Layout problems under topological constraints for computational fabrication

Marco Freire

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Layout problems under topological constraints for computational fabrication

THÈSE

présentée et soutenue publiquement le 11 juillet 2024

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Marco Freire

Composition du jury

Président :	Tamy Boubekeur	
Rapporteurs :	Tamy Boubekeur	Adobe Research
	Nobuyuki Umetani	University of Tokyo
Examinatrice :	Mélina Skouras	Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK
Invitée :	Marie-Odile Berger	Université de Lorraine, CNRS, Inria, LORIA
Directeur :	Sylvain Lefebvre	Université de Lorraine, CNRS, Inria, LORIA

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

Estudia mucho, que eso nadie te lo puede quitar.

Abuela Pilar

Acknowledgements

There are many people without whom this thesis wouldn't have been possible. Some of them speak French, some Spanish, and some of them understand English (thus luckily able to read this dissertation). This introduction is written in these languages, and specific acknowledgements are written in the language spoken by their addressee.

Cette thèse aurait été impossible sans le soutien de beaucoup de personnes. Certaines parlent français, d'autres espagnol, et quelques unes comprennent l'anglais (ayant donc le privilège de pouvoir lire ce manuscrit). Ainsi, cette introduction est écrite en trois langues, et les remerciements spécifiques sont écrits en la langue parlée par leurs destinataires.

Esta tesis habría sido imposible sin el apoyo de muchas personas. Algunas hablan francés, otras español, y ciertas entienden el inglés (teniendo así la suerte de poder leer este manuscrito). Por está razón escribo esta introducción en esos tres idiomas, y los agradecimientos específicos en el idioma que hablan sus destinatarios.

Je tiens tout d'abord à remercier Sylvain Lefebvre, mon directeur de thèse. Une thèse est le produit d'une collaboration entre doctorant et directeur de thèse ; et si être doctorant est difficile, être un bon directeur de thèse l'est tout autant. En recherche, tu m'as toujours traité en égal malgré mon peu d'expérience et m'as poussé à donner le meilleur de moi-même. Humainement, tu as su me soutenir et m'encourager pendant les moments difficiles de cette thèse. Je pense que ça aurait été difficile de mieux tomber.

J'étends ces remerciements à toute l'équipe MFX : a los hispano-(y catalano-) hablantes obviamente: Salim, Jonàs y Luis ; à Cédric et Xavier pour les pauses café et les échanges toujours calmes et sereins au sujet de nos enseignements respectifs ; à Camille qui a intégré l'équipe en même temps que moi ; et à tous les doctorants qui sont passés par là. Une petite mention spéciale pour mon co-bureau Pierre-Alexandre, toujours prêt à me montrer ses derniers projets d'impression 3D et à me distraire quand les journées de travail se font longues. Finalement, merci à tous les profs qui m'ont encouragé à procrastiner mon entrée dans la vie laborale en faisant des études interminables.

Thank you Manas for our collaboration on PCBend. It was a very ambitious research project, it was pretty hard at times and we found many obstacles on the way, but working with you was always a pleasure. Your kindness and motivation are inspiring, and I wish you the best.

Laura et Max, je vous ai rencontrés relativement tard dans ma thèse, mais vous y avez participé bien plus que vous ne pensez. J'avais mon directeur de thèse au laboratoire, mais vous avez été mes "*directeurs de sport*" pendant les

deux dernières années de ma thèse. Pendant ce temps j'ai appris et de vous, et sur moi, non seulement des nouvelles choses, mais aussi de ce que j'ai toujours été capable. Je vous remercie pour ce voyage sportif qu'on a commencé ensemble et qui continue encore.

Merci au gens du club Tarot de l'ENS Rennes, sans qui je n'aurais clairement pas eu assez de memes pour garder le moral le long de ces quatre années. Merci à ceux qui sont venus à ma soutenance. Merci pour les retrouvailles pseudo-annuelles à Cherbourg. Merci à Lucien, ma référence en droit européen qui s'est avérée être plutôt une référence en relations humaines. Merci à Clément, ma référence incontournable en coloriage, Emacs, cactus, rythme de sommeil, cuisine et sport. Courage à tous ceux qui n'ont pas encore soutenu ; vous verrez, c'est mieux après.

Merci aux gens de l'IJL, qui, m'ayant à peine rencontré, m'ont presque fait croire que j'y connaissais quelque chose en science des matériaux. En particulier merci à toi Lucie, tu es la deuxième grande surprise de ma thèse. Il nous aura fallu beaucoup de temps pour se rencontrer, mais très peu pour se connaître. Ton empathie, ta sincérité et ta transparence sont toujours rafraîchissantes, et malheureusement rares en telles quantités. Je t'en suis reconnaissant d'avoir choisi de partager tes qualités avec moi.

Clélia, tu fais techniquement partie de l'IJL, mais je te rends les égards que tu m'accordes dans ton manuscrit en te consacrant ton propre paragraphe, un rappel pour chaque fois que tu oublieras tout ce que tu fais et as fait pour moi. Tu es la première grande surprise de ma thèse. L'hasard a voulu qu'on fasse de la recherche dans la même ville, et à force de rechercher on s'est retrouvés. On a repris notre amitié là où on l'a laissée trois ans auparavant, comme si le temps ne s'était jamais écoulé. Tu m'as traîné faire du sport, on est devenus co-athlètes, et maintenant t'en subis les conséquences. Tu m'as écouté, tu m'as réconforté, et tu m'as soutenu. Tu as eu l'amabilité de me distraire avec tes problèmes quand je fuyais les miens. La thèse nous aura permis de nous retrouver, mais maintenant c'est à nous de choisir notre chemin.

A mi familia, ya sabéis que se me da mejor la palabra escrita que oral, y conocéis más al Marco callado que al Marco hablador, así que aprovecho para agradecer aquí todo lo que no os he agradecido hasta ahora.

Gracias por vuestro apoyo incondicional, por animarme a encontrar mi pasión, por inspirarme. Si hoy tengo un doctorado, es porque desde pequeño habéis alimentado mi curiosidad, habéis respondido a mis preguntas, y me habéis enseñado tantas cosas. A la parte docente de la familia le debo la pasión por la enseñanza, a la parte ingeniera la pasión por la ciencia. Habéis contribuido todos, lo creáis o no, a este manuscrito. Tanto habéis confiado en mí, y tan seguros habéis estado todos de que iba a conseguir lo que emprendiese, que a veces he tenido miedo de no ser capaz de responder a esas expectativas. De tanto pensar que el fracaso no es una opción, me sigue costando a veces darme cuenta de mis logros. Pero sé que esas expectativas son mías y no vuestras, que estaréis ahí pase lo que pase. Gracias.

A los abuelitos, gracias por haber venido a mi defensa. Me había preparado a que no pudiéseis venir, pero estoy inmensamente agradecido de que estuviéseis aquí. No habría sido lo mismo sin vosotros. Sois abuelitos porque tenéis más edad que mamá y papá, pero en realidad sois mis segundos padres.

A mamá, sé lo difícil que fue verme irme a hacer mis estudios en Francia. Gracias por haberme apoyado sabiendo lo duro que iba a ser. Irme de casa, ver "mundo", pasar buenos momentos y atravesar adversidades lejos de tí me han permitido darme cuenta de lo importante que siempre has sido para mí.

A papá, que te llamo Bernar únicamente por costumbre, es tu culpa de que en la portada ponga "*mention informatique*". Sin todas esas horas que pasé sentado en una silla al lado tuyo mirándote (sobre todo) jugar al ordenador o (a veces) programar, no hubiese considerado dedicarme a la informática. De mayor quiero ser inventor como tú.

Y finalmente a Nico, que no viaja en julio. Has crecido tanto desde que empecé mis estudios hace nueve años que se me olvida que sólo tenías siete años cuando tu hermano se fue de casa. El tiempo pasa tan rápido que estás ya cerca de poder elegir lo que quieres hacer en la vida. Parece ser que vas a elegir la vía de la enseñanza y/o la investigación en matemáticas. A lo mejor te he servido de inspiración, a lo mejor son simplemente cosas que te gustan, o a lo mejor los dos; quién sabe? Pase lo que pase, haz lo que te de la gana, lo que más te guste, y disfrútalo. Sea lo que sea, estaré ahí para responder a cualquier pregunta que tengas, e insistiré para que me lo expliques todo. Hagas lo que hagas, lo más importante es que estaré ahí para fastidiarte.

There are many people I haven't explicitly named that have helped a ton both during and before my thesis. If you are reading these acknowledgements until the end, this is probably addressed to you. Thank you.

De nombreuses personnes que je n'ai pas nommées ont m'ont beaucoup aidé, pendant la thèse ou avant. Si vous lisez ces remerciements jusqu'au bout, c'est probablement à vous que je pense en écrivant ces lignes. Merci.

Hay mucha gente a la que no he nombrado a quién le debo mucho, ya sea durante la tesis o antes. Si leéis estos agradecimientos hasta el final, probablemente estas líneas os estén dirigidas. Muchas gracias.

Feel free to interrupt at any time to ask questions.

N'hésitez pas à interrompre pour poser des questions.

No dudéis en interrumpirme para hacer preguntas.

Résumé

Les problèmes d'agencement surviennent dans de nombreux contextes en ingénieurie et en informatique. Typiquement, la résolution d'un problème d'agencement consiste en l'organisation spatiale et l'interconnexion d'un ensemble d'éléments dans un espace. Cet espace et ces interconnexions peuvent être de complexité très variable. Un ensemble de contraintes et d'objectifs complémentent la description du problème, tels que minimiser la longueur ou la surface des interconnexions, ou fixer la position de certains éléments. La planification des étages en architecture, de niveaux de jeux vidéo, l'agencement d'installations industrielles ou de circuits électroniques, sont tous des exemples de problèmes d'agencement.

Les contraintes topologiques jouent un rôle important dans l'agencement. La topologie considère des objets définis par les voisinages de leurs éléments, sans s'attarder sur leur géométrie spécifique. Par exemple, un graphe est une entité topologique, constituée uniquement des liens entre ses nœuds. Au contraire, dessiner un graphe est une opération géométrique, puisqu'elle demande de spécifier la position des nœuds.

Cette thèse se focalise sur la résolution de deux problèmes d'agencement spécifiques liés à la fabrication et la conception computationnelles sujets à des contraintes topologiques. Plus particulièrement, il s'agit de la génération d'agencements de circuits électroniques et la génération de supports pour l'impression 3D.

La première contribution est un système pour la conception d'écrans surfaciques constitués de DEL RVB à travers l'utilisation de circuits imprimés pliables. Nous plions les circuits imprimés traditionnels en utilisant des motifs de découpe localisés, créant ainsi des 'charnières' dans la plaque. Le système prend en entrée un maillage basse-résolution et produit des plans pouvant être envoyés à des services en ligne de fabrication de circuits. Suite à la fabrication, l'écran est assemblé en pliant le circuit sur une impression 3D du maillage d'origine. Les écrans fabriqués peuvent être contrôlés à travers une interface similaire à des shaders pour créer des effets lumineux impressionnants. Le problème global est découpé en sous-problèmes locaux grâce à la topologie chaînée du circuit, les plans finaux étant obtenus en 'recousant' les solutions aux sous-problèmes. Au lieu de suivre la méthode traditionelle d'agencement électronique (concevoir le schéma électrique, placer et connecter les composants) ; nous décidons du nombre de composants, leur placement et leur routage séparément pour chaque triangle au moment-même de la génération.

La deuxième contribution est un algorithme procédural pour la génération de supports pour l'impression 3D sous forme d'échafaudages. Ces supports s'impriment de manière fiable et sont stables [DHL14]. L'algorithme précédent ne considère pas les intersections entre les supports et l'objet imprimé, laissant des marques indésirables sur la surface de l'objet. De plus, la complexité de l'algorithme dépend du nombre de points à porter. Nous proposons un nouvel algorithme inspiré du *Model Synthesis* (MS) [Mer09]. Il évite implicitement les intersections et sa complexité est indépendante du nombre de points à porter. Les supports sont représentés indirectement à travers un ensemble d'étiquettes, chacune représentant une partie de la structure (par exemple une partie de pilier, de pont, ou une jonction) ; et un ensemble de contraintes d'adjacence déterminant quelles combinaisons d'étiquettes sont possibles dans toutes les directions. Les supports sont générés de haut en bas en attribuant de façon répétée une étiquette à un voxel, puis en propageant les contraintes afin d'éliminer les étiquettes rendues impossibles. Cet algorithme, les contraintes d'adjacences et les heuristiques utilisées sont conçues ensemble pour générer des supports sans essai-erreur ou retours arrière, typiques du MS et autres méthodes similaires.

Abstract

Layout problems appear in many areas of engineering and computer science. Typically, a layout problem requires to spatially arrange and interconnect a number of geometric elements in a domain. The elements can have a fixed or variable size, as well as an arbitrary shape. The domain may be a volume, a planar region or a surface. It may be fixed or allowed to reshape. The interconnections may be simple paths, shared contact regions, or both. A set of constraints and objectives complement the problem definition, such as minimizing interconnection length, fixed positions for some elements, and many others. Layout problems are ubiquitous: floorplanning in architectural design, video game level design, industrial facility layout planning, electronics physical layout design, and so on.

Topological constraints often arise in layout problems. Topology considers objects as defined by their elements' neighborhoods, without consideration for their specific geometry of placement. For example, a graph is a purely topological structure, consisting only of the relationships between its nodes. On the other hand, a graph drawing needs to specify the position of its nodes, i.e. the geometry of the graph.

This thesis focuses on tackling two specific layout problems subject to topological constraints arising in computational design and fabrication. These are electronic circuit physical layout generation and 3D printing support generation.

The first contribution is an entire system for the design of freeform RGB LED displays through bendable circuit boards. Typical rigid PCBs are made to bend by strategically using kerfing, i.e. cutting patterns into the board to create 'hinges' where it needs to fold. The system takes a low-poly mesh as an input and outputs fabrication-ready blueprints, that can be sent to any online PCB manufacturer. After fabrication, the display is obtained by folding the circuit over the 3D printed mesh. The LEDs are commonly found on commercially available LED strips and are easy to control. Thus, the display can be used through a programmable interface to generate impressive lighting effects in real time. The global layout problem is decomposed into local per-triangle sub-problems by exploiting the chain topology of the electronic circuit, the final layout being obtained by stitching the local solutions. Instead of traditionally following the physical design pipeline, i.e. schematics design, component placement and routing; we decide the number of components, their placement and their routing per-triangle on the fly.

The second contribution is a procedural algorithm for generating bridges-and-pillars supports for 3D printing. These supports have been shown to print reliably and in a stable manner in [DHL14]. Unfortunately, the previous algorithm struggles to generate supports that do not intersect the object, leaving visible scars on its surface after support removal. Additionally, its complexity scales with the number of points to support. We propose an algorithm based on *Model Synthesis* (MS) [Mer09] to generate these supports, with an implicit knowledge of object avoidance and a complexity independent of the number of points to support. Our algorithm works on a voxelized representation of the object. The supports are encoded in the algorithm with a set of labels, each representing a part of the structure (e.g. a pillar block, a bridge block, a pillar-bridge junction); and a set of adjacency constraints defining all possible label combinations in every direction. The supports for an object are generated top to bottom by repeatedly assigning labels to voxels and propagating constraints to remove inconsistent labels in the domain. The algorithm, adjacency constraints and heuristics are co-designed to avoid the need for trial-and-error or backtracking, typical of MS and similar approaches.

Contents

List of Abbreviations

AM	Additive Manufacturing
CAD	Computer-Aided Design
CD	Computational Design
CF	Computational Fabrication
CSP	Constraint Satisfaction Problem
CVT	Centroidal Voronoi Tessellation
DARPA	Defense Advanced Research Projects Agency
EDA	Electronic Design Automation
FDM	Fused Deposition Modeling
FPGA	Field-Programmable Gate Array
GD	Generative Design
GPU	Graphics Processing Unit
IC	Integrated Circuit
IPM	Inverse Procedural Modeling
LED	Light-Emitting Diode
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MS	Model Synthesis
PCB	Printed Circuit Board
PCG	Procedural Content Generation
PPAC	Power-Performance-Area-Cost
RGB	Red-Green-Blue
RTL	Register Transfer Level
SMD	Surface-Mount Device
VLSI	Very Large Scale Integration
WFC	Wave Function Collapse

Chapter 1

Introduction

Stay awhile and listen...

Deckard Cain

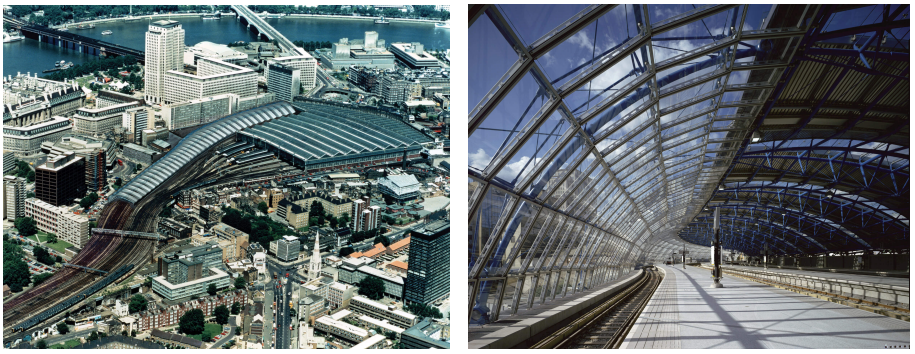
1.1 Context

This thesis takes place at the intersection of multiple domains, each with rich histories and diverse applications. This section aims to quickly present them and their objectives.

1.1.1 Computational design

The ever-increasing availability of and reliance on computers in the past decades has led to major changes in the design process and methodologies. Computers can help automate traditional design approaches, making them more time- and resource-efficient. Due to their computational power, they can also make these approaches scale to much larger instances that would have been inconceivable without computers, or would have taken considerable effort to carry out. Producing tools to help these established design methods is the goal of Computer-Aided Design (CAD).

The integration of computational tools in traditional design frameworks has also led to another, deeper paradigm shift: the lean towards computational thinking and the advent of Computational Design (CD) [dBoi22]. CD consists in a reevaluation of traditional approaches in a computation-based, algorithmic framework, defining new methods and representations for modeling and design. While originating in their presence, this type of thinking is independent from computers themselves and is related to computer (or rather computing) science instead. In architecture, this shift is reflected for example in the transition from traditional to Generative Design (GD). Computational techniques allow the design of not a single shape through a rigid representation, but of whole families of shapes through more flexible representations. Among others, these can be parametric, where a host of values with semantic meaning can be easily modified to alter the design; or fully generative, where the final design is the result of a more general algorithmic or optimization-based process.



(a) Aerial view. (b) Inside view.

Figure 1.1: International Terminal at Waterloo designed by Grimshaw Architects, completed in 1993. Photographs by Jo Reid and John Peck, obtained from the project's website. The international terminal is the long structure on the left of the aerial view.

A real-life example of this parametric design in architecture is the *International Terminal at Waterloo* [GJ12], shown in Figure 1.1. This terminal was an extension of the original station. The project was subject to especially tight

constraints, "*limited by the electrical network on one side and the roads on the other, in a winding area that narrows toward the interior*" ([GJ12]). The resulting design consists of a modular truss structure, varying in width from 50 meters at the entrance to 35 at the end. The shape was defined parametrically, easily allowing to fit a given width profile along its length. By cleverly defining the structure through a parametric and modular lens, the final building was tailor-made to fit the given constraints easier than a fully explicit approach would have allowed.

1.1.2 Computational fabrication

Many techniques previously reserved to industrial settings are becoming available to the general public. Two examples we target in this thesis are Additive Manufacturing (AM) and Printed Circuit Board (PCB) design. Established fabrication technologies such as AM or electronics prototyping have recently become accessible to everyone due to the appearance of capable and affordable 3D printers or online PCB fabrication services. These processes being widely accessible contrasts with the complexity of the processes themselves, transforming 3D models into physical objects for the former, and electronic schematics into functioning circuits for the latter. Indeed, making use of a specific fabrication technology for the first time often demands extensive knowledge of the fabrication process, significantly raising the barrier of entry, especially for newcomers.

