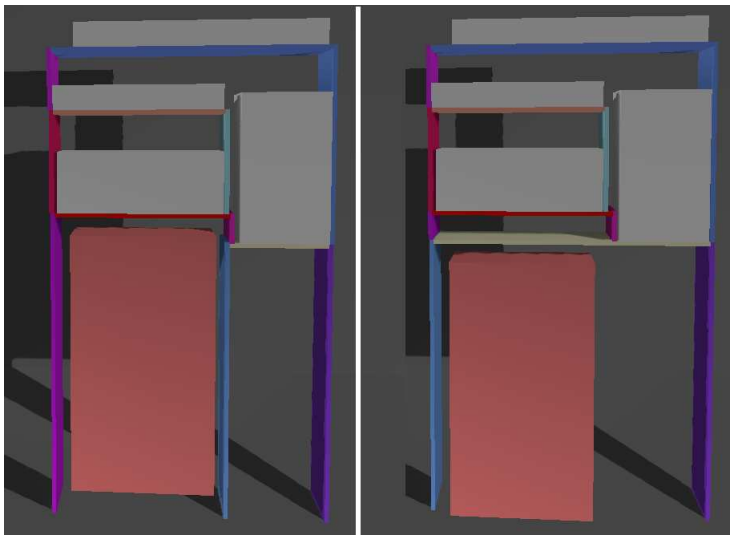


FIGURE 3.10 – The user moves the avoidance zone (red box) and ask for the system to synthesize again. In both cases, the system carefully select the snapping operation to avoid this zone.

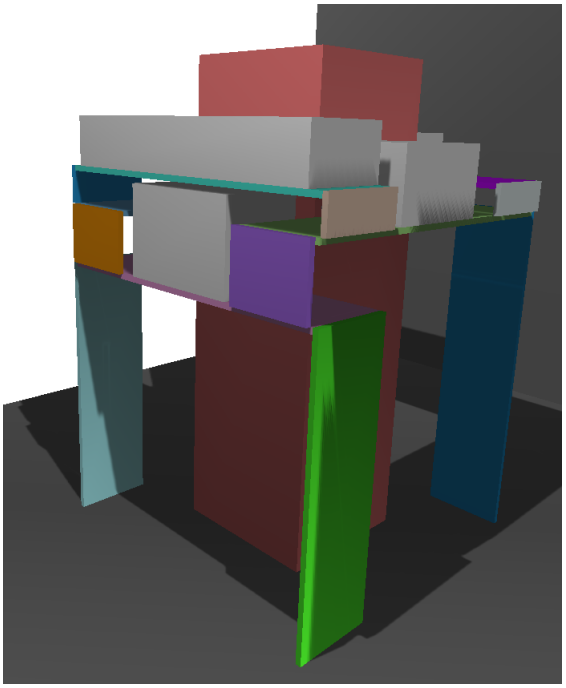


FIGURE 3.11 – The user places a pillar in red in the middle of the room, and places objects around.

the synthesized furniture might have – however having the user in the loop helps him guide the system towards a desirable result.

The previous chapter of this thesis describes a technique that automatically explores

possible designs to optimize the material space. We can imagine a similar technique based on the linear constraint system that would optimize the shelves for material wastage. The current system does not allow the user to change the topology of the furniture (only the synthesizer can do it). This would be an interesting feature to add.

Conclusion

Throughout this thesis I considered the problem of modeling objects for fabrication in different ways, by proposing algorithm that increasingly help the user with the design. My goal was to show that those modeling problems can be approached with different methodologies depending on the the profile of the user who wants to model the object, an what her intent and technical skills are.

The first part was tailored to optimize only the fabrication process. It is helpful for users who are able and are willing to use complex modeling tools, and who want to fabricate their models as close as possible to their virtual counterparts.

If the evolution of rapid manufacturing technology is as fast as the evolution of computers (e.g. by following some kind of Moore's law [72]), the price of today's state-of-the-art technologies will drop and this will make them accessible. Therefore the work based on a particular technology might become less relevant. However, helping the user to design will remain interesting, and considering fabrication and functional constraints will always be part of the process : the geometry, material and fabrication parameters of an object are all tightly coupled to its final intended use.

Modeling is a very difficult task for most casual users who are not trained to use a specific software. In this case I believe that the system has to support some part of the design process. In a sense, letting the system handle the fabrication constraints is a way to prune the design space. This pruning not only helps the user to explore the design space more efficiency, but it allows her to design *only* shape that are fabricable.

Following this trend, I considered how to let the algorithm handle various aspects of the design, up to the fully automated synthesis technique that is developed in the last chapter.

Future Work. There are some future works that might be interesting to explore. For example, instead of optimizing a parametric model of furniture to minimize the material wastage, we can imagine an interface that guides the user to the same objective during the modeling process. It would achieve the same purpose that the optimization method discussed in Section 2.2 (reducing wastage), but it would give the user a more direct control over the final shape. Also, we can imagine sketch-based modeling tools that take into account the physical and fabrication constraints as priors to help with the reconstruction of a 3D model from the sketch. These future works are both user oriented and serve the same purpose : Using fabrication constraints as a guide to modeling.

I believe that synthesis cannot replace an artist with her style, her feelings and her way to design, and the results that are shown in the last section are in no way pieces of art. They, however, can help in the first stages of the design process, or can help a user produce a furniture that *function* as intended, when aesthetics are a secondary concern. I also believe that designers will always be confronted to the reality of fabrication and will always benefit from tools that help them handle the constraints that they are facing.

Bibliographie

[1] P. Alexander, S. Allen, and D. Dutta. Part orientation and build cost determination in layered manufacturing. *Computer-Aided Design*, 30(5), 1998.

[2] D. G. Aliaga, I. Demir, B. Benes, and M. Wand. Inverse procedural modeling of 3d models for virtual worlds. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, pages 16 :1–16 :316, New York, NY, USA, 2016. ACM.

[3] S. Allen and D. Dutta. Determination and evaluation of support structures in layered manufacturing. 1995.

[4] J. W. Allison, T. P. Chen, A. L. Cohen, D. R. Smalley, D. E. Snead, and T. J. Vorgitch. Boolean layer comparison slice, 1988. US Patent 5854748, 3D Systems Inc.

[5] M. Attene. Shapes In a Box : Disassembling 3D Objects for Efficient Packing and Fabrication. *Computer Graphics Forum*, 2015.

[6] M. Bächer, E. Whiting, B. Bickel, and O. Sorkine-Hornung. Spin-it : Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph.*, 33(4) :96 :1–96 :10, July 2014.

[7] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, 1 :192–202, 1989.

[8] J. A. Bennell and J. F. Oliveira. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, 60(1) :S93–S105, 2009.

[9] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 417–424, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[10] D. Beyer, S. Gurevich, S. Mueller, H.-T. Chen, and P. Baudisch. Platener : Low-fidelity fabrication of 3d objects by substituting 3d print with laser-cut plates. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1799–1806, New York, NY, USA, 2015. ACM.

[11] P. Bhat, S. Ingram, and G. Turk. Geometric Texture Synthesis by Example. In R. Scopigno and D. Zorin, editors, *Symposium on Geometry Processing*. The Eurographics Association, 2004.

[12] M. Bokeloh, M. Wand, and H.-P. Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.*, 29(4) :104 :1–104 :10, July 2010.

[13] J. Calì, D. A. Calian, C. Amati, R. Kleinberger, A. Steed, J. Kautz, and T. Weyrich. 3d-printing of non-assembly, articulated models. *ACM Trans. Graph.*, 31(6) :130 :1–130 :8, Nov. 2012.

[14] K. Castelino and P. K. Wright. Tool-path optimization for minimizing airtime during machining. *Journal of Computing and Information Science in Engineering*, 2004.

[15] D. Ceylan, W. Li, N. J. Mitra, M. Agrawala, and M. Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics*, 32(6) :186 :1–186 :11, 2013.

[16] K. Chalasani, L. Jones, and L. Roscoe. Support generation for fused deposition modeling. In *Solid Freeform Fabrication Symposium*, pages 229–241, 1995.

[17] S. Chaudhuri, E. Kalogerakis, L. Guibas, and V. Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph.*, 30(4) :35 :1–35 :10, July 2011.

[18] X. Chen, H. Zhang, J. Lin, R. Hu, L. Lu, Q. Huang, B. Benes, D. Cohen-Or, and B. Chen. Dapper : Decompose-and-pack for 3d printing. *ACM Trans. Graph.*, 34(6) :213 :1–213 :12, Oct. 2015.

[19] X. Chen, C. Zheng, W. Xu, and K. Zhou. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph.*, 33(4) :95 :1–95 :11, July 2014.

[20] S. Choi and W. Zhu. A dynamic priority-based approach to concurrent toolpath planning for multi-material layered manufacturing. *Computer-Aided Design*, 42(12) :1095 – 1107, 2010.

[21] S. H. Choi and H. H. Cheung. A topological hierarchy-based approach to toolpath planning for multi-material layered manufacturing. *Comput. Aided Des.*, 38(2) :143–156, Feb. 2006.

[22] A. N. Christiansen, J. A. Bærentzen, M. Nobel-Jørgensen, N. Aage, and O. Sigmund. Combined shape and topology optimization of 3d structures. *Computers & Graphics*, 46 :25 – 35, 2015. Shape Modeling International 2014.

[23] P. Cignoni, N. Pietroni, L. Malomo, and R. Scopigno. Field-aligned mesh joinery. *ACM Trans. Graph.*, 33(1) :11 :1–11 :12, Feb. 2014.

[24] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithms for Bin-Packing — An Updated Survey*, pages 49–106. Springer Vienna, Vienna, 1984.

[25] S. Coros, B. Thomaszewski, G. Noris, S. Sueda, M. Forberg, R. W. Sumner, W. Matusik, and B. Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics*, 32(4) :83 :1–83 :12, 2013.

[26] P.-E. Danielsson. Euclidean Distance Mapping. *Computer Graphics And Image Processing*, 14, 1980.

[27] C. De Paoli and K. Singh. Secondskin : Sketch-based construction of layered 3d models. *ACM Trans. Graph.*, 34(4) :126 :1–126 :10, July 2015.

[28] H. Q. Dinh, F. Gelman, S. Lefebvre, and F. Claux. Modeling and toolpath generation for consumer-level 3d printing. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, pages 17 :1–17 :273, 2015.

[29] A. Dolenc and I. Mäkelä. Slicing procedures for layered manufacturing techniques. *Computer-Aided Design*, 26(2) :119–126, 1994.