The increasing accessibility of fabrication technologies, carried by the development of more affordable tools and more approachable software has enabled the development of hobbyist and *maker* groups. These gather online and around community-operated spaces offering access to computers, workshop tools and knowledgeable people willing to share their expertise. In turn, these communities contribute to making technologies ever more accessible, and help more people explore their potential and focus on their creative aspects. This shift has also had an impact on education, with proposals and studies on the integration of the "maker mindset" into classrooms and education curricula [Sta13; KNG15], with a special focus on how to make these spaces open and welcoming to everyone, independently of their gender, ethnicity or socioeconomic status [Mar15; TBS18].

This increasing interest in fabrication technologies and quick prototyping has been accompanied by significant advances from computer graphics and computational fabrication research in *fabrication-aware design* [BFR17].

An example of an actor leveraging these advances in fabrication technologies and computational tools is the generative design studio *Nervous system* [RL], founded in 2007 by Jessica Rosenkrantz and Jesse Louis-Rosenberg. Their work mixes generative design and computational fabrication to create intricate products and works of art. In their own words, "*instead of designing a specific form, [they] craft a system whose result is a myriad of distinct creations*" ([RL]). Figure 1.2 showcases two designs resulting from two of their projects, *Kinematics Dress* and *Floraform*. The dress is composed of 2279 unique triangular panels connected through 3316 hinges, all fabricated as a single folded piece requiring no assembly. The chandelier was generated using two of their generative algorithms, mimicking the biomechanics of growing leaves and flowers.

(a) Kinematics Dress 1, 2014. (b) Floraform chandelier, 2017.

Figure 1.2: Photographs from the Kinematics Dress and the Floraform projects from Nervous system. Images obtained from the project's respective websites (Floraform and Kinematics Dress). Photograph of the dress by Steve Marsel Studio.

1.1.3 Layout problems in design

Many algorithmic problems arising in Computational Fabrication (CF) take the form of a *layout problem*. In the context of this thesis, a layout problem refers to a collection of three elements: a space, a set of objects and a set of constraints. Solving the problem means generating a *valid* layout, i.e. arranging the objects into the space while satisfying the constraints, often while optimizing additional objectives.

These problems appear in many different shapes and forms, at many scales and in different fields. For example, constructing building floor plans is an architectural layout problem [WMR22], where inner walls and doors are placed in the floor to create rooms that should all be accessible from the main door. A particularly high-stakes example is facility layout planning [PMD21], where elements of a production system are properly arranged within the facility space to optimize layout area, construction cost, or transport time among other criteria. These are of course, simplified representations of the real-life problems, where many more constraints and objectives have to be taken into account in order to design practical spaces; but it shows that many commonplace problems can be modeled as layout problems.

Another variant is game level layout generation, a subset of Procedural Content Generation (PCG), often used nowadays in *roguelikes* such as *Spelunky* [Yu16]. These techniques are used to provide complex environments for the player to evolve in with minimal designer intervention, such as providing a connectivity graph between the levels and a set of possible room shapes [Ma+14]. While the stakes are lower, it is difficult to find the balance between automatic generation and authored content. Leaning too much towards the former makes content feel unoriginal and repetitive, while leaning towards the latter requires significant amounts of work [Gam16].

In particular, layout problems also arise in the field of CF. For example, physical design of PCB in electronics can be modeled as a layout problem. Components and conductive traces have to be placed on the physical area of

the board such that components are connected as specified in the schematic. Additional contraints come from the PCB fabrication process and from physical phenomena. For example, the spacing between traces should be over a threshold specified by the manufacturer to allow for tolerance during the copper etching process, or evenly distributing heat-generating components to preventing them from failing due to high temperatures.

1.2 Contributions

This thesis tackles two specific layout problems in CF: the embedding of electronic circuits onto a 3D shape, and the generation of support structures for AM. Albeit in different application fields, we employ a similar point of view to tackle these challenges. In both cases, we develop techniques exploiting topological constraints and information to simplify the solution space and the optimization process.

1.2.1 3D LED-based displays via foldable circuit boards

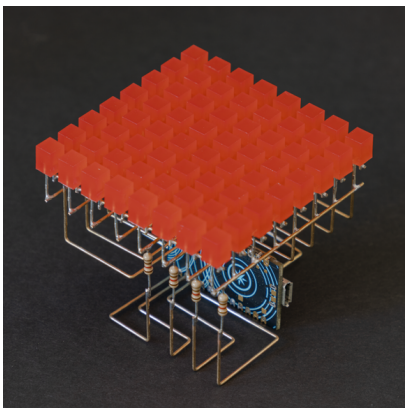
Usually, simple electronic circuits are built on top of rigid PCB, consisting of a sandwich of conductive and non-conductive layers. Components can be soldered onto these boards, and connections between these components are etched into the conductive layers [LS20]. This approach allows packing components very densely, making electronic circuits space-efficient. Additionally, affordable online PCB manufacturing services such as *PCBWay*, *JLCPCB* or *Eurocircuits* among others allow everybody to design their own custom circuits, order them and receive them fully assembled a few weeks later. Circuit design is also more accessible than ever thanks to powerful Electronic Design Automation (EDA) software suites such as the open-source suite *KiCad* [CK92], or Autodesk's *Eagle* [Cad88], recently integrated into *Fusion 360*.

Instead, in freeform electronics (see Figure 1.3), typical substrates are replaced by assemblies or skeletons of conductive wires and rods for aesthetic purposes. This makes it possible to create sculptures with a desired shape serving a specific purpose, separating form from function. Nevertheless, it comes at the expense of being harder to fabricate and often less functional. Inspired from the freedom of freeform electronics, we design foldable rigid PCB that have a much lower barrier of entry due to the availability of the technology. Flexible PCB exist and are used in particular for board interconnects in tight spaces, but are significantly more expensive [Wan+20; All23], have tighter design constraints and are harder to fabricate.

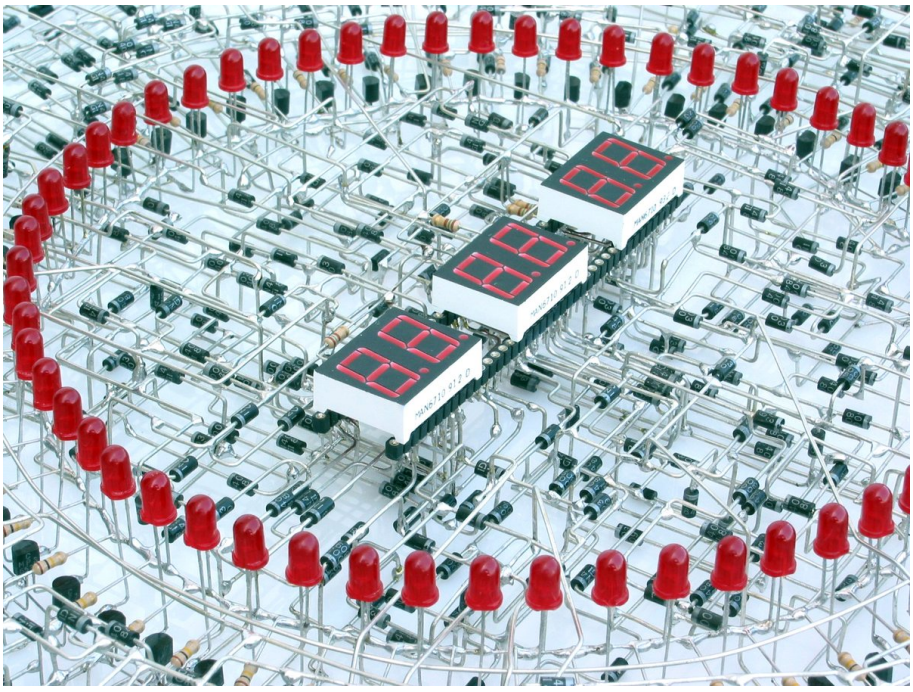
We insert kerfing patterns into our PCB, which are normally used for bending wood. These are patterns cut into the material that allow it to deform or bend in the desired way in specific regions. These allow us to take an input shape represented as a polygon mesh, unfold it along its edges, lay out an electronic circuit in that space and finally fabricate it. This PCB can then be bent into the original input shape. We chose to work with an LED chain circuit, using widely available addressable Red-Green-Blue (RGB) LED. This process results in a low-resolution freeform RGB display, that can be controlled to produce interesting lighting effects.

An LED chain is a simple electronic circuit: each LED is connected to the previous and next ones in the chain, and every one of them are connected to

(a) Freeform Light-Emitting Diode (LED) sphere by Jiří Praus from Instructables.



(b) Freeform square LED matrix by Mohit Bhoite from his personal website.



(c) Closeup of *The Clock* by Gislain Benoit from his personal website.

Figure 1.3: Examples of freeform electronics by different makers.

power and ground. The chain starts and ends at a connector. Only the nature of the circuit, an LED loop, is specified. We do not know in advance how many LED will contain the circuit, or which LED will be adjacent in the chain.

1.2.2 Procedural generation of 3D printing supports

AM technologies have become available to the larger public in the last decades. The ease of use of consumer-grade printers, the emergence of online 3D printing services, and the many ready-to-print models available online either for free or

for a small fee have made it easy for anyone to 3D print objects. Many printing methods fabricate objects by depositing material slice by slice, constructing the object bottom-to-top. Due to this, unsupported features and steep overhangs cannot be printed without additional structures supporting them.

The importance of supports in 3D printing cannot be understated [JXS18]. While they enable the fabrication of complex geometries, they require manual effort to be removed and they result in wasted material that takes up printing time. Also, after removal they can tarnish the final appearance of the print by leaving scars at the contact point between the supports and the object. Thus, finding good types of support structures and optimizing them to work well with the target fabrication technologies is a key part of additive manufacturing research.

Our algorithm procedurally generates a scaffolding structure consisting of bridges and pillars supporting chosen points on the object. The class of generated structures are defined only locally: any small region of any output structure looks like a member of a pre-designed set of shapes. While this defines the overall nature of the resulting structures, these still have to satisfy functional requirements such as supporting the input points or connecting to the printing bed.

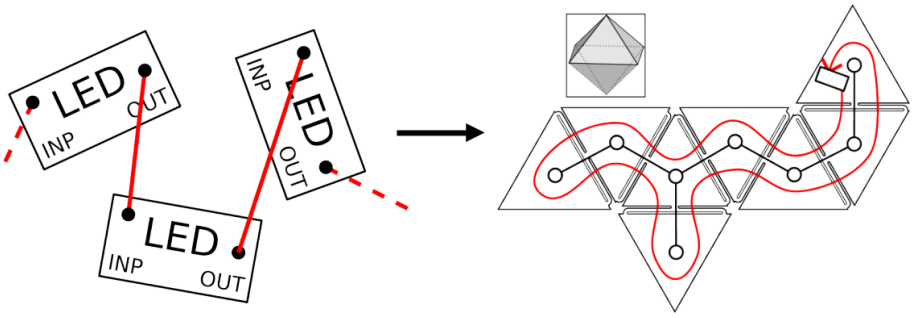
1.3 Positioning

Although dissimilar at first glance, these two problems can be viewed through the same lens.

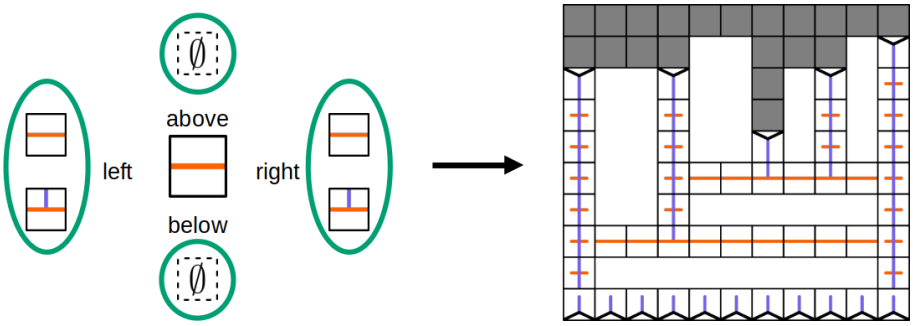
In both cases, we first frame the issue as a layout problem, which we then solve by constructing solutions defined locally, using properties derived from these definitions to design targeted synthesis algorithms. The local definitions of our solutions take the form of *topological constraints* specifying how the objects of our layout problem can be assembled together. From there, our methods generate layouts from the solutions, i.e. geometric embeddings that can be fabricated. However, the exact final geometry, its size and structure, is only implicitly **derived** from the topological constraints.

Circuit layout generation is clearly a layout problem, with electronic components having to be placed within the circuit board while taking electrical interconnects into account. We consider solutions in the shape of a loop. Locally, LEDs are chained, each having an LED before and after. Specific information such as how many LEDs the circuit has, how they are connected, and the exact physical layout are generated by our algorithm. Framing support generation for 3D printing as a layout problem requires some additional work. We work by tiling 3D space with a set of building blocks that define the support structure. These blocks come with additional information specifying how they can be locally assembled. All of this defines a space of possible support structures, among which we synthesize a solution. These approaches are loosely represented in Figure 1.4.

The goal of this thesis is to provide a useful resource for solving layout problems arising in the context of fabrication-oriented design, by putting the focus on local topological constraints and how they can be used to simplify these problems.



(a) Circuit layout generation.



(b) Support generation for 3D printing.

Figure 1.4: Schematic representation of the approach used to tackle the problems in this thesis.

1.4 Publications

This PhD work has resulted in the following publications:

- Marco Freire et al. "Procedural Bridges-and-Pillars Support Generation". In: *Eurographics 2022 - 43rd annual conference of the european association for computer graphics* (Apr. 2022), 4 pages. DOI: 10.2312/EGS.20221025;
- Marco Freire et al. "PCBend: Light up Your 3D Shapes with Foldable Circuit Boards". In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–16. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3592411.

The work on the procedural generation of supports for 3D printing was carried out during the first year of my PhD through the multiple COVID-19 lockdowns. It was a joint work with Samuel Hornus, Salim Perchy and my advisor Sylvain Lefebvre, with a lot of technical assistance from Pierre-Alexandre Hugron and Pierre Bedell. This approach was implemented in a beta version of the MFX research team's slicer *IceSL* [INR13].

The work on LED-based 3D displays is the result of a two-year-long collaboration between Camille Schreck, Pierre-Alexandre Hugron, my advisor Sylvain Lefebvre and myself, with Manas Bhargava as a joint first author and his advisor Bernd Bickel from the Institute of Science and Technology Austria. While most of the work was done jointly, it is natural in long projects involving multiple people that different people focus on different aspects according to their expertise and affinities. In particular, Manas played a significant role on mesh unfolding, bending experiments and LED placement, while I specifically focused

on hinge design, LED routing and layout generation. The code and data resulting from this work are available at https://github.com/mfremer/pcbend. The submission video for the article showcases many lighting effects and is available at https://youtu.be/g8UX-KifGmM. The submission data is available at https://mybox.inria.fr/d/cabc196c89704ec090e6/, it contains the SVG files and Gerber files used to fabricate all of the objects showcased in the corresponding chapter.

Manas and I gave a presentation on the article at the SIGGRAPH 2023 conference in Los Angeles, California. We also presented our fabricated objects at the *Bring Your Bunny (or something)* fabrication meet-up there. This was a great opportunity to showcase the objects themselves and the lighting effects to the computational fabrication and computer graphics communities, enjoy the objects brought by everybody and exchange about our respective works.

This project has also been mentioned in other media: an article on *Inria*'s website (english, french), a mention in an article (in french) by *L'Usine Nouvelle* (weekly French business magazine), and an article in *La Semaine* (weekly regional French journal).

Chapter 2

Related work

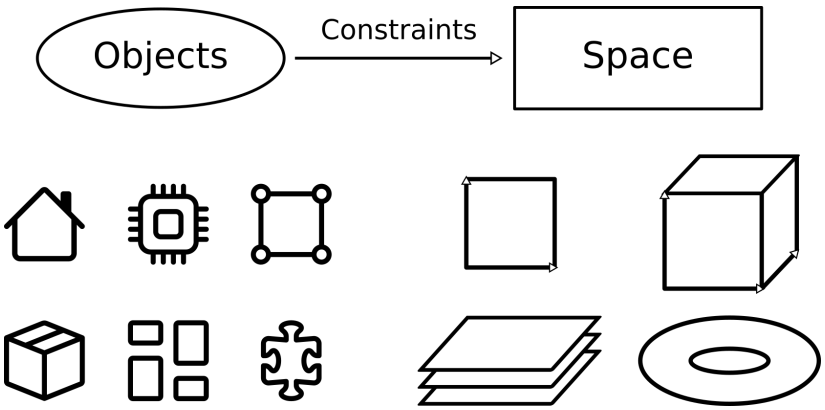


Figure 2.1: Illustration of an abstract layout problem

This chapter first introduces the general concept of layout problem considered in this thesis. Later, layout problems are considered in the context of electronics design automation and procedural design, the two main fields relevant to the contributions of this work. Electronics design automation is explored through a historical lens, highlighting the importance of computer automation and design standardization in the development of the field, two key elements in the resolution of layout problems in general. Procedural design is presented through a classification of different approaches and how they handle structured output, given that valid layouts are rarely arbitrary.

2.1 Abstract layout

This section clarifies the concept of *layout problem* used in this thesis. First, I use packing problems as an intuitive example to introduce geometric layout problems, and then graph drawing to illustrate layout problems with topological constraints. Packing problems and graph drawing are two representative examples of layout problems, often unburdened from constraints coming from industrial or engineering applications.

2.1.1 General concept

In this thesis, we define an *abstract layout problem* as consisting of three essential elements (see Figure 2.1):

- a space;
- a set of objects;
- a set of constraints.

The solution to this type of problem is a physical arrangement of the objects in the space satisfying the hard constraints and optimizing the soft constraints, i.e. objectives. Confusingly, the solution to a layout problem is also often called a layout. I use the term *valid layout* to refer to a *layout instance* that solves the problem.

The specific space, objects and constraints depend on the specific context of the layout problem, which can arise in a variety of different situations.

Spaces often are a bounded subregion of a 2D or 3D space, but can be more complicated than that. For example in electronics, for multi-layer circuit routing, copper traces can lie in any conductive layer of the circuits and can travel to other layers through conductive holes between them. Objects are usually straightforward to identify given the context. Finally, constraints come in many flavors. Hard constraints determine what type of layout instances are valid or not, i.e. acceptable as a solution in the context of the application. Soft contraints or objectives help specify what valid layouts are preferable to others.

A lot of situations can be interpreted as layout problems, so there is no canonical way to tackle them. Most layout problems except the simplest ones are often NP-hard, requiring heuristic or optimization-based approaches to solve them.

2.1.2 Packing problems

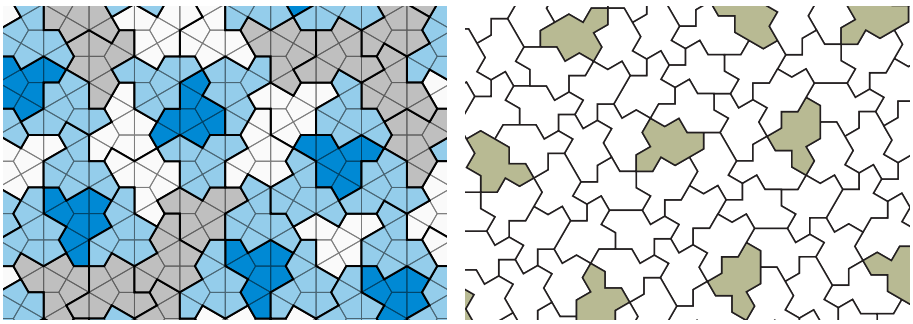
Packing problems serve as a great introduction to layout problems. In my opinion, they are the simplest to describe, as many people deal with them in their day-to-day life.

In packing problems, a set of shapes have to be put into a container without overlaps. Different methods and properties have been established depending on the type of shapes involved. Packing is often characterized by constraints on the space (2D, 3D, bounded, unbounded) or the considered shapes (rectangular, circular, irregular). Tessellations or tilings are packings that cover the space without gaps. [TOG17] provides an overview of packing and tilings from a discrete mathematics point of view. [WHS07] establishes a typology of cutting and packing problems, separating them into basic types according to criteria such as the optimization goal and the properties of the shapes and the container. Many of these problems are NP-hard, among those are the Knapsack problem [Pis05], or the bin packing problem [GJ79].

Packing problems are not only purely theoretical despite the simplicity of their description. For example, they find applications in computer graphics. [Lév+02] contributes a packing algorithm to merge irregularly-shaped parameterized charts into a single texture atlas. Packing algorithms are also heavily used in industrial fabrication applications. [RL22] provides a look at packing and cutting problems in an industrial context, from the modeling of the problem, different approaches to solve them, to specific application cases.