[30] J. Dumas, J. Hergel, and S. Lefebvre. Bridging the gap : Automated steady scaffoldings for 3d printing. *ACM Trans. Graph.*, 33(4) :98 :1–98 :10, July 2014.

[31] G. Eggers and K. Renap. Method and apparatus for automatic support generation for an object made by means of a rapid prototype production method, 2007. US Patent 20100228369, Materialize.

[32] R. T. Farouki. Exact offset procedures for simple solids. *Comput. Aided Geom. Des.*, 2(4) :257–279, Dec. 1985.

[33] M. Fischetti and I. Luzzi. Mixed-integer programming models for nesting problems. *Journal of Heuristics*, 15(3) :201–226, June 2009.

[34] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3) :652–663, Aug. 2004.

[35] W. Gao, Y. Zhang, D. C. Nazzetta, K. Ramani, and R. J. Cipra. Revomaker : Enabling multi-directional and functionally-embedded 3d printing using a rotational cuboidal platform. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 437–446. ACM, 2015.

[36] I. P. Gent, C. Jefferson, and I. Miguel. Minion : A fast, scalable, constraint solver. In *Proceedings of ECAI 2006*, pages 98–102, 2006.

[37] E. Heide. Method for generating and building support structures with deposition-based digital manufacturing systems, 07 2011. US Patent 20110178621 A1.

[38] J. Hergel and S. Lefebvre. Clean color : Improving multi-filament 3d prints. *Comput. Graph. Forum*, 33(2) :469–478, May 2014.

[39] J. Hergel and S. Lefebvre. 3d fabrication of 2d mechanisms. *Comput. Graph. Forum*, 34(2) :229–238, May 2015.

[40] K. Hildebrand, B. Bickel, and M. Alexa. Crdbrd : Shape fabrication by sliding planar slices. *Comput. Graph. Forum*, 31(2pt3) :583–592, May 2012.

[41] K. Hildebrand, B. Bickel, and M. Alexa. Orthogonal slicing for additive manufacturing. *Computers and Graphics*, 37(6) :669–675, 2013.

[42] S. Hornus, S. Lefebvre, J. Dumas, and F. Claux. Tight printable enclosures for additive manufacturing. Research Report RR-8712, Inria, Apr. 2015.

[43] K. Hu, S. Jin, and C. C. Wang. Support slimming for single material based additive manufacturing. *Computer-Aided Design*, 65 :1 – 10, 2015.

[44] P. Huang, C. C. Wang, and Y. Chen. *Algorithms for Layered Manufacturing in Image Space*. ASME Press, 2014.

[45] X. Huang, C. Ye, S. Wu, K. Guo, and J. Mo. Sloping wall structure support generation for fused deposition modeling. *The International Journal of Advanced Manufacturing Technology*, 42(11-12) :1074–1081, 2009.

[46] Y. Huang, J. Zhang, X. Hu, G. Song, Z. Liu, L. Yu, and L. Liu. Framefab : Robotic fabrication of frame shapes. *ACM Transactions on Graphics*, 35, 2016.

[47] A. Ion, J. Frohnhofen, L. Wall, R. Kovacs, M. Alistar, J. Lindsay, P. Lopes, H.-T. Chen, and P. Baudisch. Metamaterial mechanism. 2016.

[48] A. Johnson. Clipper - an open source freeware library for clipping and offsetting lines and polygons, 2010.

[49] D. R. Jones. A fully general, exact algorithm for nesting irregular shapes. *J. of Global Optimization*, 59(2-3) :367–404, July 2014.

[50] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.*, 31(4) :55 :1–55 :11, July 2012.

[51] B. Koo, W. Li, J. Yao, M. Agrawala, and N. J. Mitra. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph.*, 33(6) :217 :1–217 :9, Nov. 2014.

[52] Y. Koyama, S. Sueda, E. Steinhardt, T. Igarashi, A. Shamir, and W. Matusik. Autoconnect : Computational design of 3d-printable connectors. *ACM Trans. Graph.*, 34(6) :231 :1–231 :11, Oct. 2015.

[53] V. Kreavoy, D. Julius, and A. Sheffer. Model composition from interchangeable components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 129–138, Washington, DC, USA, 2007. IEEE Computer Society.

[54] E. Kritchman, H. Gothait, and G. Miller. System and method for printing and supporting three dimensional objects, 04 2008. US Patent 7364686.

[55] G. S. Kumar, P. Pandithevan, and A. R. Ambatti. Fractal raster tool paths for layered manufacturing of porous objects. *Virtual and Physical Prototyping*, 4(2) :91–104, 2009.

[56] T. Langlois, A. Shamir, D. Dror, W. Matusik, and D. I. Levin. Stochastic structural analysis for context-aware design and fabrication. *ACM Transactions on Graphics*, 35(6), 2016.

[57] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. Converting 3d furniture models to fabricatable parts and connectors. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 85 :1–85 :6, New York, NY, USA, 2011. ACM.

[58] S. Lefebvre. Icesl : A gpu accelerated modeler and slicer. In *European Forum on Additive Manufacturing*, 2013.

[59] H. Li, R. Hu, I. Alhashim, and H. Zhang. Foldabilizing furniture. *ACM Transactions on Graphics, (Proc. of SIGGRAPH 2015)*, 34(4), 2015.

[60] Z. Li. *Compaction Algorithms for Non-convex Polygons and Their Applications*. PhD thesis, Cambridge, MA, USA, 1995. UMI Order No. GAX95-00088.

[61] L. Lu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, and B. Chen. Build-to-last : Strength to weight 3d printed objects. *ACM Trans. Graph.*, 33(4) :97 :1–97 :10, July 2014.

[62] L. Luo, I. Baran, S. Rusinkiewicz, and W. Matusik. Chopper : Partitioning models into 3D-printable parts. *ACM Transactions on Graphics*, 31(6) :129 :1–129 :9, 2012.

[63] C. Ma, N. Vining, S. Lefebvre, and A. Sheffer. Game Level Layout from Design Specification. *Computer Graphics Forum*, 2014.

[64] J. Martínez, J. Dumas, and S. Lefebvre. Procedural voronoi foams for additive manufacturing. *ACM Trans. Graph.*, 35(4) :44 :1–44 :12, July 2016.

[65] J. Martínez, J. Dumas, S. Lefebvre, and L.-Y. Wei. Structure and appearance optimization for controllable shape design. *ACM Trans. Graph.*, 34(6) :229 :1–229 :11, Oct. 2015.

[66] J. McCrae, N. Umetani, and K. Singh. Flatfitfab : Interactive modeling with planar sections. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 13–22, New York, NY, USA, 2014. ACM.

[67] A. Medeiros e Sá, V. M. Mello, K. Rodriguez Echavarria, and D. Covill. Adaptive voids. *The Visual Computer*, 31(6) :799–808, 2015.

[68] V. Megaro, B. Thomaszewski, D. Gauge, E. Grinspun, S. Coros, and M. Gross. Chacra : An interactive design system for rapid character crafting. In *Symposium on Computer Animation (SCA)*, pages 123–130, 2014.

[69] À. Méndez-Feliu and M. Sbert. From obscurances to ambient occlusion : A survey. *The Visual Computer*, 25(2) :181–196, 2009.

[70] P. Merrell and D. Manocha. Continuous model synthesis. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 158 :1–158 :7, New York, NY, USA, 2008. ACM.

[71] N. J. Mitra, M. Pauly, M. Wand, and D. Ceylan. Symmetry in 3D Geometry : Extraction and Applications. *Computer Graphics Forum*, 2013.

[72] G. E. Moore. Readings in computer architecture. chapter Cramming More Components Onto Integrated Circuits, pages 56–59. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[73] C. Mota. The rise of personal fabrication. In *Proceedings of the 8th ACM Conference on Creativity and Cognition*, C&#38 ;C '11, pages 279–288, New York, NY, USA, 2011. ACM.

[74] S. Mueller, S. Im, S. Gurevich, A. Teibrich, L. Pfisterer, F. Guimbretière, and P. Baudisch. Wireprint : 3d printed previews for fast prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 273–280, New York, NY, USA, 2014. ACM.

[75] S. Mueller, B. Kruck, and P. Baudisch. Laserorigami : Laser-cutting 3d objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2585–2592, New York, NY, USA, 2013. ACM.

[76] S. Mueller, P. Lopes, and P. 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.

[77] S. Mueller, T. Mohr, K. Guenther, J. Frohnhofen, and P. Baudisch. fabrickation : Fast 3d printing of functional objects by integrating construction kit building blocks.

In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3827–3834, New York, NY, USA, 2014. ACM.

[78] G. Nishida, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and A. Bousseau. Interactive sketching of urban procedural models. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 2016.

[79] L. Olsen, F. F. Samavati, M. C. Sousa, and Joaquim. Sketch-based modeling : A survey. *Computers And Graphics*, 33(1) :85 – 103, 2009.

[80] P. M. Pandey, N. V. Reddy, and S. G. Dhande. Slicing procedures in layered manufacturing : a review. 9(5), 2003.

[81] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[82] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3) :253–262, July 1989.

[83] R. Prévost, E. Whiting, S. Lefebvre, and O. Sorkine-Hornung. Make It Stand : Balancing shapes for 3D fabrication. *ACM Transactions on Graphics*, 32(4) :81 :1–81 :10, 2013.

[84] P. Prusinkiewicz. Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86*, pages 247–253, Toronto, Ont., Canada, Canada, 1986. Canadian Information Processing Society.

[85] P. Prusinkiewicz, M. James, and R. Mech. Synthetic topiary. In *SIGGRAPH '94 : Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1994. ACM Press.

[86] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[87] T. Reiner, N. A. Carr, R. Mech, O. Stava, C. Dachsbacher, and G. S. P. Miller. Dual-color mixing for fused deposition modeling printers. *Computer Graphics Forum*, 33(2) :479–486, 2014.

[88] T. Reiner and S. Lefebvre. Interactive Modeling of Support-free Shapes for Fabrication. In T. Bashford-Rogers and L. P. Santos, editors, *EG 2016 - Short Papers*. The Eurographics Association, 2016.

[89] J. R. Rossignac and A. A. G. Requicha. Offsetting operations in solid modelling. *Comput. Aided Geom. Des.*, 3(2) :129–148, Aug. 1986.

[90] D. Saakes, T. Cambazard, J. Mitani, and T. 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.

[91] E. Sabourin, S. A. Houser, and J. H. Bøhn. Accurate exterior, fast interior layered manufacturing. *Rapid Prototyping Journal*, 3(2) :44–52, 1997.