Packing problems are the embodiment of geometric layout problems. Constraints are purely geometrical, there is no notion of adjacency between different objects, or connectivity. Adding this type of concerns leads to layout problems such as graph drawing.

Additionally, tiling problems made news in late 2022 due to the discovery of the *hat* tile by David Smith, a hobbyist mathematician. This shape produces a tiling of the plane with no arbitrarily large periodic parts, i.e. it is an aperiodic monotile. This discovery and later the article [Smi+23b] proving this property solved the longstanding *einstein* problem. The resulting tiling uses both the hat and its reflection. The *spectre* family of shapes, derived from the hat [Smi+23a] was discovered a few months later. These shapes tile the plane without mirrored versions of itself, making it a chiral aperiodic monotile. Both tilings are illustrated in Figure 2.2. Tilings are explored in detail in [GS87; Fat21].

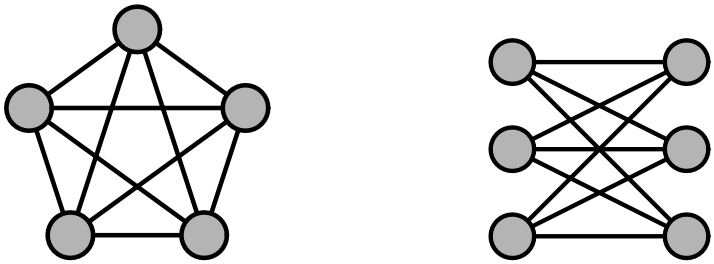


(a) Hat tiling. (b) Spectre tiling.

Figure 2.2: Illustrations of tilings made with the hat and spectre tiles respectively. Images by D. Smith, J. S. Myers, C. Kaplan and C. Goodman-Strauss, licensed under CC BY 4.0 Deed.

2.1.3 Graph drawing

Graph drawing problems introduces the idea of adjacency between objects by adding edges between some of them, adding a topological component to the layout problem. [Tam13] provides a comprehensive overview of graph drawing and visualization. Graph drawing encompasses a large class of problems. Most relevant to layout are planarity testing, crossing minimization and graph embeddings, which I will be going over in this section.



(a) K_5 (b) $K_{3,3}$

Figure 2.3: Illustration of K_5 and $K_{3,3}$, the forbidden minors characterizing graph planarity.

Planarity testing and embedding studies the properties of planar graphs. Planar graphs are graphs that can be drawn in the plane without crossings: distinct edges can only intersect at common endpoints. While easy to define informally, proper reasoning about planarity requires topological tools to state a rigorous definition of a graph drawing. Wagner [Wag37] provided in 1937 a simple characterization of planar graphs based on minors. A minor is obtained from the original graph by vertex or edge deletion, or edge contraction. Wagner's characterization is the following: a graph is planar if and only if its minors include neither K_5 or $K_{3,3}$ (see Figure 2.3). The first linear algorithm for planarity was provided fifty-one years later in 1974 by Hopcroft and Tarjan [HT74]. These results are fundamental to layout problems with topological constraints: they state that some graphs fundamentally cannot be embedded without crossings in the plane, meaning that some layout problems are unsolvable as is.

Among other topics, topological graph theory [MT01; Whi01] extends the study of graph embeddings to topological spaces more complex than the plane

such as general two-dimensional surfaces (topological 2-manifolds). They define multiple topological parameters for graphs: the crossing number, the thickness or the genus among others. The crossing number counts the minimum number of edge intersections among all drawings of the graph in the plane, measuring how far the graph is from being planar. The thickness is the minimum number of planar subgraphs whose union is the original graph. These two numbers are particularly relevant for electronic circuit layout. Using the classification theorem [DH07; Bra21] for compact connected orientable surfaces stating that any such surface is homeomorphic to a sphere with $g \geq 0$ handles, g being the genus of the surface. The orientable surface of genus 0 and 1 are the sphere and the torus respectively, and the surface of genus g is "a torus with g holes". Planar graphs are exactly those that can be embedded on the sphere (the genus 0 surface). A key result in topological graph theory is that any finite graph can be embedded without crossings into a surface with a large enough genus. The minimum genus of this surface is the genus of the graph.

It has been proven that computing the crossing number [GJ83], the thickness [Man83] and the genus [Tho89] of a graph is NP-hard. Not only that, but even adding one edge to planar graphs makes computing the crossing number NP-hard [CM12]! However, given a surface, there is a linear time algorithm to compute an embedding of a graph, or if none exist exhibit an obstructing minor [Moh99; KMR08].

Wagner's theorem mentioned above is a weaker version of a much stronger result, the Robertson-Seymour theorem, proved in the twentieth paper [RS04] of a series of twenty-three [RS09], constituting the *Graph Minors project*. This theorem states that every minor-closed family of (finite) graphs admits a forbidden minor characterization. In other words, given a family of graphs defined by a property that is preserved by taking a minor of the graph, there exists a finite set of forbidden minors such that, if a graph has any of them as a minor, then it cannot have the property. In particular, for any fixed surface, graphs embeddable in it can be defined by a family of forbidden minors. This does not provide a computable approach for graph embeddings, as for surfaces as simple as the torus, the forbidden minors set is not known and might be impractically large.

Additionally, mathematicians have studied since the 1960s *thick embeddings* [Bar93] of graphs into three-(and higher [GG12]) dimensional Euclidean space. This type of embeddings considers the thickness of the geometric realization of the graph, i.e. nodes are spheres and edges connecting them are cylinders, both with a non-zero radius. Some graphs can be packed more tightly than others in the same volume depending on properties such as their maximal degree. This approach has found applications in graph drawing, in particular an algorithm for 3D orthogonal graph drawings [ESW96]. Considering the physical dimensions of objects is an essential part of solving layout problems.

More complex relationships between objects can be represented with hypergraphs, as hyperedges can connect any number of nodes. A common approach to deal with them consists in transforming the hypergraph into a regular graph by replacing hyperedges with a hypervertex, connected to all vertices incident to the hyperedge by a star or a tree. These transformations are particularly common in (hyper)graphs representing electronic circuits. Indeed, hypergraph partitioning and clustering have been studied at length [PM07], and are commonly used to break down the complexity of problems arising in Very Large Scale

Integration (VLSI) layout [Kah+11]. Also, Chimani and Gutwenger [CG07] introduce the hypergraph crossing number problem, based on the minor crossing number [BFM06] which is itself a minor-monotone generalization of the traditional crossing number defined for graphs. They provide an algorithm for edge insertion in hypergraphs while minimizing the minor crossing number, relevant for electronic circuits. Hypergraphs have been traditionally less studied due to their permissive structure, making it harder to establish properties and design algorithms around it. Keep in mind that many graph problems are already computationally hard, and this gets even worse for hypergraphs. In the last few years, they have garnered the interest of the machine learning community, as they expand from graph learning [Ham20; Wu+22] to hypergraph learning [Gao+21].

Summary

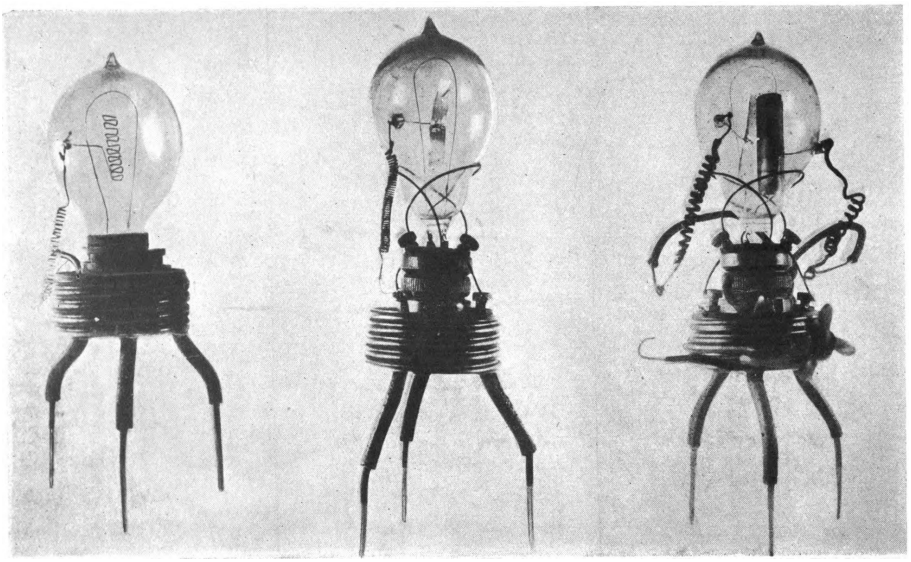
Topological constraints add a significant amount of complexity to layout problems. Whereas in packing problems, the main goal is often to minimize cost by maximizing packing density, here we have hard connectivity constraints that can prevent valid layout from even existing. On the other hand, these constraints highly restrict the space of valid layouts, making some search strategies relying on the restricted structure of the solutions possible. The following sections in this chapter explore how layout problems arise in the fields of Electronic Design Automation (EDA) and Generative Design (GD), which are central to the contributions of this thesis.

2.2 Layout and electronic design automation

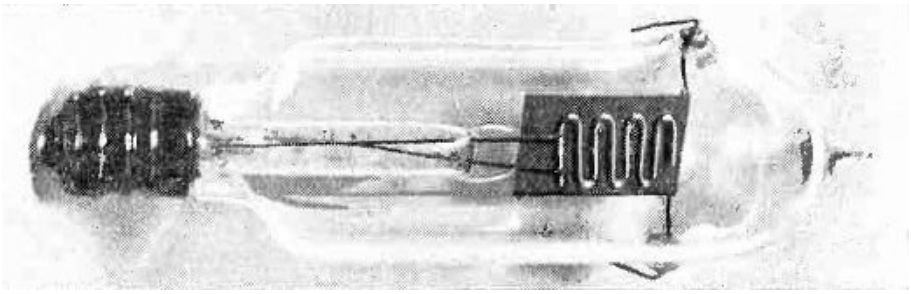
In this section, I go over the history of modern electronic circuits from the 1950s whose rapid development prompted the birth of EDA tools and research, in which layout problems are central. This history is mainly told here from an American point-of-view. The development of electronics in other parts of the world is a deeply interesting topic, please refer to other resources for a more global and complete history.

2.2.1 A short history of microelectronics

To understand the role of EDA in the modern electronics industry, one first has to understand the evolution of the latter during the twentieth century. This evolution was mainly enabled by the switch from vacuum tubes to transistors, and from discrete electronics with point-to-point connections to printed circuits and integrated circuits. These technological breakthroughs made electronics easy and cheap to mass-produce, and allowed a level of circuit complexity never seen before. This radical change required research and investment in computer-assisted design of electronics, leading to the rise of EDA and the creation of software tools now commonplace both at the industrial and amateur level.



(a) First prototype Fleming valves (1904).



(b) First prototype Audion tube (1906).

Figure 2.4: Early versions of vacuum tubes. Images from Wikimedia Commons (a), (b), licensed under CC0.

2.2.1.1 Smaller and smaller: the transistor and miniaturization

The early-twentieth century equivalent of transistors is the vacuum tube, more specifically the thermionic triode [Gua12; DM22]. The vacuum tube diode (Fleming valve) was invented by Fleming in 1904, and later the triode (Audion tube) in 1906 by de Forest. The triode was the first device able to amplify a signal, and became widely used in communications. These tubes consist of a hot filament (cathode) that emit electrons towards a metallic plate (anode) by thermionic emission, all in glass-encased vacuum. This basic design is a *di*ode containing only two electrodes. Current only flows in one direction since electrons cannot be emitted from the anode. These are shown in Figure 2.4. *Tri*odes, *tetr*odes and so on can be created by inserting additional electrodes (grids) between the cathode and anode. By applying a voltage to the grids, it is possible to control the magnitude of the current between the cathode and the anode. A triode with a single grid can then act either as an amplifier or a switch, the latter being the main building block of digital electronics. Vacuum tubes were fragile, energy-hungry and bulky, prompting a search for better

alternatives.

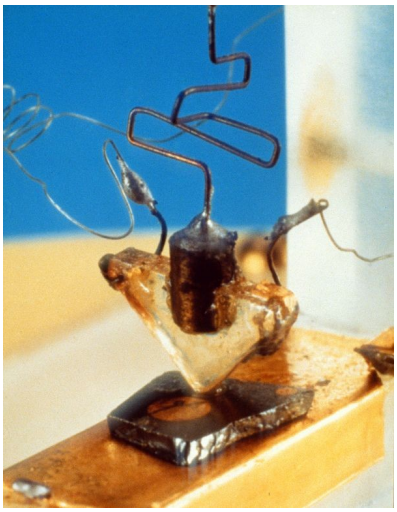


Figure 2.5: Bardeen and Brattain's first point-contact transistor (1947). Image from the Computer History Museum's website, Copyright Alcatel-Lucent USA Inc.

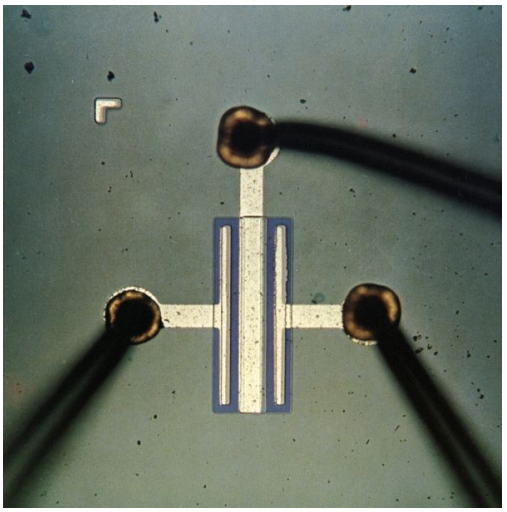


Figure 2.6: Fairchild's FI100 MOS transistor (1964). Image from the Computer History Museum's website. Copyright Fairchild Camera and Instrument Corporation.

This search culminated in the demonstration of the bipolar transistor in 1947 by Bardeen and Brattain in its point-contact variant (see Figure 2.5), and later in 1948 by Shockley in its junction variant. These used germanium initially, the switch to silicon came later in the 1950s. They obtained the 1956 Nobel prize in physics "*for their researches on semiconductors and their discovery of their transistor effect*" ([Nob24]), providing a solid-state alternative to vacuum tubes. The germanium bipolar junction transistor was also independently invented by German physicists Herbert Mataré and Heinrich Welker in 1948 in Paris, dubbed the *transistron* [Com24a; Dor04]. Finally, the modern transistor or Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) was invented by Atalla and Kahng in 1959, based in the concept of FET proposed by Lilienfeld in 1925 [Lil30]. Apart from being significantly more reliable than vacuum tubes, these transistors can be fabricated cheaper at a much smaller scale and support higher frequencies. This enables more complex circuits to be built thanks in part to their simple structure (see Figure 2.6). It is interesting to note that early semiconductor and computing research was largely funded by US government contracts and the Department of Defense [Com+99], the proportion tapering down during the late 1960s.

2.2.1.2 Larger and larger: printed and integrated circuits

Eleven years after the demonstration of the first working (bipolar) transistor in 1947 at Bell Labs by Bardeen, Brattain and Shockley; the vice-president of electronic technology at Bell Labs wrote an article on the impact of the development of transistors in electronic design. In a now famous quote he states:

> *For some time now, [...] electronic man has known how* in principle *to extend his visual, tactile and mental abilities to the digital*

> *transmission and processing of all kinds of information. However, all these functions suffer from what has been called 'the tyranny of numbers'. Such systems, because of their complex digital nature, require hundreds, thousands, and sometimes tens of thousands of electron devices.* ([MP58])

In context, this refers to the difficulty of designing electronic systems with vacuum tubes and the ensuing reliability issues and serves as an introduction of the transistor as a partial solution to this problem. This observation about the complexity of electronic systems remains relevant today and designing tools to handle this complexity is the main motivation behind the development of EDA. This complexity calls for miniaturization, as Feynman remarks: "*I do know that computing machines are very large; they fill rooms [...] the possibilities of computers are very interesting — if they could be made to be more complicated by several orders of magnitude.*" ([Fey60])

The transition from discrete components with point-to-point connections to PCB and Integrated Circuit (IC) is essential for modern electronics. Printed circuits were mainly developed by Paul Eisler, an Austrian engineer who emigrated to Britain in 1936 after the rise to power of Austrofascism. Other attempts to create printed circuits were carried out before [Har03], but did not become widespread. Eisler recounts in his autobiography [EW89], how he combined his engineering experience with the knowledge on printing technology he acquired working as a technical editor in the press to come up with the (literally) *printed* circuit concept. He recognized the need in the telecommunications industry for an easier and cheaper method for building circuits, and his aim became "*to produce a circuit board onto which strips of metal could be adhered using a printing process*". The first demonstration of this concept was a fully-working radio he built in 1936, relying on a handmade circuit board. He presented this prototype to british company Plessey's director in charge of radio production, who rejected his invention. As Eisler himself says in his autobiography:

Figure 2.7: Radio with the first Printed Circuit Board (PCB) by Paul Eisler (1942). Image from the Science & Society Picture Library website. Copyright Science Museum / Science & Society Picture Library.

> *The reason given for refusal was unexpected [...]. It was pointed out to me that the work which my invention would help replace was carried out by girls and 'girls are cheaper and more flexible'.*

In the following years, he continued thinking about how to automate the fabrication process and how to frame his invention within the war effort for the emerging World War II, to which most economic resources were dedicated at the time. During these years, he developed the *foil technique* in which

insulator plates clad with a conductive foil have the circuit pattern printed on with resistant inks and the rest of the foil is chemically etched, leaving behind only the desired pattern. In 1942, he demonstrated this new technique for the first time (see Figure 2.7), garnering little interest at first. These demonstrations gained him new contacts, and he ended up reaching engineers and military personnel, employees from electronic firms and ministries related to war production. Once again, nobody saw use for printed circuits in military equipment or anywhere else. However, American military also attended these demonstrations, and later made use of printed circuits in proximity fuses for war applications. According to a 1947 circular [BC47] from the United States' National Bureau of Standards, printed circuit mass-production started in 1945. Nowadays, printed circuit boards are ubiquitous and used for many different applications.

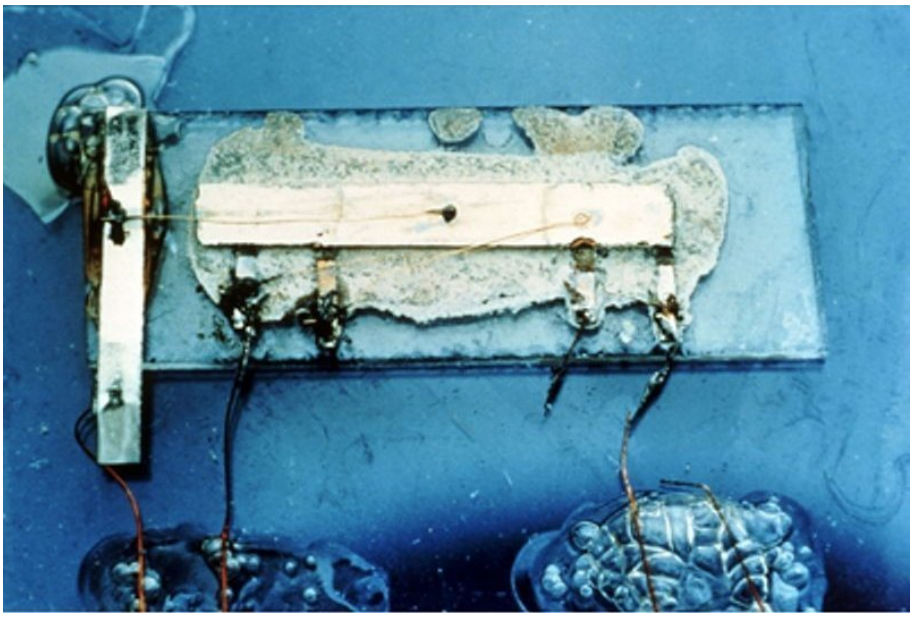


Figure 2.8: Kilby's original hybrid IC, with flying wire connections (1958). Image from the Computer History Museum's website. Copyright Texas Instruments Inc.

The second critical innovation is the development of the IC, attributed in [HC21] to the military's increasing need for electronics miniaturization in arms development. This invention has been attributed to multiple people: to Jack Kilby for the first hybrid IC (i.e. separate integrated components connected with flying wires) in 1958 (see Figure 2.8), to Robert Noyce for the first monolithic IC (i.e. fully integrated) in 1959 (see Figure 2.9). Later [Loj06] among others highlighted the essential contributions of people such as Jean Hoerni and Kurt Lehovec. Legal battles over patents on the IC technologies ensued during the sixties. Integration in electronics means that all components and interconnects are embedded or *integrated* into a common substrate, most often silicon. Compared to printed circuit boards, integrated circuits have three main advantages: size, cost and performance. Integrated transistors can be fabricated much smaller than discrete transistors: the typical dimensions of a surface-mounted small-outline transistor (SOT23) are $2.9 \times 1.3 \times 1$ mm [NXP17],

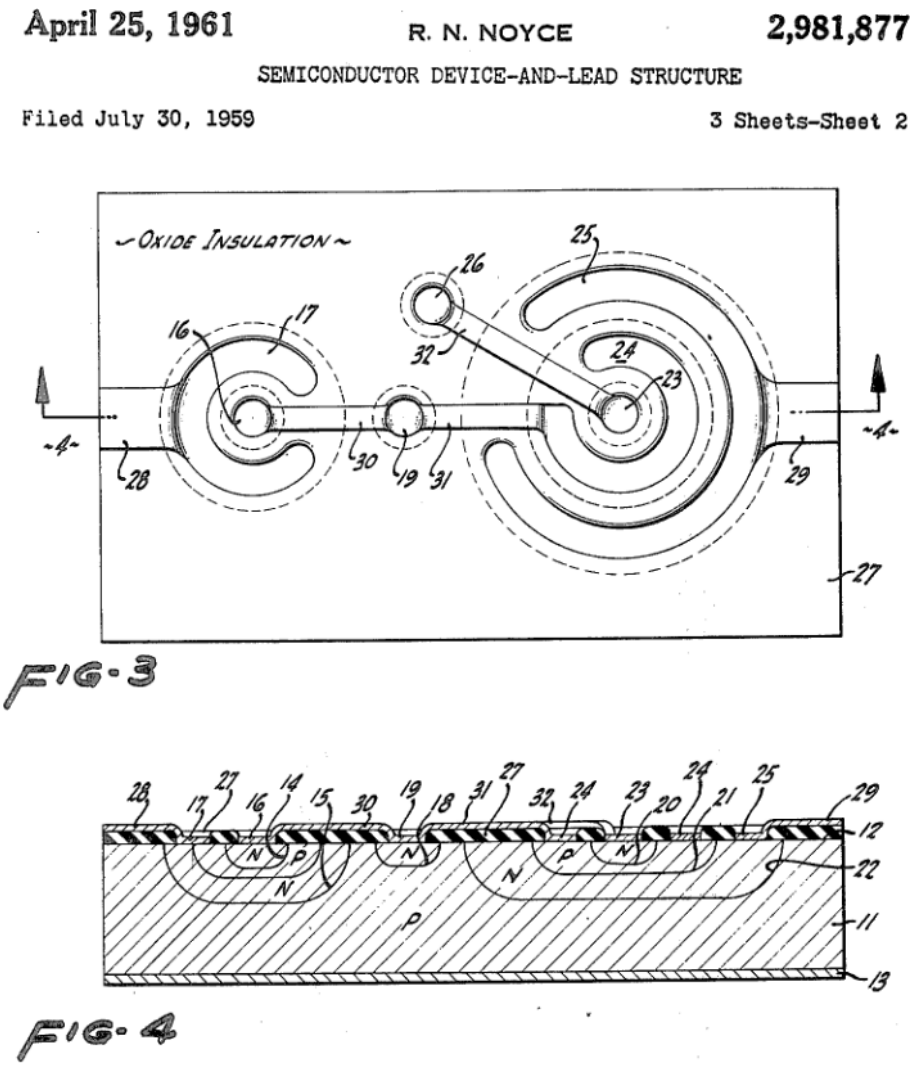


Figure 2.9: Figures from Noyce's patent [Noy61] illustrating a monolithic IC (1959).

while the dimensions of current-day integrated transistors are measured in tens of nanometers [MS22]. They also support higher frequencies and require less power to operate.

In 1965, Moore made the observation (later dubbed *Moore's law*) that the level of integrated chip complexity that can be manufactured for minimal cost doubled every year [Moo65], an observation that he readjusted in 1975 to a doubling every two years. As Moore himself remarked, "*it has become a self-fulfilling prophecy*", as the industry adopted it as a growth target. Hutcheson [Dan05] provides a more detailed historical and economic perspective on Moore's law. Additionally, Dennard scaling [Den+74] states that the power per mm^2 of MOSFET transistors stays constant as their size gets smaller. Combined with Moore's law, this provided a scaling law for performance per joule, doubling approximately every year and a half. This type of progress through

Figure 2.10: Rubylith operators (1970 ca.) from the Computer History Museum website. Copyright Intel Corporation.

miniaturization is specific to microelectronics, where making systems smaller makes them simultaneously faster, cheaper, more economical and surprisingly, also more reliable [LS20]. Indeed, at larger scales of integration, fewer discrete components are required to assemble a circuit, thus reducing the number of possible points of failure. Moore's law starts showing signs of slowing [HP19] in the 2000s due to industry nearing scales where fundamental limits of the transistor fabrication process arise. Dennard scaling does around 2007 due to unforeseen power dissipation at the nanometric scale leading to thermal problems [Boh07]. This required the industry to change their design methodology to keep improving Power-Performance-Area-Cost (PPAC) metrics. The intricate layout of integrated circuits makes their design a huge challenge, requiring a high level of expertise as well as dedicated EDA tools. The next section summarizes the history of EDA.

2.2.2 A short history of circuit design

This section summarizes the history of EDA and in particular how chip planning and physical design for digital circuits evolved along with it starting in the 1950s. It is important to note that much effort in the field was directed to the development of tools for circuit specification, synthesis, simulation and verification, which I will be mostly overlooking in this section. For a more transversal view of the evolution of the field up to 1988, please refer to [Com88]'s preface.

Already before the invention of the integrated circuit (see Section 2.2.1.2), IBM engineers and designers realize that "*In the design and development of today's complex computers, the ratio of routine and repetitive work to creative engineering is getting larger and larger*" ([KCG58]). This is the design

counterpart to the *tyranny of numbers* observation made at the same time. For this reason, they used early computers to assist with the repetitive parts of electronics design, with a system they dubbed the "*Design Mechanization System*". In this system, creative tasks such as engineering and logic design were done manually, whereas mechanical tasks such as record keeping, logic checking, wiring planning and documentation printing were done with a computer. This was partly enabled [KCG58; Boy04] by the use of their *Standard Modular System* in the 1950s, progressively switching to the more advanced *Solid Logic Technology* (SLT) at the end of the 1960s. Their computers consisted of arrays of standardized card-mounted electronic components, connected by wire wrapping in a backplane. The switch to the denser SLT modules was done because monolithic integrated circuit technology was deemed not mature enough at the time for production.

When integrated circuits started gaining traction during the 1960s, coordinated efforts in design automation started to emerge, gathering experts across industry, engineering and research. The main venue for design automation research starting in 1964 was the ACM/IEEE Design Automation Conference, where many groundbreaking works were published during this era. For placement and routing, such works include continuous planar routing [Hig69], channel routing for printed circuit boards [HS71] and standard cell designs [KSP73], force-directed placement [Qui75] or arbitrarily-shaped blocks placement [Pv79]. Full design IC systems started appearing, such as the workstation-based LTX system [PDS76] for LSI layout outputting mask geometry using circuit component and connectivity descriptions as input. Another design technique developed during this time is symbolic layout with compaction [GN76; CKS77; Dun80]. Instead of directly dealing with mask layout, i.e. the intricate geometric data used to fabricate photomasks, they create an abstract or symbolic representation, where groups of primitives are represented each with a different symbol. The layout is then drafted in a coarser grid, where the main information is adjacency and connectivity between symbols, making it essentially a topological representation. Symbolic translation then instantiates the geometric primitives, and compaction packs them as tightly as possible to create an area-efficient layout. Procedural circuit design started with the first *silicon compiler* [Joh79], where circuits were assembled from flexible blocks represented by programs, instead of the fixed cells from standard libraries. The invention of optimization by simulated annealing [KGV83] led to great success in different aspects of circuit layout. [San03] considers this time as the foundational period for EDA, where the bases were laid for the fruitful development of this field.

Still, it is important to remember that chip layouts at the time (1970s) were mostly hand-drawn. Faggin explains the process at the time in his account [Fag09] of the development of the first microprocessor (the Intel 4004 released in 1971) as its lead designer. Layouts were manually drafted at a large scale on graph paper, and then photographically reduced to fabricate masks for manufacturing (see Figure 2.10). The exponential growth in complexity of IC rendered this manual approach prohibitively inefficient. As Lienig et al. remark, this "*implies that the human designers of electronic systems need to improve their productivity at the same exponential rate*" ([LB17]). Jansen recalls that this "*first generation of EDA did not contain a tool for automation of an implementation step*" ([Jan03]).

Different design styles emerged during this period [New82; Ein85; NS87],

some more amenable than others to automation. These can be categorized into full custom and semi-custom, which includes standard cell [Com24b], macro cell [Tok+88], and gate array designs [Com24b]. These styles have different design and fabrication costs, different degrees of design reuse, result in more or less efficient layouts, and need different production volumes to be economically viable. *Full custom* design is the least constrained, blocks can be placed anywhere and routed in any fashion. This allows full control over the final layout, making high-performance, high-efficiency designs possible at a great cost. Constrained design styles such as *standard cell* or *gate array* require significantly less design effort and are easier to create automated tools for. In standard cell design, the basic building blocks are small cells consisting of a few transistors implementing simple functions (such as logic gates or flip-flops). Cells have all the same height, are placed in rows, and mostly connected together using the space between the rows (channels). *Gate array* designs have basic logic elements arranged in a fixed regular grid, and different interconnect patterns can generate different circuits. Field-Programmable Gate Array (FPGA) are a special case of gate array, where both the basic logic elements and the interconnects are field-programmable instead of mask-programmable, i.e. the circuit can be reconfigured after fabrication. Finally, *macro cell* designs consist of multiple interconnected large functional blocks, typically used for components for which other approaches are inefficient (such as standard cell for memory circuits). The advantages and disadvantages of each style are detailed further in [LS20]. Standard cells are found in technology libraries distributed by the foundry, i.e. the semiconductor fabrication plant. Macro cell libraries are marketed by semiconductor intellectual property vendors, so called because they are traded as rights to use and copy the design.

The switch to fully computer-generated layouts was partly due to the VLSI revolution at the end of the 1970s, mainly carried by Lynn Conway and Carver Mead (who gave *Moore's law* its name). As Conway recalls in [Con12], the crucial contribution was the invention of scalable design rules. Before then, integrated circuit design mostly relied on arcane design rules set by the different fabricators, leading to long turnaround times and requiring designers to familiarize themselves with new rulesets every few years due to process progress and technique changes. Additionally, the design's surface area was mostly covered with connections instead of transistors, minimizing the benefits of miniaturization. Conway designed a set of rules based on a minimum feature length λ dependent on the process resolution, and every design rule was defined as a multiple of λ. Thus, different manufacturing technologies could be targeted by changing a single parameter. Instead of targeting heavy layout compaction for each fabrication technique with custom-tailored design rules, she thought that designing a simpler, unified set of rules would help new and veteran designers to speed up and make creating layouts easier. The responsibility of performance improvements were left to Moore's law and Dennard scaling. This ushered the VLSI revolution, culminating in the exposition of this new methodology in the first accessible VLSI design textbook [MC80]. These efforts were a collaboration between Xerox Palo Alto Research Center and Caltech, which lead to the creation of the Defense Advanced Research Projects Agency (DARPA)-funded VLSI project starting in 1978 [Com+99]. Her career in computer architecture started at IBM in the 60s, from which she was fired due to her gender transition, causing her to restart her professional career from scratch. This prevented her from

getting credit for dynamic instruction scheduling, the basis behind superscalar processors which she developed during that time, until the early 2000s. This revolution resulted in a great simplification of the development of EDA tools, which was previously nearly impossible due to the rapid change of design rulesets and their complexity. In particular, this made checking for geometric design rule violations straightforward. This created a clear separation between chip design and fabrication, which were previously indissociable, also enabling design reuse. Finally, the Metal Oxide Semiconductor Implementation Service (MOSIS) created in 1981 was a result of this revolution. This system enabled american students enrolled in VLSI design courses to produce actual chips based on their assignments. They pioneered the modern fabless/foundry model, where chip designers send their designs to exterior semiconductor foundries which are separate business entities. This made fabrication services more accessible to academia, pushing research forward.

The appearance of different design styles and of standardized design rules are a reflection of the higher levels of abstraction required to design chips of increasing complexity [San03]. Through the 1960s and early 1970s, it was still possible and common to design chips at the transistor level. In the 1980s, circuits were designed around logic gates, and with the emergence of Hardware Description Languages later in the decade, at the Register Transfer Level (RTL). RTL modeling describes circuits as a flow of and operations on data between registers. Simpler design rules at higher degrees of abstraction meant easier design methodologies and shorter design times, at the cost of much more wasted space in chips [CKS77]. Layout efficiency being a crucial element of PPAC scaling, this targeted research efforts towards developing more efficient EDA tools for physical design, able to handle less constrained design styles. These movements and changes in design styles in the industry have been interpreted [Eur09] through Makimoto's wave model [Mak13]. This model breaks down innovation into an initial stage and a series of alternating waves. The initial stage consists of a disruptive phase, where the previous technology becomes obsolete, and an exponential growth phase. Then, the innovation process settles into alternating standardization and customization trends.

Reviewing the tightly coupled history of microelectronics developments and IC development shows interesting patterns. In particular, much progress has been driven by the realization that the enormous task of VLSI design requires automation, and slowed by the apparent difficulty of the problem. In the next section, I break down the physical layout design process and highlight its difficulty.

2.2.3 Physical layout design

Physical design is one of the steps in electronic systems design [LS20]. Electronics system design starts with a *specification*, where the functions, requirements and interactions of the system are established. This leads to *circuit design*, where a structural description is build based on the specification. This structural description consists of a schematic diagram or a netlist specifying what electrical units compose the design and how they are interconnected. This description is often hierarchical, breaking the design into modules at different levels of abstraction. Next is *physical design*, where the structural description is transformed into a fabrication specification (or layout). This specification

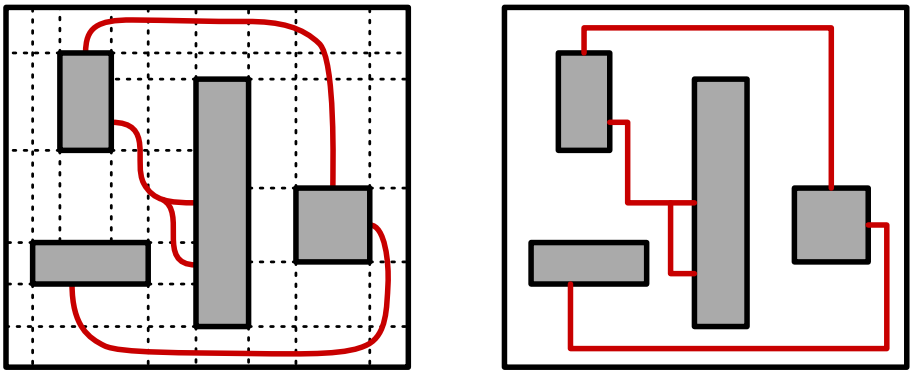
is finally used to manufacture the final circuit. Of course, circuit and physical design are heavily informed by fabrication process-specific knowledge. Physical layout thus takes a shapeless description of a circuit and turns it into a concrete realization, while taking into account a set of optimization goals and constraints. This realization contains the placement of the components, conductive traces, the shape and dimensions of the board or circuit, and all information necessary for the fabrication process. To ensure that it functions properly and that it is manufacturable, the layout has to satisfy electrical and design rules respectively.

Physical design has a significant impact in the PPAC metrics of the final circuit. For example, in a layout where components are farther from each other than they need be, wires will be longer than necessary, inducing signal delays that limit the clock frequency at which the circuit can operate. If the components are not tightly packed, the final design will take more space, and fewer circuits will be able to be fabricated per silicon wafer, making the fabrication cost higher. While this process is different for (digital) IC and PCB due to the different fabrication processes and possibilities of the medium, the main concepts, problems and techniques remain similar, so I will not make a distinction between them in this section. Additionally, this section will not go in-depth into the algorithmic aspects of these steps, as many references already cover this topic [SY01; She04; Kah+11]

The starting point of physical design is the netlist. It stores the connectivity of the electronic circuit by listing the nets in the circuit, i.e. the sets of interconnected components. In other words, the netlist represents the topology of the circuit. Physical design can be broken down into smaller steps: partitioning, floorplanning, placement and routing. Partitioning and floorplanning help separate the layout problem into smaller tractable instances. Partitioning takes the initial circuit and splits it into modules while minimizing inter-module connections. Floorplanning gives a shape and a location to every module, determining their arrangement in the final layout and their external connections. Due to them being particularly constrained, floorplanning also takes into consideration power and ground structures as well as clock planning. These two steps determine the external characteristics of the modules. Next, placement and routing are used to determine the internal module characteristics.

Placement determines the location and orientation of all basic units within the bounds of a module so that they do not overlap. Routing creates conductive traces between all of these units according to the connectivity information provided in the netlist. Both steps are limited by constraints and guided by optimization goals. Placement is tricky for a particular reason: a successful placement is (among other criteria) one that can be routed, but the only way to guarantee routability is to route the design. Unfortunately, this is way too expensive of an approach for most circuits. Indeed, in Chapter 3 we circumvent this problem by placing and routing small sections of the overall circuit instead of all at once. In the general case, placement optimization goals are designed to make it so good placement scores make it likely that the design will be routable. For example, placement often tries to minimize the estimated total length of connections between units, and to avoid routing congestions (i.e. areas where many interconnects might need to be routed). In good placements, tightly interconnected blocks will be close to each other: it is easy to imagine how convoluted the routing might be if these blocks were placed at opposite ends of the design. Similarly, by avoiding routing congestion, algorithms encourage

designs where routing is distributed all over, meaning that no gap will be too narrow for the number of connections having to go through.



(a) Global routing: the general routes interconnects follow are decided.

(b) Detailed routing: the exact position of every interconnect is computed.

Figure 2.11: Illustration of global and detailed routing in a schematic example.

Routing starts with a placement, and consists in realizing the interconnects specified in the netlist, usually with an upper bound on the number of available routing layers. The main problem with this step is that for a given placement, routing might be either impossible or too complex to route in a reasonable time. Power, ground and the clock nets are often routed first since they are particularly constrained. Deciding the order in which connections are routed is important to create a good layout. Often, connections that have already been routed have to be ripped up to allow other connections to be routed. Due to the size of the instances, for IC placement and routing are often split into a global and detailed phase. Global placement globally places the units without taking into account their shape and size with some overlaps allowed, then detailed placement and legalization aligns the units to a grid and resolves overlaps by performing local modifications of the global placement. Global routing starts with a subdivision of the circuit into coarse cells (channels or switchboxes), and computes an overall routing topology in these cells, i.e. computes the general trajectory of connections around the components. Thus, each net is assigned to a series of cells (within the limit of their capacity), and detailed routing precisely routes the trajectory of the connections within these cells (see Figure 2.11). It is interesting to note that placement and routing are closely related to hypergraph embeddings and topological parameters (such as the crossing number or the thickness) introduced in Section 2.1.

Unsurprisingly, all of these constrained optimization problems arising in physical design are NP-hard, even in simplified forms [SB80; SY01]. Additionally, their solution space often scales as an exponential or factorial function of the size of the input. For this reason, solving them most often requires heuristic approaches, where the optimization process is guided to avoid uninteresting regions of the search space and instead towards types of layouts that might solve the problem reasonably well. This means in particular that solutions obtained with heuristic methods are rarely globally optimal, but in exchange algorithms terminate in a reasonable time. This makes it hard to evaluate the quality of a solution, since there is no optimal solution to compare it against. Instead,

Figure 2.12: *NEVER trust the autorouter* t-shirt design. Created by and copyrighted to Chris Gammell, cropped from his 2014 twitter post.

quality assessment requires metrics that are often used as optimization goals during the process. This also explains the historical fact that no be-all end-all, canonical solution has been found to layout design, requiring instead constant algorithmic innovation to tackle harder and harder problems. Following this realization, Chapter 3 focuses on a specific class of circuit that can be broken down into smaller instances instead of trying to find valid layouts for general circuits. This allows us to consistently obtain good results in highly constrained spaces that would be otherwise impossible with general methods.