[92] G. Saul, M. Lau, J. Mitani, and T. Igarashi. Sketchchair : An all-in-one chair design system for end users. In *Proceedings of the Fifth International Conference*

on Tangible, Embedded, and Embodied Interaction, TEI '11, pages 73–80, New York, NY, USA, 2011. ACM.

[93] R. Schmidt and N. Umetani. Branching support structures for 3d printing. In *ACM SIGGRAPH 2014 Studio*, SIGGRAPH '14, pages 9 :1–9 :1, New York, NY, USA, 2014. ACM.

[94] A. Schulz, A. Shamir, D. I. W. Levin, P. Sitthi-amorn, and W. Matusik. Design and fabrication by example. *ACM Trans. Graph.*, 33(4) :62 :1–62 :11, July 2014.

[95] Y. Schwartzburg and M. Pauly. Design and optimization of orthogonally intersecting planar surfaces. pages 191–199, 2012.

[96] Y. Schwartzburg and M. Pauly. Fabrication-aware design with intersecting planar pieces. *Computer Graphics Forum*, 32(2pt3) :317–326, 2013.

[97] M. Shugrina, A. Shamir, and W. Matusik. Fab forms : Customizable objects for fabrication with validity and geometry caching. *ACM Trans. Graph.*, 34(4) :100 :1–100 :12, July 2015.

[98] O. Sigmund. A 99 line topology optimization code written in matlab. *Struct. Multidiscip. Optim.*, 21(2) :120–127, Apr. 2001.

[99] O. Sigmund and K. Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6) :1031–1055, 2013.

[100] P. Sitthi-Amorn, J. E. Ramos, Y. Wangy, J. Kwan, J. Lan, W. Wang, and W. Matusik. Multifab : A machine vision assisted platform for multi-material 3d printing. *ACM Trans. Graph.*, 34(4) :129 :1–129 :11, July 2015.

[101] M. Skouras, B. Thomaszewski, S. Coros, B. Bickel, and M. Gross. Computational design of actuated deformable characters. *ACM Transactions on Graphics*, 32(4) :82 :1–82 :10, 2013.

[102] P. Song, B. Deng, Z. Wang, Z. Dong, W. Li, C.-W. Fu, and L. Liu. Cofifab : Coarse-to-fine fabrication of large 3d objects. *ACM Trans. Graph.*, 35(4) :45 :1–45 :11, July 2016.

[103] P. Song, Z. Fu, L. Liu, and C.-W. Fu. Printing 3D objects with interlocking parts. *Computer Aided Geometric Design*, 35–36 :137 – 148, 2015.

[104] O. Stava, J. Vanek, B. Benes, N. Carr, and R. Měch. Stress relief : Improving structural strength of 3d printable objects. *ACM Trans. Graph.*, 31(4) :48 :1–48 :11, July 2012.

[105] G. Stiny, J. Gips, G. Stiny, and J. Gips. Shape grammars and the generative specification of painting and sculpture. In *Segmentation of Buildings for 3DGeneralisation. In : Proceedings of the Workshop on generalisation and multiple representation , Leicester*, 1971.

[106] G. Strano, L. Hao, R. Everson, and K. Evans. A new approach to the design and optimisation of support structures in additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 66(9-12) :1247–1254, 2013.

[107] Y. S. Suh, M. J. Wozny, et al. Adaptive slicing of solid freeform fabrication processes. In *Solid Freeform Fabrication Symposium*, pages 404–411, 1994.

[108] L. H. Sullivan. The tall office building artistically considered. *Lippincott's Monthly Magazine*, 1896.

[109] A. M. Sykora. *Nesting problems Exact and heuristic algorithms.* PhD thesis, 2013.

[110] K. Takayama, R. Schmidt, K. Singh, T. Igarashi, T. Boubekeur, and O. Sorkine. GeoBrush : Interactive Mesh Geometry Cloning. *Computer Graphics Forum*, 2011.

[111] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2) :11 :1–11 :14, Apr. 2011.

[112] M. Taufik and P. K. Jain. Role of build orientation in layered manufacturing : a review. *International Journal of Manufacturing Technology and Management*, 27(1-3) :47–73, 2013.

[113] M. Taufik and P. K. Jain. Volumetric error control in layered manufacturing. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2014.

[114] B. Thomaszewski, S. Coros, D. Gauge, V. Megaro, E. Grinspun, and M. Gross. Computational design of linkage-based characters. *ACM Trans. Graph.*, 33(4) :64 :1–64 :9, July 2014.

[115] K. Thrimurthulu, P. M. Pandey, and N. V. Reddy. Optimum part deposition orientation in fused deposition modeling. *International Journal of Machine Tools and Manufacture*, 44(6) :585 – 594, 2004.

[116] J. Tyberg and J. H. Bøhn. Local adaptive slicing. *Rapid Prototyping Journal*, 4(3) :118–127, 1998.

[117] N. Umetani, T. Igarashi, and N. J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4) :86 :1–86 :11, July 2012.

[118] N. Umetani, D. M. Kaufman, T. Igarashi, and E. Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4) :90 :1–90 :12, July 2011.

[119] F. G. Ureta, C. Tymms, and D. Zorin. Interactive Modeling of Mechanical Objects. *Computer Graphics Forum*, 2016.

[120] T. Van Hook. Real-time shaded nc milling display. In *Proceedings of SIGGRAPH*, 1986.