If all that were not enough, physical layout design methods do not operate in a geometric vacuum. The final results have to be fabricable, and function according to the specification. This gets harder as circuits get smaller and need to meet harsher performance standards. Reliability, thermal management and cooling, electromagnetic compatibility and recycling have to be taken into account when designing an electronic system. These concerns have an impact on every step of physical design, and add additional goals and constraints to the optimization process, making it harder to automate. [LB17] provides an overview of these concerns in general electronic system design, and [LS20] explains how to mitigate these negative effects through physical design choices.

2.2.4 The struggles of electronic design automation

Given the complicated history of microelectronics, the breadth of design styles and the industry paradigm changes, it is to be expected that EDA users express unsatisfaction towards EDA tools, which cannot catch up with their needs fast enough. Indeed, many professional and amateur electronics designers prefer to have manual control over the physical design process. In particular, PCB autorouters face harsh criticism, as exemplified by the phrase *never trust the*

autorouter that made it onto a t-shirt (see Figure 2.12). It is common to find people criticizing these tools in online spaces, and well-established companies developing EDA tools for PCB design such as Altium, Cadence or Autodesk are aware of this. Altium refers to the development of autorouters as a "*history of failed design automation*" ([Mar17]), Cadence [Cad22] calls for a change in the typical design workflow to take advantage of the capabilities of their autorouter, and Autodesk discourages users from "*forming an unhealthy dependency on your autorouter*" ([Sat17]). The bad reputation of autorouters is a testament to the complexity of the physical design process, in particular the placement and routing steps.

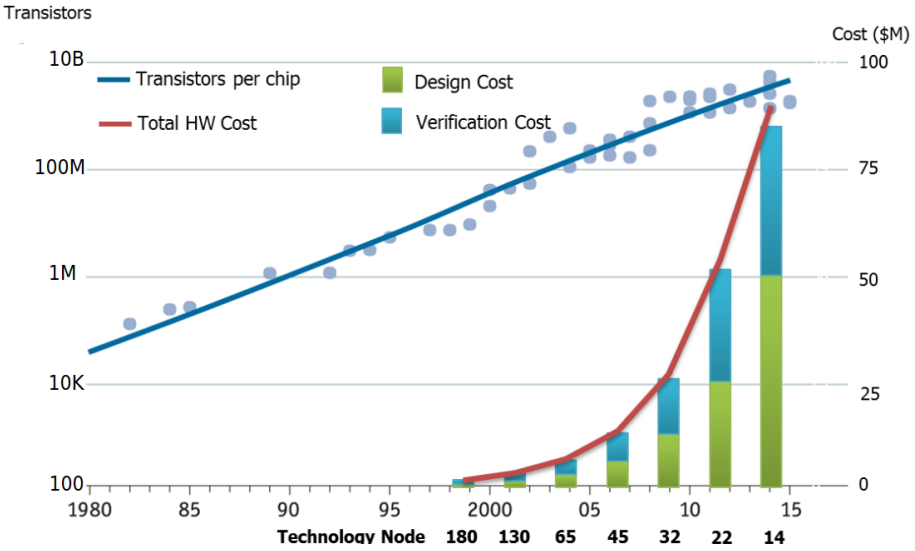


Figure 2.13: Evolution of transistor count (logarithmic scale) and design cost (linear scale) of IC over time, from [And18].

EDA tools for IC are not exempt from problems either. Already in 1997 the (United States') National Technology Roadmap for Semiconductors [Sem97] warned that the number of transistors on a chip was increasing at a much faster pace than the designer productivity improved, thus requiring growing numbers of designers on each project. The 2001 Technology Roadmap for Semiconductors reports that chip "*devote thousands of engineer-years (and a design team of hundreds) to a single design*" ([All+02]). This phenomenon is called the *design productivity gap* (see Figure 2.13) and actors in this sector have tried over the past decades to mitigate it. The roadmap also states that "*many advanced companies believe that the EDA industry continues to fall further behind in understanding the nature of current design problems*".

According to [LS20], many steps of the physical design process are still carried out by hand today. In particular, most analog circuits are designed manually due to the larger number of constraints in analog design compared to digital circuit design. Full-custom designs, high-density PCB are designed with tools assisting the manual design process. Critical elements of the layout also need to be carefully carried out by hand, such as floorplanning which handles a wide range of objects and constraints, or clock networks routing on whose quality the performance of a design heavily depends.

Summary

The discovery of the transistor and integration during the mid-twentieth century mark the beginning of modern electronics. Moore's law and Dennard scaling have guided technological developments until the late-2000s, effectively signifying that better performance could be achieved at a constant power per area exclusively through miniaturization. To cope with the increasing levels of circuit complexity resulting from these fabrication advances, computer assistance and automation soon became necessary starting in the 70s, giving rise to the field of EDA. EDA aims to provide tools to automate the circuit design process, in particular physical or layout design which consists in generating a geometric layout of a circuit from a topological description. Layout design encompasses many computationally hard problems such as placement and routing, which are great examples of concrete layout problems with real-world goals and constraints.

2.3 Layout and procedural design

This section provides a short overview of procedural or generative techniques in computer graphics, and how they relate to layout problems. Generative or procedural design is defined as "*a design approach that uses algorithms to generate designs*" ([CSL20]). This is an intentionally very broad description, coming from the term's use in computational design for architecture. It encompasses a wide range of approaches, mainly categorized into procedural noise, exemplar-based and tile-based. These have applications in many different areas such as shape modeling, texture synthesis, procedural content generation for video games or generative art.

Historically, procedural techniques have served as a way to generate visually complex content while circumventing memory limitations. For example, Braben and Bell's 1984 space game *Elite* had 32 KiB to work with on the BBC Micro computer, and only around 22 KiB after taking into account screen graphics [Bra11]. Due to this, they represented the 8 galaxies, each containing 256 planets, procedurally. All of the information constituting the universe was extracted from a set of three 16-bit seeds [Mox]. Similarly, Toy, Wichman and Arnold's 1980 dungeoncrawler *Rogue* provided an endlessly replayable experience through procedurally generated levels, items and monsters [Cra21]. The demoscene [Tas04; Mol12] is also a great example of procedural techniques used to overcome technological limitations. Demos started as introductions to cracked video games distributed through snail mail in the 1980s, contemporaneous with the introduction of the home computer. They evolved to be more and more complex as a proof of the crackers' skills, and later fully detached from video games. Demos became standalone programs around which a whole subculture was created, especially in northern and western Europe. They showcased advanced real-time graphics animations and effects set to music, to create a production that looked impossible for the hardware they were executed on. The two main challenges are the real-time constraint and hardware memory limitations. This means that demos have to run at thirty to sixty frames per second, all within a tight memory budget. Some categories even enforce a maximum executable size, typically 64 KiB or 4 Kib. This lead demosceners to heavily use procedural techniques to generate visual effects and geometry, since barely any data can be stored explicitly in memory.

As [Sme+14] states, the two main advantages of procedural techniques are *data amplification* and *data compression*. This means that procedural modeling can be used to represent large amounts of model from a single compact system with few rules, and conversely that complex geometries and signals can be encoded with relatively low amounts of data. These advantages are still relevant today, since the demand for increasingly complex content for computer graphics applications is steadily rising, and tessellation, geometry and mesh shaders allow for on-the-fly geometry creation on the Graphics Processing Unit (GPU).

2.3.1 Classification of procedural techniques

Procedural techniques come in a variety of flavors, each with their own advantages and disadvantages. They are often separated in three main categories: *procedural noise*, *example-based synthesis* and *tile-based synthesis*. I will particularly focus on tile-based approaches since they are more directly related to layout problems.

2.3.1.1 Procedural noise functions

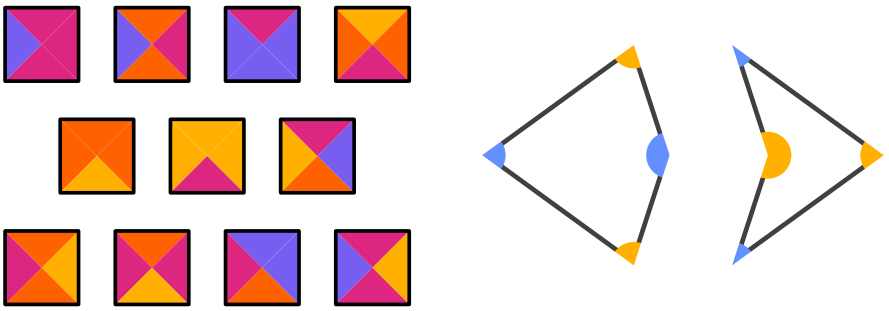
Noise can be informally defined as a random unstructured pattern characterized by its frequency contents. Procedural noise functions are described by program code instead of being represented by data. This usually provides a memory efficient, multi-resolution method that allows efficiently adding visual detail to images, either through textures or geometry. [Lag+10] provides a formal definition and a classification of procedural noise functions. There are two main approaches to compute procedural noise functions. Lattice noises consist of random values and possibly gradients over a discrete lattice that are interpolated to obtain a continuous noise, e.g. Perlin noise [Per85]. The other approach is sparse convolution noise, which consist of a sum of randomly positioned and weighted kernels, e.g. Gabor noise [Lag+09]. [Ebe03] provides an in-depth exploration of procedural texturing and modeling, complemented with design methods specific to these techniques. Due to the stochastic nature of procedural noise functions, this approach struggles to generate highly structured data.

2.3.1.2 Example-based synthesis

Example-based synthesis [Wei+09; Lef14] aims to create arbitrarily large amounts of content that replicate the characteristic features of a set of provided exemplars. This allows to replicate data such as textures that are hard or expensive to obtain, such as extending a small texture obtained from a photograph. The main idea is to create an output that locally looks the same as the input. For textures, this means recreating pixel neighborhoods in the output based on those in the input. If the input is local and stationary, i.e. every pixel is characterized by its neighborhood and the characterization is consistent over all the input, the synthesis preserves perceptual quality. For example, [MWT11; TWZ22] generate a structured output consisting of different objects with spatial distributions similar to those in provided exemplars. In some sense, they use a small solution to an object packing problem to generate a packing in a larger space. Breaking any of the two aforementioned hypotheses requires additional work to produce good results. For example, non-stationary (or spatially varying)

texture synthesis requires control maps identifying similar regions of the input to produce coherent results. Again, this leads to issues when handling highly structured data.

2.3.1.3 Tile-based synthesis



(a) Minimal set of eleven four-coloured Wang tiles described in [JR21].

(b) Penrose's kite and dart tiles. Vertex colors have must match to assemble multiple tiles.

Figure 2.14: Two types of aperiodic tile sets.

Tile-based synthesis [Lag07] encodes complex signals by generating them over a small set of tiles that aperiodically tessellate the space. A tiling is periodic if there exists a translation that preserves it, or aperiodic if none exist. A tile set is aperiodic if it cannot produce a periodic tiling. Aperiodicity is a desired property for content generation, since it avoids visual artifacts due to structured repetition. Additionally, this requires the tile contents to seamlessly match across tile boundaries. The classic examples of aperiodic tile sets in computer graphics are Wang tiles and corner tiles. Wang tiles were proposed in 1961 [Wan61] and popularized in 1965 [Wan65] by Hao Wang. They are sets of square tiles where each edge has a specific color. Two tiles can be adjacent to each other in a specific direction only if their corresponding edge colors match. The smallest set of aperiodic Wang tiles consists of eleven tiles with four edge colors, initially found in 2015 [JR21] (see Figure 2.14a). Also, they proved that this set is minimal, in the sense that any set with fewer than eleven tiles of four colors cannot be aperiodic.

Regular Wang tiles do not constrain their neighbors diagonally, in particular any two tiles can be placed diagonally from each other by carefully choosing the two remaining tiles that complete the square. This problem was first identified in [Coh+03] and can possibly lead to visual artifacts. Corner tiles, with colors on the corners instead of the edges, were introduced [LD06] to solve this problem. Wang tiles were introduced to computer graphics in [Sta97] for texture synthesis, and have been used for many applications since, such as surface modeling, non-photorealistic rendering or landscape modeling [Lag+08]. Another type of aperiodic tiling are Penrose tilings, discovered in 1974 [Pen79]. The most famous one consists of a kite and a dart, shown in Figure 2.14b. These have to be assembled in a specific way to form aperiodic tilings, represented by patterns over the tiles in the Figure. Maybe the newly discovered hat and spectre tiles (see Section 2.1.2) will also be used for computer graphics applications in the future.

Tile-based methods encode a topological information through edge, corner colors, or adjacency patterns. In computer graphics, these topological constraints are rarely exploited, since the most relevant property is aperiodicity, which prevents visual artifacts from appearing. [ZJL14] and [BWL18] present tile-based methods for (respectively) 1D and 2D pattern generation with topology control. This allows generating patterns with a specified number of connected components or holes by tracking this information with topology descriptors over tiles. These contain identifiers over the cell boundaries that correspond to connected components within the tiles, and can be combined when multiple tiles are assembled to obtain a global description of the synthesised result.

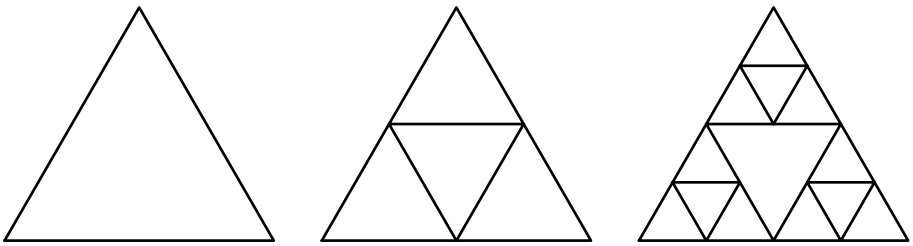
2.3.2 Structure-aware design

Most of the procedural techniques described previously have at best a tenuous grasp on the structure of the generated content. This is not particularly surprising, as the notion of structure makes sense intuitively to a person but is particularly challenging to formalize. Indeed, structure is a key part of the design of synthetic objects, in the sense that their form and shape is informed by their purpose, be it aesthetic or functional. In their presentation of structure-aware shape processing, [Mit+14] states that "*shape structure is about the arrangement and relations between shape parts*", parts being entities having semantic significance influencing their geometry. Relations can be of different natures, including pairwise geometric constraints (e.g. parallelism, coplanarity), higher-order relations such as symmetry, or functional relations (e.g. what properties of an arrangement of parts makes it function as a chair).

In the rest of this section I consider rewriting systems and model synthesis as examples of structure-aware design. Rewriting systems in particular provide an illustration of the synthesis process in structure-aware shape processing, while model synthesis uses adjacency constraints and neighborhood information to generate outputs with local structures originating from an example. Finally, I end the section with a short consideration of inverse procedural modeling techniques that try to provide solutions to the challenge of creating procedural systems that generate the desired type of output. Of course, this is not an exhaustive inventory of techniques in structure-aware design, but rather an introduction to how the notion of structure can be captured through clever models and algorithms.

2.3.2.1 Rewriting systems

While tile-based approaches and some others presented previously are local in nature and have a topological component to them, rewriting systems fully take advantage of locality. Rewriting systems in their most general form consist of objects and rules specifying how they transform. For an in-depth formal approach to rewriting systems, please refer to [Ter03]. Two quintessential examples of rewriting systems are phrase structure grammars and L-systems. Phrase structure grammars were introduced by Chomsky in 1956 [Cho56] as a model for the description of language. It consists of an alphabet of symbols and a set of production rules each transforming an input string of symbols into an output string. The rules can be applied sequentially to an initial symbol, and all words obtained in this manner constitutes the language of the



(0) Axiom: `F-G-G` (1) `F-G+F+G-F-GG-GG` (2) `F-G+F+G-F-GG+F-G+F+G-F+GG-F-G+F+G-F-GGGG-GGGG`

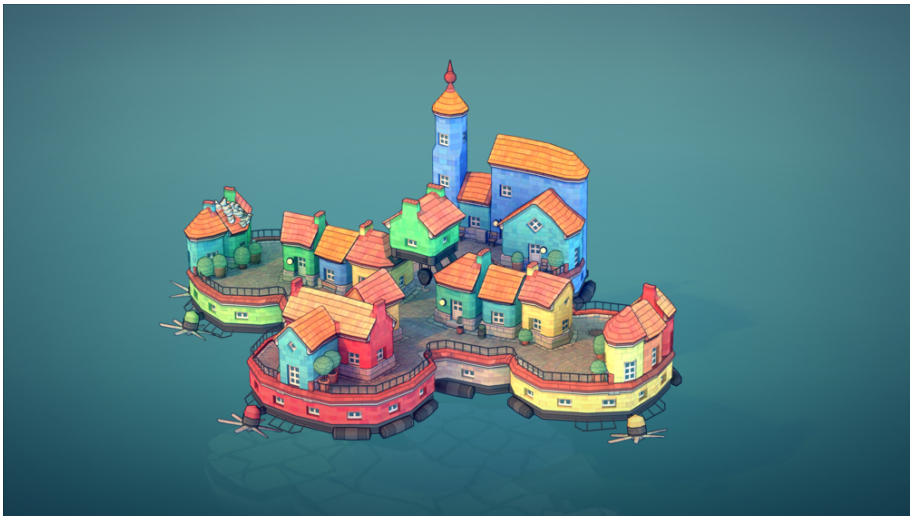
Figure 2.15: First three rewriting steps of an L-system generating the Sierpiński triangle. At every step, `F` is replaced with `F-G+F+G-F` and `G` with `GG`. The drawing is generated with Turtle graphics, interpreting `F/G` as moving forward, `-/+` as turning by $\mp 120°$ clockwise.

grammar. L-systems were introduced by Lindenmayer in 1968 [Lin68a; Lin68b] to model the growth of filamentous organisms containing linear and branching structures. They were subsequently developed in the following decades by Prusinkiewicz [PL90] and others as a tool for the computational modeling of plant structure and growth. The main differentce between L-systems and phrase structure grammars is the fact that the former is a parallel rewriting system, i.e. all rules that can be applied at a time are applied simultaneously. Strings of symbols obtained with an L-system can be interpreted graphically to generate space-filling curves or plant models (see Figure 2.15). The simplest L-systems are deterministic and context-free (D0L-systems), i.e. production rules transform a single symbol into a string of symbols, independently of the surroundings of the input symbol. Many other variants exists, such as context-sensitive (rules look at the surroundings of the input string), stochastic (multiple applicable rules are chosen according to a probability) or parametric (values are attached to the symbols). L-systems and their variants have been used in computer graphics for fractal generation [Pru86], space-filling curves [PLF90] or smooth subdivision curves [Pru+03] and surfaces [SPS04; KMB06]. Prusinkiewicz attributes the success of L-systems for plant modeling and subdivision algorithms to "*the ease of expressing geometric algorithms that operate locally on structures with a varying number of components. This ease is achieved through index-free notation that emphasizes the topological relations between components*" ([PSS10]). There are also rewriting systems operating over more complicated objects. These include arrays [Kir64; Dac70; Ros87], graphs [Ros72; Pfa72], general 2D shapes via shape grammars [SG71; Sti75] or 2D and 3D shapes via split grammars [Won+03]. They are usually described as picture languages in the 2D case, and are used for pattern recognition, image processing, or artistic and design purposes. L-systems, graph, shape and split grammars have also found applications in building, facade and floorplan modeling [Sme+14], and even robot design optimization [Zha+20].

2.3.2.2 Model synthesis

This section focuses on Model Synthesis (MS) [Mer07] and WFC [Gum16], two similar example-based procedural synthesis algorithms. These garnered attention through their use in the video games *Bad North*, published in 2018 by Plausible Concept and later *Townscaper*, published in 2021 by Oskar Stålberg,

(a) Island example from Bad North.



(b) City example from Townscaper.

Figure 2.16: Examples of models generated with Wave Function Collapse (WFC) illustrated in promotional images, respectively from the Bad North website, Copyright 2018 Plausible Concept; and the Townscaper Steam page, Copyright 2020 Oskar Stålberg.

a member of Plausible Concept. The use of WFC is well-documented by Oskar Stålberg online, through conferences, technical posts and interviews [AI 22]. In Bad North, the islands on which the main gameplay happens are generated procedurally on a 3D square grid, with different features such as beaches, hills, cliffs, houses and trees.