[121] J. Vanek, J. A. G. Galicia, and B. Benes. Clever Support : Efficient Support Structure Generation for Digital Fabrication. *Computer Graphics Forum*, 2014.

[122] J. Vanek, J. A. G. Galicia, B. Benes, R. Měch, N. Carr, O. Stava, and G. S. Miller. Packmerger : A 3d print volume optimizer. *Computer Graphics Forum*, 2014.

[123] O. Šťava, B. Beneš, R. Měch, D. G. Aliaga, and P. Krištof. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*, 29(2) :665–674, 2010.

[124] P. K. Wah, K. G. Murty, A. Joneja, and L. C. Chiu. Tool path optimization in layered manufacturing. *Iie Transactions*, 34(4), 2002.

[125] L. Wang and E. Whiting. Buoyancy optimization for computational fabrication. *Computer Graphics Forum*, 35(2) :49–58, 2016.

[126] W. Wang, H. Chao, J. Tong, Z. Yang, X. Tong, H. Li, X. Liu, and L. Liu. Saliency-preserving slicing optimization for effective 3D printing. *Computer Graphics Forum*, 34(6) :148–160, 2015.

[127] W. Wang, T. Y. Wang, Z. Yang, L. Liu, X. Tong, W. Tong, J. Deng, F. Chen, and X. Liu. Cost-effective printing of 3d objects with skin-frame structures. *ACM Trans. Graph.*, 32(6) :177 :1–177 :10, Nov. 2013.

[128] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009.

[129] C. White. King kong : The building of 1933 new york city. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[130] M. T. Wong, D. E. Zongker, and D. H. Salesin. Computer-generated floral ornament. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 423–434, New York, NY, USA, 1998. ACM.

[131] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3) :669–677, July 2003.

[132] J. Wu, C. Dick, and R. Westermann. A system for high-resolution topology optimization. *IEEE Transactions on Visualization and Computer Graphics*, 22(3) :1195–1208, Mar. 2016.

[133] J. Wu, C. C. Wang, X. Zhang, and R. Westermann. Self-supporting rhombic infill structures for additive manufacturing. *Computer-Aided Design*, pages –, 2016.

[134] R. Wu, H. Peng, F. Guimbretière, and S. Marschner. Printing arbitrary meshes with a 5dof wireframe printer. *ACM Trans. Graph.*, 35(4) :101 :1–101 :9, July 2016.

[135] H. Xiaomao, Y. Chunsheng, and H. Yongjun. Tool path planning based on end-point build-in optimization in rapid prototyping. *Proceedings of the Institution of Mechanical Engineers (Part C)*, 2011.

[136] B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. McCrae, and K. Singh. True2form : 3d curve networks from 2d sketches via selective regularization. *Transactions on Graphics (Proc. SIGGRAPH 2014)*, 33(4), 2014.

[137] U. Yaman, N. Butt, E. Sacks, and C. Hoffmann. Slice coherence in a query-based architecture for 3d heterogeneous printing. *Computer-Aided Design*, 75–76 :27 – 38, 2016.

[138] Y.-L. Yang, J. Wang, and N. J. Mitra. Reforming Shapes for Material-aware Fabrication. *Computer Graphics Forum*, 2015.

[139] M. Yao, Z. Chen, L. Luo, R. Wang, and H. Wang. Level-set-based partitioning and packing optimization of a printable model. *ACM Transactions on Graphics*, 34(6) :214 :1–214 :11, 2015.

[140] X. Zhang, X. Le, A. Panotopoulou, E. Whiting, and C. C. L. Wang. Perceptual models of preference in 3d printing direction. *ACM Trans. Graph.*, 34(6) :215 :1–215 :12, Oct. 2015.

[141] X. Zhang, Y. Xia, J. Wang, Z. Yang, C. Tu, and W. Wang. Medial axis tree—an internal supporting structure for 3d printing. *Computer Aided Geometric Design*, 35–36 :149 – 162, 2015. Geometric Modeling and Processing 2015.

[142] Y. Zheng, D. Cohen-Or, and N. J. Mitra. Smart variations : Functional substructures for part compatibility. *Computer Graphics Forum*, 32(2pt2) :195–204, 2013.

[143] Q. Zhou, J. Panetta, and D. Zorin. Worst-case structural analysis. *ACM Transactions on Graphics*, 32(4), 2013.

[144] Y. Zhou, S. Sueda, W. Matusik, and A. Shamir. Boxelization : Folding 3d objects into boxes. *ACM Trans. Graph.*, 33(4) :71 :1–71 :8, July 2014.

[145] L. Zhu, W. Xu, J. Snyder, Y. Liu, G. Wang, and B. Guo. Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6) :127 :1–127 :10, Nov. 2012.

Synthèse de formes fabricables à partir de spécifications partielles

Résumé long en français

Le développement des Fablabs et des Makerspaces montre que les techniques de fabrication rapide deviennent de plus en plus populaires. Une grande majorité de ces lieux est équipée d'imprimante 3D à fil fondu et de découpe laser, qui permettent de fabriquer des objets à faible coût. Leur popularisation à conduit au développement d'imprimantes de moins en moins chères et de plus en plus facile à utiliser.

Ces technologies de fabrication rapide, issues des techniques de prototypage rapide comme l'impression 3D permettent de fabriquer des pièces uniques sans demander d' expertise particulière du procédé mis en œuvre. En revanche, la modélisation de nouveaux objets tout comme la personnalisation d'objets existants restent difficiles : en effet, les techniques de prototypage rapide imposent des contraintes sur la géométrie du modèle qui doivent être respectées (e.g. epaisseurs minimale, angles limites). De plus, l'explosion de ces technologies a modifié le profil de l'utilisateur qui manipulait les machines. Au départ, il s'agissait d'experts designers qui étaient formés pour manipuler ces technologies, aujourd'hui les machines sont utilisées par des amateurs enthousiastes, des enseignants, qui souvent ne sont pas formés pour utiliser ces machines. Cette évolution a créé un besoin de nouveaux outils logiciels utilisables par un plus grand nombre de personnes. Cette thèse présente un ensemble de techniques qui ont pour point commun d'assister l'utilisateur dans la modélisation d'un objet, en tenant compte des contraintes du procédé qui permettra de le fabriquer. À cette fin, l'algorithme prend en charge tout ou partie de la modélisation. Les problèmes présenté dans cette thèse sont d'ordre combinatoires et ont requis le développement d'approches heuristiques dédiées.

Dans le premier chapitre, je propose d'améliorer la qualité des objets fabriqués avec une imprimante 3D en minimisant certains défauts qui apparaissent lors de la fabrication. Je m'intéresse en particulier aux techniques de fabrication par fil fondu (FFF) qui sont les plus répandues. Le filament est poussé par un moteur pas à pas dans une tête d'impression qui est chauffée. La tête dépose la plastique couche après couche pour fabriquer un objet complet. Certaines imprimantes possèdent plusieurs têtes d'impression, ce qui permet de déposer des matériaux de natures différentes. En revanche, la qualité de l'impression souffre de la présence de plusieurs matériaux : du plastique fondu tombe par gravité des têtes d'impression qui ne sont pas en train d'imprimer, et vient se coller sur l'impression, laissant apparaître des défauts visibles sur l'objet. Ce problème est appelé oozing et le résoudre n'est pas simple : augmenter la vitesse certes réduit le problème, mais réduit aussi la qualité de surface. De plus, modifier l'imprimante pour ajouter un dispositif physique limitant les dépots parasites a un coût, et augmente le temps d'impression.

À la place, nous avons décidé de modifier les chemins d'impression en exploitant les degrés de liberté du procédé. Nous introduisons trois techniques pour améliorer significativement la qualité d'impression. Premièrement, en changeant l'orientation de la pièce

sur le plateau d'impression pour minimiser l'impact du oozing. Ensuite, nous fabriquons un rempart autour de la pièce pour nettoyer les têtes d'impression. Dernièrement, nous introduisons une technique de planification de chemin pour éviter ou cacher au mieux les défauts qui apparaissent à cause du oozing. L'algorithme de planification est basé sur la recherche de zones peu visible du modèle grâce au calcul de l'occlusion ambiante. La partie finale de ce chapitre est dédié aux comparaisons de différents logiciels d'impression.

Dans le second chapitre, je propose d'aider l'utilisateur à prendre en compte les contraintes de fabrication pendant la modélisation. Mes techniques utilisent des informations partielles sur la forme que l'utilisateur souhaite fabriquer. J'ai appliqué cette méthodologie à deux problèmes distincts.

Le premier est la conception de mécanismes. En effet, comme le montre le succès des applications de simulation physique "bac à sable" (par exemple Algodoo, Phyzical Sandboxes, ou même MineCraft) qui permettent aux utilisateurs novices de modéliser des mécanismes simples, il existe un fort intérêt pour ce type d'objets. De plus, les mécanismes sont complexes à modéliser même pour des experts de part les nombreuses parties mobiles impliquées. Dans les applications facile d'utilisation, un ensemble de mécanismes sont modélisés en 2D en plaçant des roues dentées, des axes, et d'autres composants de bases a travers une interface intuitive. L'expérience de l'utilisateur reste agréable car il n'a pas à s'inquiéter des détails géométriques complexes nécessairent à la fabrication du vrai mécanisme. Dans ce chapitre, nous proposons donc de transformer un modèle de mécanisme 2D en mécanisme 3D qui peut être directement fabriqué (sans assemblage) sur des machines à dépôt de fils fondu. L'idée est de donner à l'utilisateur de ces logiciels la possibilité de fabriquer leurs modèles et de les voir fonctionner dans le monde réel.

Pour cela, nous résolvons plusieurs problèmes. Le mécanisme 2D d'entrée permet à certaines pièces de se chevaucher pendant la simulation. Ces pièces 2D qui se chevauchent doivent être transformées en pièces 3D qui ne se collisionnent pas dans le mécanisme fabriqué. La forme exacte du mécanisme 3D est déduite de la forme des pièces en 2D ainsi que de la simulation du mécanisme en 2D. Nous utilisons aussi de l'optimisation topologique pour synthétiser un châssis qui maintient le mécanisme fabriqué en une seule pièce, en prenant en compte les forces exercées sur les axes pendant la simulation.