MS focuses on 3D model generation, while WFC focuses on 2D texture synthesis, but both rely on constraint satisfaction and are closely related. This approach is the basis for the procedural support generation for 3D printing detailed in Chapter 4. The main idea behind MS and WFC is to consider a discrete 2D or 3D space consisting of cells and a set of labels. These labels can

be assembled according to a set of adjacency rules, determining which labels can be adjacent to others in a specific direction. At any point of the algorithm, each cell contains a list of possible labels it can be assigned. At every iteration, a cell is selected and a label is assigned to it. Then a constraint propagation step removes labels rendered impossible in other cells due to the adjacency rules. The algorithm continues until every cell has been assigned a label. Adjacency rules are typically extracted from a segmented exemplar, where every possible combination of adjacent segments is added to the list of allowed adjacencies. These algorithms fail if at any point some cell has no allowed labels. In that case, the full process can be restarted from scratch, a region of the output can be regenerated, or backtracking can be used to undo previous choices of labels. If the algorithm succeeds, visually interesting results can be obtained by associating colors or geometry to the labels.

These algorithms combine tile-based and example-based approaches, and incorporate topological information through adjacency constraints. They can be customized in a number of ways. First, different types of information can be extracted from the exemplar. MS only extracts adjacency information along the main axes of the grid, while WFC extracts all possible $N \times N$ neighborhoods and their number of occurrences. Next, different heuristics can be used to choose the next cell to assign a label to. MS proceeds in a scanline order, while WFC chooses the cell that accepts the fewer number of possible labels. Additionally, once a cell is selected, the label can be chosen in a number of ways. Finally, MS starts from a trivial solution and works by blocks: once a block is chosen, block cells are set to an undetermined state and new content is synthesized within. In contrast WFC operates on the whole grid from the start. The specific differences between MS and WFC are summarized in [Mer21]. Merrell later extended MS to work on shapes not defined on a grid [MM08] and to be able to specify simple constraints on the output [MM09]. Recently, [Mer23] provided a method to generate polygonal shapes by generating a graph grammar (i.e. rewriting systems on graphs) from an example.

Designing an exemplar that when given to WFC produces an output with the desired visual, geometric, structural or topological properties is a significant challenge. It often heavily relies on trial and error, and predicting the effect on the output of a modification of the input is particularly difficult.

2.3.2.3 Designing procedural models

Procedural modeling essentially requires to construct an abstract set of rules that, when interpreted suitably, generates the desired content. One of the main issues with this approach is the lack of controllability during this process, and the difficulty of expressing intent [Sme+14]. Indeed, it is very hard to predict what effect a change in the rules or the parameter values will have on the final result. Inverse Procedural Modeling (IPM) tries to accomplish the opposite, i.e. create a procedural model that can reproduce the provided input data [Ali+16]. This can be done at different levels. For example it is possible to try to reproduce the data with a fixed model, finding the best set of parameters to match the output of the model to the input data. Some approaches try to fully construct a procedural model instead of just finding parameters for a preexisting one. By starting with traditionally created content that expresses the artists' or designers' intent and using IPM to generate a procedural model

to produce similar outputs, it is possible to circumvent the design problem of procedural models. At that point all virtues of procedural modeling can be exploited, creating a range of data replicating the initial input by changing parameters or production rules.

[BWS10; Bok+12; Kal+12] define and use notions of similarity between shapes and symmetry under transformations within a shape. Two regions from two different shapes are similar if they are geometrically matched and locally topologically equivalent. Two regions within a shape are symmetric under a transformation if the transformation maps the first onto the second while preserving the local topology. The first notion allows to formalize the notion of the output of a procedural method locally resembling the example used to generate it. Indeed, requiring that every neighborhood of the output is present in the output, as is done in WFC and [Mer23], is encoded by this notion of similarity. Symmetry helps define regions that can support similar shape operations that modify the geometry within, which is closely related to rewriting systems.

Still IPM presents a significant challenge today. In particular, it is hard to apply procedural methods to design for computational fabrication, since control and low-overhead is essential for these applications, even though the field might benefit from it in the future [BFR17].

Summary

Generative systems, in particular tile-based approaches, rewriting systems, and model synthesis are closely related to layout problems with topological constraints. In one way or another, a space is filled with tiles, labels or symbols determined by topological constraints or rewriting rules. These encode information about the structure of all possible generated results. In any case, layouts generated according to these rules or constraints are guaranteed to be valid by construction. Thus there is no need for a verification step, which is often hard to do as exemplified in Section 2.2. The hard part remains the design of the generative system itself, since predicting changes in the result based on changes on the system is far from trivial.

Chapter 3

3D LED-based displays via foldable circuit boards

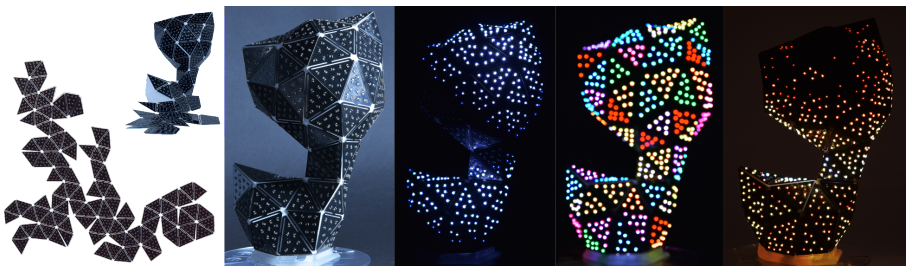


Figure 3.1: Starting from a 3D mesh our method automatically generates design files to produce an on-surface display composed of individually addressable Red-Green-Blue (RGB) Light-Emitting Diodes (LEDs). The circuit board is manufactured through standard Printed Circuit Board (PCB) production services, including component soldering. The user then folds the fabricated board back onto a 3D printed support. The final model becomes a curved display, onto which intricate light patterns can be programmed in a shader-like manner.

This chapter proposes a computational design approach for covering a surface with individually addressable RGB LED, effectively forming a low-resolution surface screen. To achieve a low-cost and scalable approach, we propose creating designs from flat PCB panels bent in-place along the surface of a 3D printed core. Working with standard rigid PCBs enables the use of established PCB manufacturing services, allowing the fabrication of designs with several hundred LEDs. Our approach optimizes the PCB geometry for folding, and then jointly optimizes the LED packing, circuit and routing, solving a challenging layout problem under strict manufacturing requirements. Unlike paper, PCBs cannot bend beyond a certain point without breaking. Therefore, we introduce parametric cut patterns acting as hinges, designed to allow bending while remaining compact. To tackle the joint optimization of placement, circuit and routing, we propose a specialized algorithm that splits the global problem into one subproblem per triangle, which is then individually solved. Our technique generates PCB blueprints in a completely automated way. After being fabricated by a PCB manufacturing service, the boards are bent and glued by the user onto the 3D printed support. We demonstrate our technique on a range of physical models and virtual examples, creating intricate surface light patterns from hundreds of LEDs.

This work is the result of a two-year-long collaboration between Camille Schreck, Pierre-Alexandre Hugron, my advisor Sylvain Lefebvre and myself, with Manas Bhargava and his advisor Bernd Bickel from the Institute of Science and Technology Austria. It resulted in a publication [Fre+23] in *ACM Transactions of Graphics* with Manas and I as joint first authors. As stated in Section 1.4, Manas played a significant role on mesh unfolding, bending experiments and LED placement, while I specifically focused on hinge design, LED routing and layout generation. The code and data resulting from this work are available at `https://github.com/mfremer/pcbend`. The submission video showcasing multiple lighting effects is available at `https://youtu.be/g8UX-KifGmM`. These effects are impossible to display in a static format, and even the video does not fully do them justice. I would highly encourage the reader to watch it if possible, and this chapter will make reference to it multiple times. Finally, the submission data is available at `https://mybox.inria.fr/d/cabc196c89704ec090e6/`, it contains the SVG files and Gerber files used to fabricate all of the objects showcased in this chapter.

Manas and I gave a presentation on the article at the SIGGRAPH 2023 conference in Los Angeles, California. We also presented our fabricated objects at the *Bring Your Bunny (or something)* fabrication meet-up there. This was a great opportunity to showcase the objects themselves and the lighting effects to the computational fabrication and computer graphics communities, enjoy the objects brought by everybody and exchange about our respective works.

This project has also been mentioned in other media: an article on *Inria*'s website (english, french), a mention in an article (in french) by *L'Usine Nouvelle* (weekly French business magazine), and an article in *La Semaine* (weekly regional French journal).

In total, we fabricated the following objects with two different types of LED a smaller 1.5×1.5 mm^2 one (`1515`) and a larger 5.0×5.0 mm^2 one (`5050`). The objects were fabricated in two batches, the first at a lower density with an older version of our system, and the second one at the maximum density currently possible with our system (see Figure 3.23 for a comparison).

- two *icosa* (86 and 201 `1515` LEDs);
- three *cat* (979 and 1881 `1515` LEDs, 460 `5050` LEDs);
- two full *sqtorus* (4×678 and 4×2011 `1515` LEDs);
- two *star* (166 and 447 `1515` LEDs);
- one *batman* (289 `5050` LEDs);
- one *dome* (458 `5050` LEDs).

3.1 Introduction

Light installations are ubiquitous in modern homes and cities. They decorate rooms, streets, shops and hotels, and can be art pieces by themselves. We use light everyday and everywhere not only as a commodity, but also to impact mood and productivity, and to highlight a space, an art piece, or even entire buildings. In this work we explore how to design objects covered with hundreds of individually addressable lighting elements. The obtained luminaires can combine shape and colored light in novel and intricate manners, producing on-surface animated light patterns, effectively acting as free-form displays. We rely on bright and colorful addressable RGB LEDs such as the *Adafruit NeoPixels* (WS2812B). We refer to these as *LED pixels*. While LED pixels are very popular amongst hobbyists and designers, their use on free-form layouts has been limited: circuit design within complex geometric outlines and manual soldering quickly becomes impractical with hundreds of components (see Figure 1.3). While promising methods are being explored to prototype circuits along curved surfaces (see Section 3.2) they suffer from similar scalability issues.

To match our vision of creating on-surface displays with hundreds of LEDs, we set out the following requirements for our system:

- **Simplicity of use and design automation:** The only required user input has to be the target surface mesh. Our approach should then automatically generate a fabrication-ready design, in a reasonable amount of time, without requiring any modelling assistance.
- **LED coverage:** We want the LEDs to cover the surface as densely as possible, with the option for the user to reduce the coverage density if so desired.

- **Fabrication scalability:** Since LED pixels come in packages as small as 1.5 by 1.5 mm, we are considering the production of objects supporting several hundreds of them. The fabrication process has to be reliable and automated, avoiding the long and tedious task of hand soldering and tracking potential issues across hundreds of components.
- **Availability and cost:** Fabrication methods for the results should be readily available to users, requiring no specialized tooling or equipment on their part, with designs fabricable at a reasonable cost.
- **Lighting effect design:** It should be easy to program and experiment with lighting effects onto the target surfaces.

With the above requirements, our system aims to enable the design of large on-surface displays that can be easily manufactured and controlled.

Limitation. Our system expects the input mesh to be a surface with appropriately scaled triangles of reasonable quality and size.

3.1.1 Design principles

Our set of requirements led us to the following choices in the design of our system, which is illustrated in Figure 3.2.

Fabrication. To ensure fabrication *scalability* and *availability*, we target traditional (non-flexible) PCB. Such PCBs can be fabricated at a relatively low cost from several online services – e.g. *PCBWay, JLCPCB, Beta Layout, Eurocircuits* – with components automatically soldered by a pick-and-place machine. We thus inherit the reliability, moderate cost and automation of well-established manufacturing processes.

PCBs along surfaces. Instead of producing many flat PCBs and connecting them all — a prohibitively tedious task —, or using rigid-flex/flex PCBs — more expensive and harder to design — to wrap around a surface, we propose to *bend* a rigid PCB into shape along the target surface. This effectively forms a "skin" atop a 3D printed support as shown in Figures 3.1 and 3.2. To this end, we design specialized *kerfing* patterns [Kal20; Lee+18] that we call *hinges*. This captures the idea that the deformation is concentrated on these areas, while the rest of the PCB remains rigid. These allow the PCB to bend along specific cut patterns, while enabling required electric signals to go through. We seek to use small hinges to leave as much space as possible for LEDs. We thus study different hinge geometries and their admissible folding angles given their design parameters. This setup naturally leads to an unfolding problem: the input mesh surface is cut and flattened into the plane to obtain the board geometry. However, an important difference when compared to unfolding for paper or cardboard is that the PCB hinges bend with a limited radius of curvature. This requires trimming and offsetting the triangles to make space for hinges in the unfolding.

Circuit design. Our unfolder generates the PCB geometry from a 3D mesh. Next, we densely cover the board with LEDs connected by a valid circuit satisfying all manufacturing constraints. We rely on RGB LEDs that can be

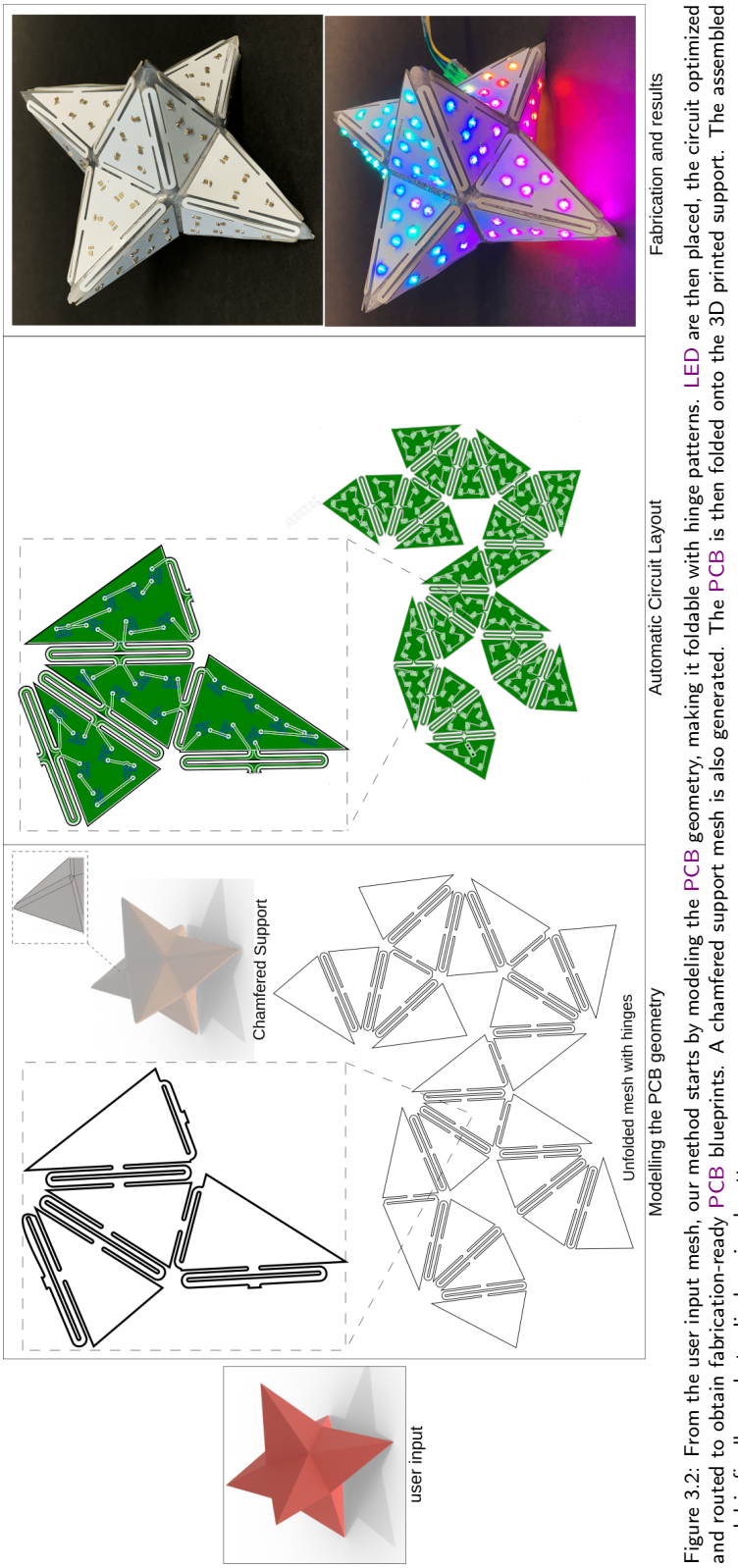


Figure 3.2: From the user input mesh, our method starts by modeling the PCB geometry, making it foldable with hinge patterns. LED are then placed, the circuit optimized and routed to obtain fabrication-ready PCB blueprints. A chamfered support mesh is also generated. The PCB is then folded onto the 3D printed support. The assembled model is finally ready to display visual patterns.

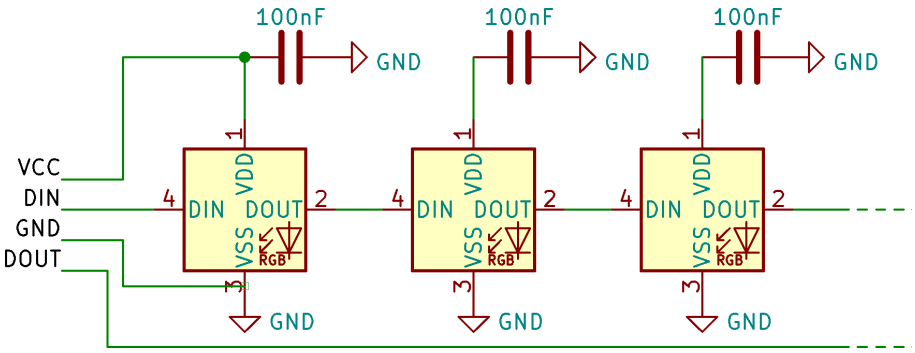


Figure 3.3: Schematics of a chain of LED pixels, Each LED is connected to gnd and vcc, with a decoupling capacitor next to it. LEDs are chained through their `data-in`, `data-out` pins, passing information along the chain. The connector `DOUT` pin on the left can be used to connect multiple chains together.

chained in a sequence while being individually addressable, connecting each `data-out` pin of a previous LED to the `data-in` pin of the next, see Figure 3.3. The chain topology of the circuit and the PCB geometry introduce a strong interdependence between circuit placement and routing.

Indeed, considering the data pins only, forming a non-intersecting chain through the many LEDs and hinges amounts to computing a Hamiltonian path over the entire design. This path is not abstract: it has to go around components and has to satisfy geometric constraints from PCB manufacturing (e.g. minimal width of 0.2 mm, minimal clearance of 0.2 to 0.25 mm). This makes optimizing for routing globally intractable, with the likelihood of failure rapidly increasing with the number of placed LEDs. In fact, there is no guarantee a valid routing is even possible given a specific placement, as stated in Section 2.2.3. To tackle this challenge we propose a specialized placer and router that exploits the structure of the circuit as well as the structure of the unfolded PCB. Our algorithm splits the global place-and-route problem into local per-triangle problems that can be efficiently solved, in particular exploiting the freedom of ordering the LEDs along the chain.

Interactivity. We facilitate the exploration of interesting light patterns by proposing a shader-like programming interface. It allows to quickly implement different patterns and visualize them interactively on the 3D-display, through a live-coding interface.

3.1.2 Using our system

The user starts designing a display from only a target surface mesh, the type of LED to use and (optionally) a desired spacing between the LEDs which controls the packing density. From the user's perspective the modeling process is entirely automated. Our approach optimizes the PCB geometry, LED placement, circuit schematics, layout, and routing automatically. A set of fabrication-ready PCB blueprints and a bill of materials (component listing) is output, as well as an object to be 3D printed. The PCB is sent for fabrication to an online service, while the 3D object is either printed through an online service or in-house. Once folded and glued onto the surface, the PCB provides a dense coverage

of individually addressable LED pixels. The LEDs are then driven from an external microcontroller connected to the PCB. To demonstrate the feasibility of our approach, we fabricate several designs and demonstrate the variety of lighting effects that can be produced through a shadertoy-inspired live coding interface.