J'ai aussi appliqué cette méthodologie au design de meubles avec des modèles paramétrique. Ces modèles paramétriques sont faciles à trouver sur des sites spécialisés comme Thingiverse. Ils définissent des fonctions qui génèrent des formes à partir d'un ensemble de paramètres vers un espace de design. Ces paramètres sont choisis par l'utilisateur afin d'avoir une forme plaisante, qui remplie une fonction. Dans certains cas, les valeurs exactes des paramètres ne sont pas importantes pour l'utilisateur, mais le sont pour le procédé de fabrication (ici, une découpe laser est utilisé pour découper les différentes parties du meuble qui sont assemblées plus tard). En revanche, elles impactent le procédé de fabrication. Dans de telles situations, les paramètres peuvent être optimisés automatiquement. Dans ces travaux, nous introduisons le problème de « waste-minimizing-furniture-design » (modélisation minimisant le gâchis) dans le cas de meubles composé de pièce plates découpés au laser. En particulier, nous étudions le lien entre la modélisation et le gâchis de

bois. En mixant les deux, nous donnons a l'utilisateur la capacité de prendre des décisions basé de modélisation basées sur le gâchis du bois.

Le lien établi entre le gâchis de bois et le modèle paramétrique est basé sur un algorithme qui explore l'espace de design en modifiant les paramètres du modèle de façon a optimiser l'espace matériel : une planche dans laquelle des différentes pièces du meuble sont placées pour gâcher le moins de bois possible. L'impact des paramètres sur la taille et la forme des pièces qui composent le meuble est pris en compte pendant l'optimisation pour modifier seulement les paramètres qui ont une influence sur l'espace matériel.

Enfin, dans certains cas (e.g. Grand public) l'utilisateur n'est pas forcément à même de modéliser ces formes via des logiciels spécialisés. Pour ce cas précis, je propose une technique de synthèse de meubles à partir de spécifications fonctionnelles, e.g. la spécification de poids à porter dans l'espace.

Le dernier chapitre de cette thèse porte sur une technique de synthèse de meuble inspiré des techniques d'optimisation topologique. Contrairement au technique présentée dans le second chapitre, l'utilisateur ne fournit pas d'information sur la forme de l'objet qu'il souhaite fabriquer, mais seulement sur sa fonction. Après avoir spécifié un ensemble de poids a porter dans l'espace, il laisse le système synthétiser le meuble qui permet de supporter tous ces objets. L'algorithme est inspiré des techniques de génération de support pour les techniques d'impression 3D. La synthèse génère aussi un ensemble de contraintes sur la forme du meuble, qui permet a l'utilisateur de l'éditer, pour minimiser l'utilisation de matière, sans s'inquiéter de la topologie de la forme finale. Cette technique simplifie la modélisation a l'extrême et ne prend pas en compte l'esthétique de l'objet synthétisé.

Français Les techniques de fabrication rapide, issues des techniques de prototypage rapide comme l'impression 3D ou la découpe laser permettent de fabriquer des pièces uniques sans demander d'expertise particulière du procédé mis en œuvre. En revanche la modélisation de nouveaux objets tout comme la personnalisation d'objets existants restent difficiles : En effet, les techniques de prototypages rapides imposent des contraintes sur la géométrie du modèle qui doivent être respectées.

Cette thèse présente un ensemble de techniques qui ont pour point commun d'assister l'utilisateur dans la modélisation d'un objet, en tenant compte des contraintes du procédé qui permettra de le fabriquer. A cette fin, l'algorithme prend en charge tout ou partie de la modélisation.

En particulier, les problématiques suivantes sont abordées :
Tout d'abord, je propose d'améliorer la qualité des objets fabriqués avec une imprimante 3D en minimisant certains défauts qui apparaissent lors de la fabrication. Les approches développées modifient uniquement les algorithmes de pilotage de l'imprimante.

En second lieu, je propose d'aider l'utilisateur à prendre en compte les contraintes de fabrication pendant la modélisation. Mes techniques utilisent des informations partielles sur la forme que l'utilisateur souhaite fabriquer, comme le dessin en deux dimensions d'un mécanisme, ou un modèle paramétrique qui définit un meuble. L'algorithme optimise une forme finale qui améliore des critères liés à sa fabrication (gaspillage, encombrement, etc.).

Enfin, dans certains cas (e.g. grand public) l'utilisateur n'est pas forcément à même de modéliser ces formes via des logiciels spécialisés. Pour ce cas précis, je propose une technique de synthèse de meubles à partir de spécifications fonctionnelles, e.g. la spécification de poids à porter dans l'espace.

English The Rapid Manufacturing techniques that emerged from Rapid Prototyping techniques such as 3D printing or laser cutting allow to fabricate unique objects. However, the design of those objects with existing CAD software remain a difficult task : rapid prototyping processes impose constraints on the geometry of the model.

This thesis presents a set of techniques that assist the user in the design of an object by taking into account the constraints of the fabrication process. To achieve this, the algorithm automatically performs part of the modelling process.

The following problems have been tackled :
First, I propose to improve the quality of 3D printed objects by minimizing defects that appear during the fabrication. The technique developed impacts only the algorithm that drives the printer.

Then, I propose to help the user to take into account the fabrication constraints during the modelling process. My techniques rely on partial information about the shape that the user wants to fabricate like the 2D sketch of a mechanism or a parametric model of a furniture. The algorithm optimizes the initial shape to improve fabrication objectives (Wastage, etc.)

Finally, in some cases, the user does not know how to operate dedicated software. In this case, I propose a synthesis technique of furniture from functionnal specification, e.g. loads that have to be supported in space.