3.1.3 Outline

This chapter is structured as follows. First, Section 3.2 introduces related work in the fields of 3D electronics, surface unfolding and shape packing. Next, Section 3.3 deals with the general problem of modeling the flat shape of the PCB, including unfolding the initial shape, hinge design and hinge insertion. With the PCB shape computed, Section 3.4 explains the principles behind our automatic circuit layout generator, producing a functioning circuit within the shape by splitting the global problem into per-triangle local ones. Then, Section 3.5 showcases our results, from fabricated objects to virtual examples, all illustrating a variety of lighting effects. Finally, Section 3.6 concludes the chapter with a discussion on our approach. Appendix 3.A provides a few more virtual examples with the maximal density possible with our system.

3.2 Related work

This section follows the introduction to layout problems in electronics design automation from Section 2.2. It quickly summarizes the main points relevant to PCB design and then dives into different approaches to 3D circuitry, namely direct on-surface circuit fabrication and circuit deformation from planar to 3D. The section concludes with some useful background on surface unfolding and shape packing, relevant to the understanding of our approach.

Electronics design typically starts from a circuit schematic after which a PCB is modeled. Adequate components are chosen and soldered onto the board to realize the circuit. Typical PCBs are copper-clad glass-reinforced epoxy laminates. Electrical interconnects are etched into their surface to create conductive traces and attachment pads for soldering [LS20]. *Vias* are inserted to connect traces across layers. PCB manufacturing is a mature technology, ubiquitously used across many industries. Components range from millimetric Surface-Mount Devices (SMDs) to larger components with pins extending out to facilitate manual soldering and prototyping. Today, online services allow anyone to manufacture industrial-grade multi-layer PCBs with pre-soldered SMDs. However, the difficulty is shifted to the PCB modeling process, as shown in Section 2.2.4. This is a task under strict geometric and electrical requirements, and professional tools [CK92; Cad88; Alt05] require training and expertise. Therefore, efforts are devoted to exploring simpler prototyping and modeling tools, often in conjunction with novel design capabilities. In particular, researchers explore ways to move beyond the planar nature of PCBs and design curved circuits. We discuss the approaches most related to our work next. For an exhaustive overview of this topic we refer the readers to recent surveys, e.g.[Wu+20; Ric+21].

3.2.1 3D and on-surface circuit fabrication

Direct 3D fabrication of circuits is an overarching goal of modern electronics design. The technologies being developed are focused around on-surface deposition of conductive materials [MID92; US17; Tor+13] and additive manufacturing techniques [Swa+19; Flo+17; He+21; Zhu+20a]. A variety of materials are used such as carbon, graphene or copper enriched filaments [Flo+17] and silicon [Zhu+20a]. These techniques present several challenges for our purpose. Depositing along existing surfaces is limited to specific shapes due to reachability constraints, in particular in concave regions. When layered the deposited conductive traces present a sharp increase in resistance across layer interfaces [HMB21], limiting usability to low-current and sensor applications. Finally, many of these conductive materials cannot be soldered onto. Thus only through-hole or large SMD components can be glued, unless chemical post-processes are applied such as copper-plating [Ang+18; Kim+19].

3.2.2 Deforming circuitry: planar to 3D

A different direction of research aims to fabricate the circuits in a planar fashion and deform them into free-form shapes. Our work belongs to this category. The deformation can be created in different ways, e.g. thermoforming [Plo+17; HMB21], self-morphing under heat [Wan+20], wearable textile substrates [Par+21] or manual folding [Olb+15]. Other techniques produce planar transfer stamps [Hod+14; GS18; Zhu+20b] to help a user place conductive traces on a target surface. Rigid-flexible PCBs can also be used to produce curved designs [Alt14]. For instance, Muscolo et al. [MMC19] connect ten triangular circuit elements through flexible areas to produce a curved sensor. Such PCBs are however significantly more expensive than regular PCBs [Wan+20; All23] and impose stricter manufacturing constraints. The extreme flexibility would make assembly very difficult too when working with large designs. Note that while the initial configuration is planar, the final circuit topology can be more complex as several sheets can be glued together [Yam+19] with traces connected across [Olb+15]. Circuits may contain several conductive layers with connections across isolating layers [HMB21; Yan+22]. The planar-to-3D line of research is especially active regarding the interactive design of objects augmented with electronics. Most 2D fabrication techniques are accessible to individual users, for instance, drawing or inkjet printing with conductive inks [Rus+11; Kaw+13; JSC15; Wan+18] or laser cutting of specially prepared laminated copper–kapton sheets [Yan+22]. This led to the development of interactive tools assisting users in creating foldable designs with sensors, actuators and display elements [QB10; Olb+15; Oh+18; Par+21]. Curvature can also be facilitated by introducing spatially varying kerfing patterns [GS19]. Kerfing is traditionally used in woodworking to attain a wide range of shapes through different cut patterns [Rod+24; SK24]. Interactive techniques are designed with the user in the loop, exploring means of fabrication allowing quick iterations of do-it-yourself (DIY) prototypes. Therefore the tools do not need to be fully automated when it comes to placement and routing, and the employed techniques require manual intervention. In particular, DIY materials such as paper or 3D printed filaments do not allow soldering, making attaching components a delicate manual process. Overall these techniques do not scale

easily to circuits with hundreds of components in terms of manual labor.

Curved displays. Among the aforementioned approaches, a number of techniques prototype displays as applications. The approach of Torres et al. [Tor+17] assists users in optimizing luminaires producing specific light diffusion patterns. The luminaires contain in the order of ten RGB LEDs. Placement and routing is performed by the user with assistance from the system to trace routes. Olberding et al. [OWS14], Lee et al. [Lee+20] and Hanton et al. [Han+20] fabricate display surfaces using thin-film electroluminescence. These techniques produce impressive segmented displays, with large and homogeneously lit surfaces. They however do not provide the flexibility in colors and brightness that RGB pixels are capable of. Prior works demonstrate LED arrays going from tens of pixels [OWS14; Plo+16; Plo+17] to a grid of 25×16 standard LEDs manually glued on paper [Rus+11]. Such array arrangements however require a dense routing pattern that would be extremely challenging along arbitrary surfaces and layouts, both geometrically and for fabrication. In conclusion, none of the aforementioned techniques can address the generation and fabrication of dense LED pixel arrangements along surfaces at the scales we envision. While designing PCBs can be intimidating, our approach fully automates the process.

3.2.3 Background on surface unfolding

As our method relies on unfolding we provide some background in this section. Edge-unfolding or simply unfolding is a process of cutting a 3D model represented as a polyhedral surface along its edges and flatten the surface onto the plane without introducing any distortion or overlaps between faces [Kon03]. Origami and kirigami, traditional Japanese art forms of folding and cutting paper, have been extensively studied [CZ18] and are strongly related to unfolding. Obtaining a non-overlapping unfolding might require cutting the surface into multiple disconnected *patches*. Finding an unfolding with a minimal number of disconnected patches is computationally hard [She75; DO07]. Not only that, there exist ununfoldable non-convex polyhedra and it is an open problem whether every convex polyhedron can be unfolded into a single non-ovelapping patch [DO07]. Thus, algorithms typically rely on heuristics. Straub and Prautzsch [SP11] use a minimum perimeter heuristic [Sch97]. This results in the unfolding which has the minimum perimeter but does not guarantee that there are no intersections. This is resolved in a post-process, introducing further cut edges by solving a minimum-set cover problem (Section 2.2 in [SP11]). This approach is however known to still create several patches on complex cases. Several improvements have been explored, using for instance genetic algorithms [Tak+11] or simulated annealing [Kor+20]. Most prior works focus on folding paper and cardboard along crease lines. This does not directly apply to our case due to the bending limitations of the PCB but provides a strong foundation on which our algorithm is based.

3.2.4 Background on shape packing

Distributing components on the surface is an instance of shape packing inside a bounded domain, with additional strict constraints: non-overlapping components and circuit routability. For more context on packing problems, please

refer to Section 2.1.2. A general approach of tiling shapes in a bounded 2D domain is a hard problem [FPT81; El-+09] leading most algorithms to resort to heuristics. A typical approach is to use a Centroidal Voronoi Tessellation (CVT) to distribute points and spawn tiles of various shapes in a bounded domain, as in for instance [Hau01; SLK05; Dal+06]. These works however do not guarantee a non-overlapping result. Others explore the layout of tiles directly on a surface, for filigrees [Che+16], mosaics [Hu+16], volumetric elements [Fan+22] and fabricable tilings [Che+17]. However, tiles are deformed to ensure that they fit compactly, which cannot be done with components. Xu et al. [Xu+20] use a self-supervised network to solve a tiling task. They optimize for maximal coverage, resulting in stacked layouts of tiles. This however does not allow for uniform distributions. To achieve a dense packing in our context we propose a specialized packer that runs in an optimization loop alongside the circuit router.

3.3 Modeling the PCB geometry

Our method proceeds in two main steps: PCB geometry modeling and circuit synthesis (Figure 3.2). In this section we detail the first part of the process. In Section 3.3.1 we describe our *hinges*: parameterized kerfing patterns that make the PCB foldable, while allowing electrical signals to be routed through. We use different hinges depending on the dihedral angle between connected triangles. We determine their admissible bending angles in Section 3.3.2. Thanks to the hinges, we can tackle the problem as an unfolding task: we seek to produce a flat PCB layout, with hinges in between rigid pieces (triangles) that can be folded onto a 3D printed support. We describe our specialized unfolder computing the PCB outline in Section 3.3.3 and the modeling of the support structure in Section 3.3.5.

3.3.1 Hinge designs

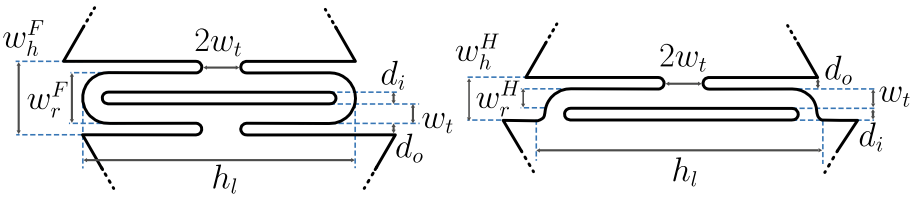


Figure 3.4: Hinge designs and parameters. Left: full hinge, right: half hinge. Track width (w_t) is 1.7 mm, inner diameter (d_i) and outer diameter (d_o) are 1.0 mm each. The rigid width is w_r^F = 4.4 mm for full hinges and w_r^H = 1.7 mm for half hinges. The total hinge width is w_h^F = 6.4 mm for full hinges and w_h^H = 3.7 mm for half hinges. The curved sections are circular arcs.

We consider two types of hinges, shown in Figure 3.4. We refer to them as *full hinges* (Figure 3.4, left) and *half hinges* (Figure 3.4, right). Mechanically, both consist of one or two long torsion elements connected by shorter rigid sections (w_r^F for full hinges and w_r^H for half hinges). Torsion is applied by the sections connecting the triangles and the hinge, acting as levers. As the hinges take up space in the final design and reduce the area available for LEDs, all parameters but the hinge lengths h_l are set as small as possible under the manufacturing

constraints defined by the PCB fabrication service (see Figure 3.13). Thus, both our hinge designs are parameterized by just their hinge lengths h_l.

We considered other hinge designs, such as a single repeating zigzag pattern going from one triangle to the other, but ended up settling on this specific one. The key advantages of this design are the limited footprint, easily fit between two triangles without much effort, and the two channels through the hinge. Having two channels allows us to have the chain go forwards and backwards through each hinge, making it easy to embed a loop in the circuit board and consider each triangle independently. Other hinge designs offered too much flexibility in exchange for larger footprints, but most of the time the extra flexibility was not needed, thus leading to wasted space. Even the full hinge design sometimes offers too much flexibility, which lead to the design of the half hinge. Choosing a hinge design is an exercise in compromise. We want hinges to be as flexible as possible to reach the desired bending angles while remaining as small as possible and having two separate ways through the hinge. Our chosen designs satisfy these requirements, but others, maybe even better, might exist. If additional requirements need to be met, the designs must then be adapted.

3.3.2 Hinge bending

We use 0.6 mm thick PCB with FR4 substrate for fabrication. The hinges are fabricated flat on the PCB and are bent during manual assembly. It is therefore important to determine how much a hinge can bend without damage.

To calculate the maximum bending angles, we conducted an experiment where we progressibely bend hinges of different lengths. We observe that during progressive bending a visible yellowing gradually appears on the PCB material before breakage (see Figure 3.5). This yellowing effect serves as an indication of stress accumulation. We observed that damage to traces on hinges only occurs after yellowing.

(a) Full hinge completely bent on itself. (b) Unbent hinge exhibiting yellowing signs.

Figure 3.5: Illustrations of hinge bending.

We define the maximum allowed bending angle θ based on the first occurrence of yellowing while bending the hinges. We experimentally determined the value of θ for varying hinge lengths for both hinge types in the following way. We fabricated both full and half hinges of varying lengths using 0.6 mm thick FR4 with two copper layers. They were slowly bent by hand on a circular dial on

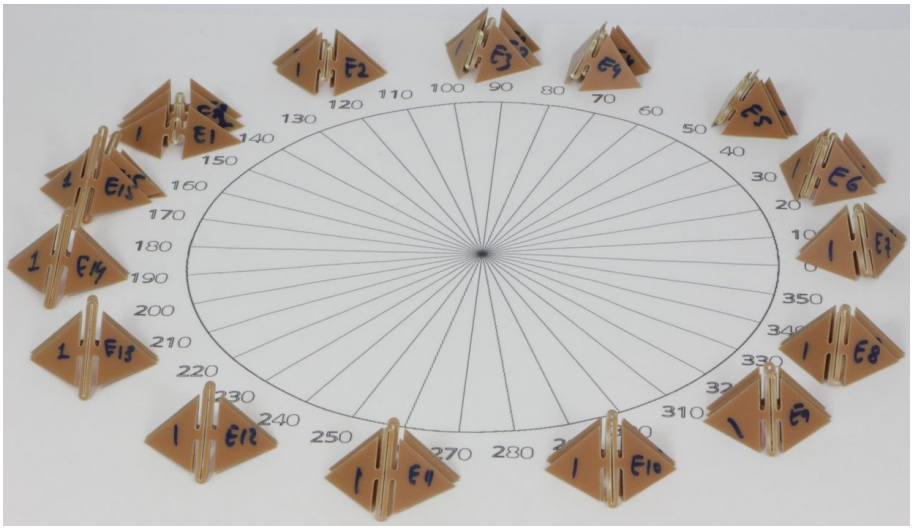


Figure 3.6: Illustration of the bending angle experimental setup.

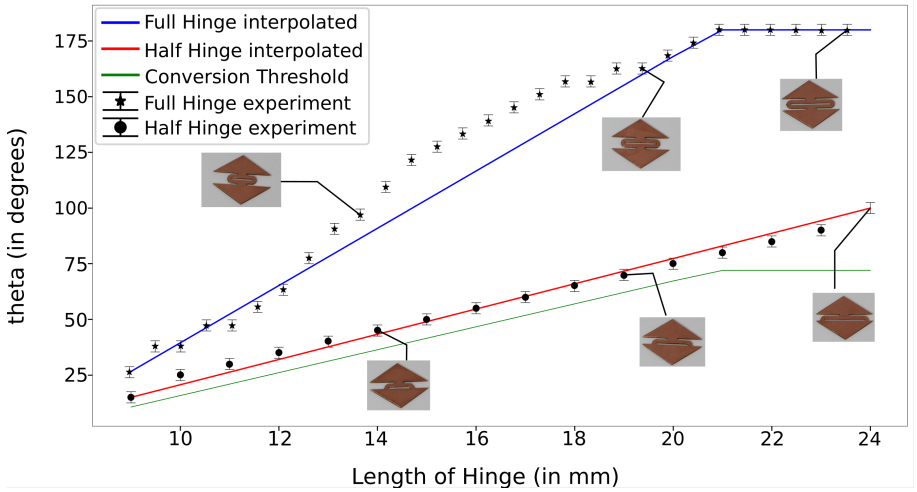


Figure 3.7: Safe bending angle for varying hinge length for both full hinges and half hinges. In green, we see the safety conversion threshold below which we safely replace a full hinge with a half hinge.

which the bending angle was read. We recorded the angle of the first naked-eye visible yellowing for each hinge. A conservative safe bending angle is derived from this value to avoid any yellowing of the hinges. This safe bending angle is further tested by performing a series of fatigue tests. The setup is shown in Figure 3.6. Figure 3.7 shows the resulting maximum angle plot. We observed in the experimental data that the first yellowing angle θ could be simply modeled by an affine function. Additionally this data allowed us to define a conversion threshold allowing us to decide if, given a desired bending angle, a half hinge is enough or a full hinge is needed. Given this experimental data for both full and half hinges, we can consider how to obtain the overall PCB layout.

3.3.3 Unfolding the mesh, folding the PCB

We obtain the PCB geometry by unfolding the input mesh onto the plane. Our unfolder is inspired by the work of Takahashi et al. [Tak+11] which targets paper models. The output of the unfolder is a set of non-intersecting flat patches made of connected triangles. The patches can be folded back onto the initial surface by folding along the edges between the connected triangles. However, unlike paper which bends sharply along a crease line, our hinges curve progressively as the torsion elements twist. As we discuss next, this bending requires redefining the geometry of the triangles in the unfolding as well as modifying the geometry of the support to ensure that the PCB will fold properly in place. We model the bent shape of a hinge as three articulated sections: a middle section of length $w_r^{F|H}$ connected to two sections of length d_o for full hinges and of lengths d_o and d_i for half hinges, see Figure 3.4. In practice, the middle section bends slightly but we found this to be negligible.

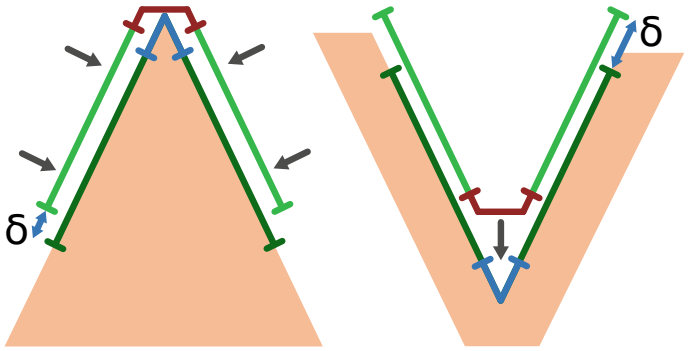


Figure 3.8: Consequences of ignoring the hinge curvature as seen from the side on convex and concave edges. The assumed folding behavior (crease line) is shown in blue, the actual one in red, with the triangles attached to the hinge in green. This moves the triangles away from the opposite edge by an offset δ, introducing a global discrepancy.

Length discrepancies. A standard unfolding provides no space to insert hinges in between triangles. However, as hinges bend they take on a curved shape that has to be considered. Otherwise folding the result back onto the surface will be impossible as illustrated in Figure 3.8. On convex edges the hinge cannot perfectly wrap around the support, pulling the triangles towards the edge. On concave edges, the hinge cannot be flush with the support, pushing the triangles away from the edge. These effects accumulate with every edge as the design is folded, making it impossible to match the support. Note that the mismatch depends only on the dihedral angle between the adjacent triangles when considering a single edge.

Trimming. This problem was already observed in [An+18] in the case of self-folding 3D printed geometries, also bending along hinges with a non-negligible footprint. In their case, bending is due to the release when heating the object of residual stress accumulated during printing. We adopt a similar solution to theirs for hinge insertion. We overcome this problem by trimming the triangles neighboring an edge by an amount that depends on the edge dihedral angle θ

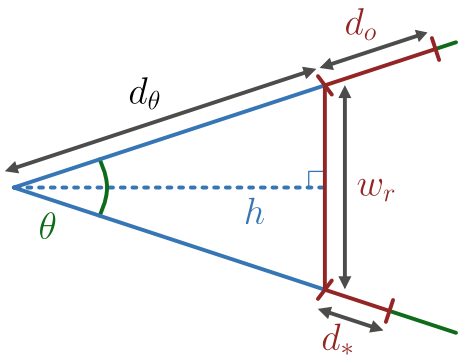


Figure 3.9: Illustration for the dihedral offset computation for both hinges. $d_* = d_o$ for full hinges, $d_* = d_i$ for half hinges.

and the hinge parameters. We compute a *dihedral offset* δ_θ for each edge that corresponds to the amount to remove from the triangles on each side of the edge. It is computed as follows (see Figure 3.9):

$$\delta_\theta^* = d_* + d_\theta = d_* + \frac{w_r}{2\sin(\theta/2)}, \qquad * \in \{i, o\} \tag{3.1}$$

The total amount trimmed from the triangles in the unfolding is $2 \cdot \delta_\theta^o$ for a full hinge and $\delta_\theta^i + \delta_\theta^o$. This leaves a gap that is always larger or equal to the hinge width. Since $d_i = d_o$ for our hinges, dihedral offsets are thus symmetric for both full and half hinges. We then insert the full hinge in the gap and snap the triangles back on each side. During hinge insertion the algorithm verifies whether new overlaps are created between triangles. If this occurs, the overlapping triangles are shrunk automatically by a small amount to just avoid the overlap. This happens rarely in practice: on our results only the *sqtorus* model requires a small shrinkage of 0.3 mm on two triangles. Trimming and hinge insertion are shown in Figure 3.10. These are performed between each connected triangle in the unfolding. We refer to the triangle after trimming and hinge insertion operation as the *trimmed triangle* in the rest of the chapter.

Checking hinge suitability. After inserting the hinges in our unfolded mesh, we verify that they can suitably bend by the required angle using our experimental data (see Section 3.3.2). We compute the length for each hinge and use the affine models to check if it can bend within the safety margins to the necessary bending angle. If a hinge is not suitable we report the mesh as incompatible and suggest that the mesh be upscaled by a certain amount.

Adding half hinges. We also use the experimental data to select the hinge type. We favor half hinges as they require less PCB area and trimming than full hinges. We choose half hinges whenever the required bending angle is below the safe threshold angle for a half hinge (see Figure 3.7). Table 3.2 reports the use of half hinges and full hinges on our test models. Naturally, a smoother model with smaller dihedral angles has fewer full hinges, whereas models with sharp angles have to use full hinges more often.

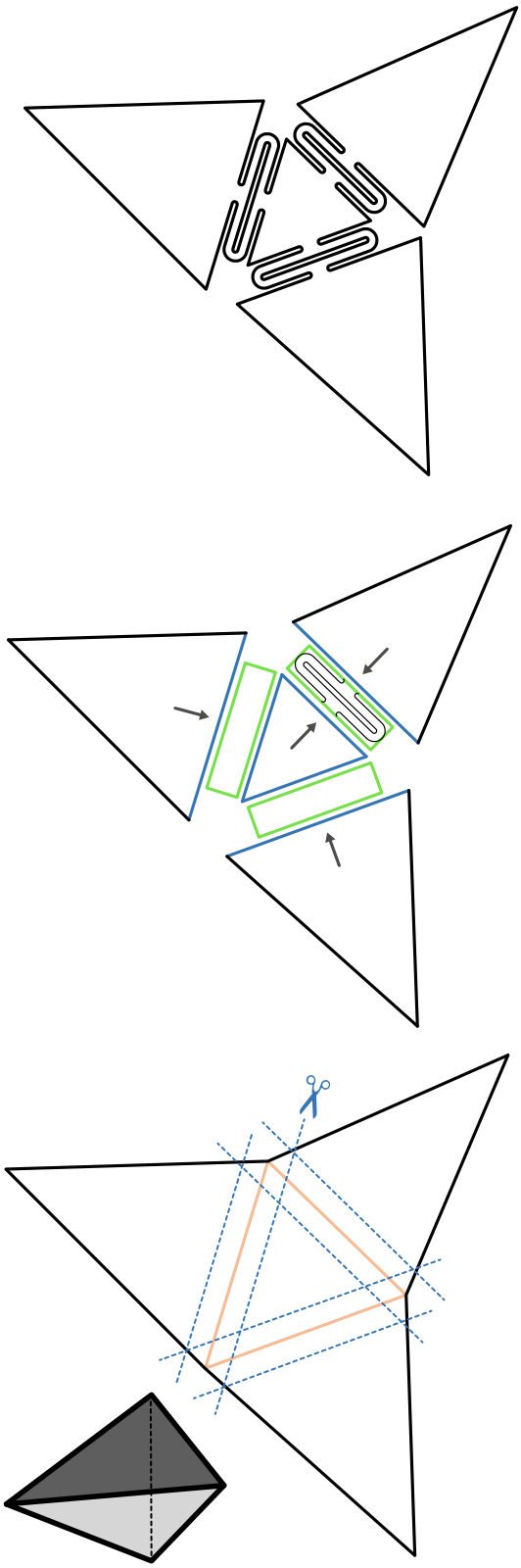


Figure 3.10: Hinge insertion. On the left, triangles are trimmed according to the dihedral offsets. In the middle, hinges are inserted between adjacent triangles, which are then snapped to remove gaps. On the right, the final geometry is shown.

Choice of unfolder heuristic. Unfolding a mesh with minimal patch count is computationally hard, as seen in Section 3.2.3. Thus, algorithms rely on various heuristics depending on their application. In this work, we rely on the minimum dihedral angle and minimum perimeter heuristics. The minimum dihedral angle heuristic penalizes edges that have high dihedral (i.e. sharp) angles and introduces cuts along such edges. The resulting unfolding contains only hinge edges that have comparatively small dihedral angles, making it easier to fold. Small dihedral angles also ensure that we can use half hinges more often, leaving more space for LEDs. The minimum perimeter heuristic instead minimizes the overall perimeter of the unfolding, tending to cut shorter edges where hinges would bend less and empirically results in fewer patches. These two heuristics illustrate a compromise between obtaining a better unfolding with fewer patches, and obtaining an unfolding better suited to our approach, where sharp edges are preferably cut and long edges preferably preserved. The minimal dihedral angle heuristic results in a single patch unfolding for most of our examples (see Section 3.5.5). We expose the choice of heuristic as a design parameter, so the user can choose the one that gives the best result for each model.

3.3.4 Compatible input mesh

One key limitation of our approach are the constraints on the input mesh for it to be compatible with our system. The input mesh is incompatible if any of the hinges is too short for the required bending angle. In that case, the mesh is rejected and will need to be upscaled before using it as an input to the pipeline. A compatible mesh has all edges above the angle-dependent constraint on hinge length, and triangles able to fit at least a module. These requirements are not too constraining: our system works on models with irregular triangles such as *sqtorus* and *star* (see Figures 3.26, 3.28 and the corresponding SVGs in the submission data).

3.3.5 Chamfered support structure

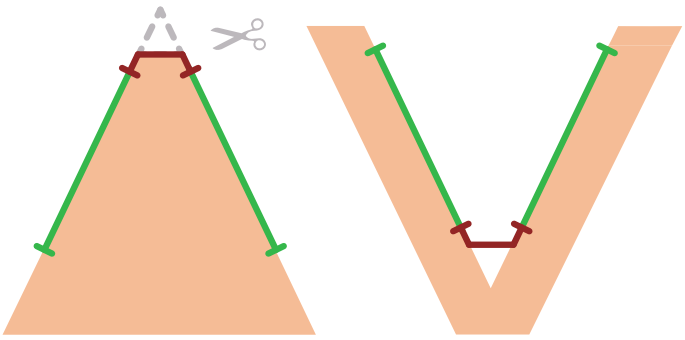


Figure 3.11: Effect of hinge insertion after trimming and chamfering, side view of convex and concave edges. After folding the hinge is now flush with the support. Triangles are no longer displaced from their intended position.

The unfolded mesh needs to be folded back onto the support structure. To facilitate this operation we need to chamfer our support structure. The edges

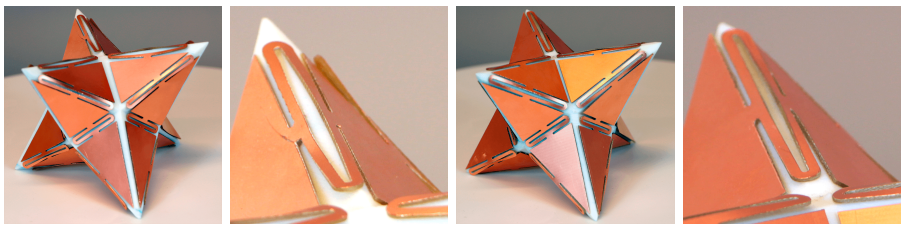
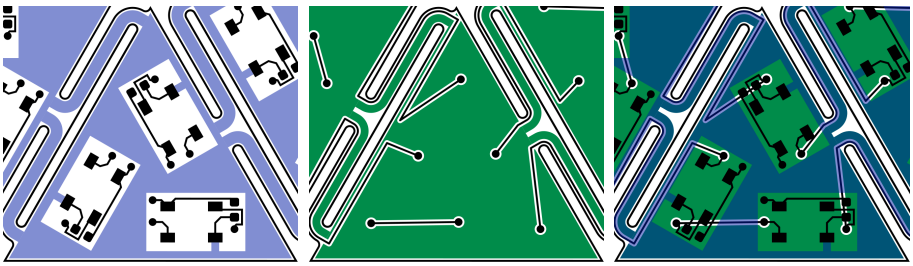


Figure 3.12: Impact of chamfering on hinges. Left: No chamfering. Right: With chamfering. Although chamfering is barely noticeable on the mesh, it greatly reduces hinge deformation.

of the support structure are chamfered such that the hinges can rest of them properly, see Figure 3.11. The convex edges of the support structures are offset by $h = d_\theta \cos(\theta/2)$. Figure 3.12 shows the impact of the chamfer on the state of the hinges after assembly.

3.4 Automatic circuit layout



(a) Top layer. (b) Bottom layer. (c) Both superimposed.

Figure 3.13: Top and bottom view of the PCB blueprint for a triangle. The vcc plane is on the top layer, while the gnd plane is on the bottom. Space for the components is carved out from the vcc plane. Data traces are carved out from the gnd plane and are 0.2 mm wide. vcc and gnd are respectively 1.2 mm and 0.8 mm wide through the hinges.

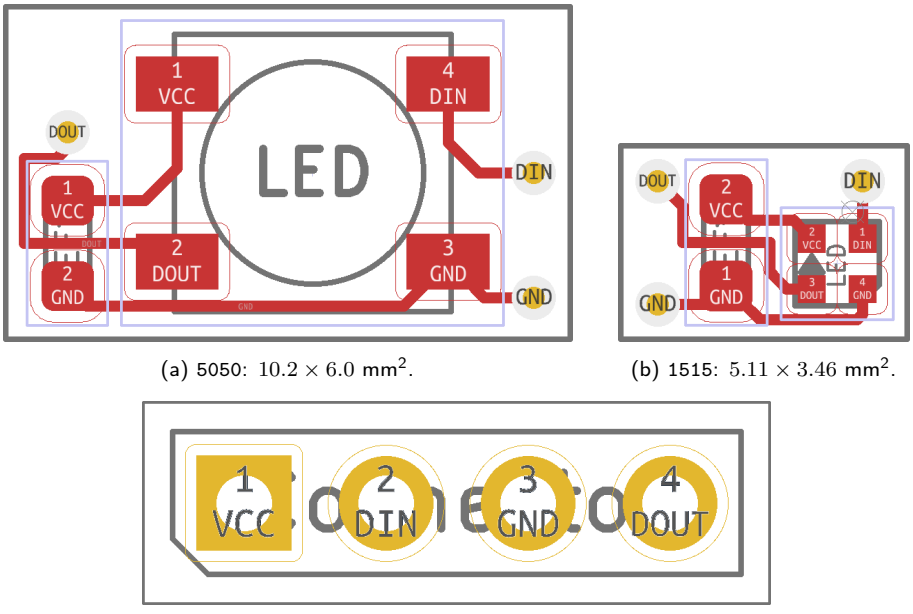
Section 3.3 explains how to fully determine the PCB outline. We now detail in this section how to place the LEDs and generate a circuit layout within the board. The circuit has to form a sequential chain connecting each `data-out` pin of a previous LED to the `data-in` pin of the next, as shown in Figure 3.3. Note that each LED is paired with a 100nF decoupling capacitor connected to ground (`gnd`) and power (`vcc`). We use both layers of the PCB: the top layer carries `vcc` and the surface mounted components (LEDs, capacitors, connector), the bottom layer carries `gnd` and the data traces. The components connect to the bottom layer through *vias*: copper plated holes connecting traces across the PCB layers.

A main observation underlying our approach is that we have full freedom in choosing the order of the LEDs along the sequence, and hence which `data-in`/`data-out` pairs have to be connected by traces. This ordering is a key degree of freedom in making the place-and-route problem tractable. Note that this degree of freedom is normally *not* considered in PCB design [LS20], as one rarely has the opportunity to reorder components in a circuit schematic. In particular, auto-routers in PCB design software can help with tracing routes

— more generally nets — but typically do not perform placement and do not optimize the schematics.

We propose a specialized placer and router that exploits the structure of the circuit as well as the structure of the unfolded PCB. Our algorithm splits the global place-and-route problem into local per-triangle problems that can be efficiently solved, in particular exploiting the freedom of ordering the LEDs along the chain. It is described in Section 3.4. Each disconnected patch of the unfolding becomes a single PCB, each having its own connector with `vcc`, `gnd`, `data-in` and `data-out` pins. The data pins are respectively connected to the `data-in` pin of the first and `data-out` pin of the last LED in the sequence. In a design with multiple patches, each patch is processed independently.

`gnd` and `vcc` planes. Our circuit design uses a `vcc` plane on the top layer, and a `gnd` plane on the bottom layer: initially the layers are fully covered by a conductive copper plane connected to `vcc`/`gnd`, and the other elements (i.e. component and via footprints, data traces) are *carved out* from these planes. This is illustrated in Figure 3.13. This approach allows us to mainly focus on the geometry of the data traces. In addition, using `vcc` and `gnd` planes instead of traces minimizes their resistance. This is important on large designs where narrow copper traces can be multiple-meter long, enough to have a non-negligible resistance.



(a) 5050: 10.2×6.0 mm^2.

(b) 1515: 5.11×3.46 mm^2.

(c) Connector: 11.2×3.6 mm^2.

Figure 3.14: Connector and modules containing an LED and a capacitor, shown at the same scale. Data traces enter a module through the DIN via and exit it through the `DOUT` via. Modules are connected to `vcc` through the LED `vcc` pad, and to `gnd` through the `gnd` via. The larger module (5050) contains a 5.0×5.0 mm^2 WS2812B LED and the smaller module (1515) a 1.5×1.5 mm^2 SK6805-EC15 LED.

Modules We group an LED and its accompanying decoupling capacitor into a *module*: a fixed circuit with a rectangular outline, having predefined solder pads and internal traces, as well as specified points where external traces should connect. Figure 3.14 shows LED modules for different LED sizes as well as the connector. Our algorithm does not manipulate the individual components, but modules as a whole, positioning and connecting them. Each module is replaced by its corresponding blueprint when producing fabrication files. This allows us to abstract the exact layout of the components away during the layout phase, and only consider the rectangular geometry of the module.

3.4.1 Overall strategy

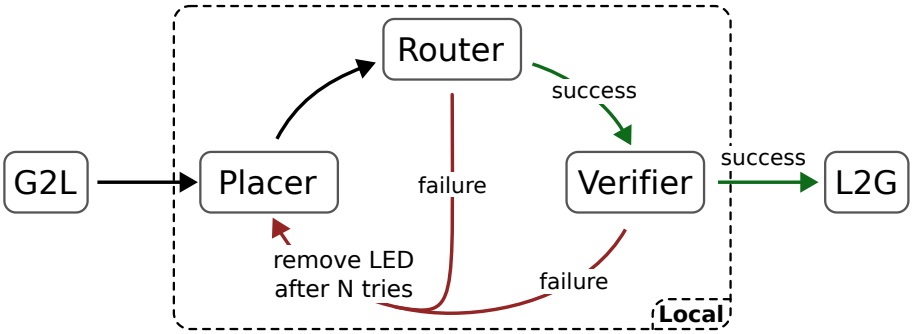


Figure 3.15: PCB blueprint generation pipeline. *G2L* and *L2G* are short for *global-to-local* and *local-to-global* respectively. Everything in the dashed box is done locally per-triangle.

Generating the circuit requires solving for both the *placement* of the modules, their *order* along the chain and the *routing* of the signals between them. These problems — placement, ordering and routing — are interdependent. In particular, some combinations of module placement and order may make routing impossible due to spacing constraints. This is a likely occurrence under our objectives: we target a dense packing of modules chained together by data traces, starting from and coming back to the connector, going through all hinges and every single module. Our overall strategy is summarized in Figure 3.15. We first divide the global problem into a set of small, independent, per-triangle problems. To deal with the interdependence of placement, ordering and routing we iterate between a placer step followed by an ordering-routing step. For clarity we next refer to the latter step simply as *routing*. We quickly loop between placement and routing, generating a new, different placement whenever routing fails. If no solution is found after a fixed number of iterations, the number of modules is reduced. Thus, the placer samples the space of possible layouts for the modules until routing succeeds. Once a valid placement and routing is found, the *verifier* checks if the resulting local layout satisfies additional design and electrical constraints. If not, the triangle goes back to the placement and routing loop. Finally, when all triangles have a valid layout, they are stitched into a global PCB blueprint ready for fabrication.

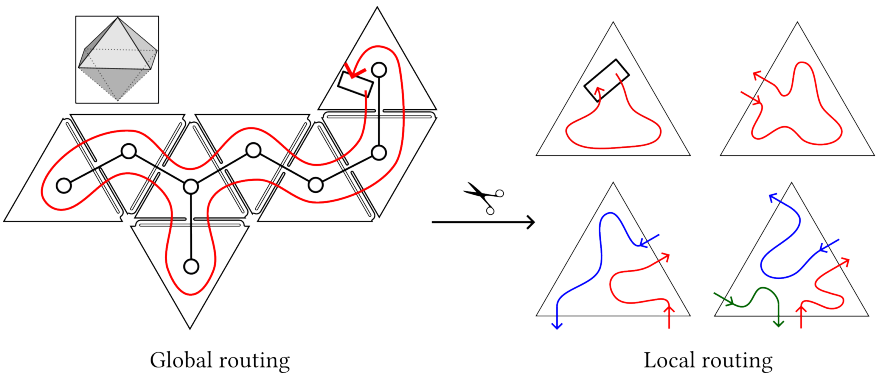


Figure 3.16: Each unfolded patch is traversed in depth-first order starting at the connector (left), traversing hinges right-side first. This counter-clockwise order defines a cycle on the unfolding and fixes the inputs and outputs of each triangle (right).

3.4.2 Global to local

We split the place-order-route problem into per-triangle, independent subproblems. Following a global routing strategy allows us to determine the inputs and outputs to every triangle, and how the data chain flows through the hinges globally. Once this is decided, we can compute a local layout for every triangle independently. Our global routing strategy is illustrated in Figure 3.16. It relies on the fact that unfolded patches have an underlying tree structure. A depth-first traversal on the tree starting at the connector defines a cycle over the unfolding. The cycle goes twice through every hinge in opposite directions, once per side of the hinge. During the traversal each hinge is visited first through its right side. This process defines the inputs and outputs of every triangle, and how they connect to each other within the triangle. We define a *sub-chain* to be a (possibly empty) sequence of modules connected to an input and output to the triangle. Each module that is later placed within a triangle will belong to one of these sub-chains. A triangle has between one and three incident hinges, and the same number of sub-chains. We also use this depth-first traversal to orient half hinges, which are asymmetrical. They are oriented such that their wide sides always point to the children triangles as defined by the traversal order. This facilitates routing as explained in Section 3.4.3.2.

3.4.3 Per-triangle circuit layout

Each triangle is processed independently, making parallel computations possible. Within each triangle we first place modules (Section 3.4.3.1) and then route the signals (Section 3.4.3.2). The resulting routing is then verified to ensure that additional design and electrical constraints are satisfied. (Section 3.4.3.3). This method generates a new placements and routing until a valid layout is found, potentially reducing the number of modules during the process. (Section 3.4.3.4).

3.4.3.1 Module placement

The objective of the placer is to pack as many rectangular modules as possible in the triangle, distributing them as uniformly as possible if there is sufficient space. The placer works on the *trimmed* triangle from the unfolding. This

resembles packing problems in the literature [Xu+20], with strict boundary and overlap constraints as well as a uniform distribution requirement. We propose a custom placer combining a classical CVT with a rigid body collision solver (*Box2D* [Cat24]).

For now let us assume a desired number of modules M is given. We first evenly distribute M points with a CVT (Step 1). These initial points represent the center of the modules to be placed. A rigid body rectangle is then positioned centered on each point, with a random rotation (Step 2). Next, the rigid body solver resolves all collisions, including those with the triangle outline (Step 3). We run it for 50 iterations (experimentally determined) and verify whether a collision-free solution is obtained. If not, the process restarts — for the sake of clarity we postpone this discussion for a few paragraphs. If the test succeeds, the modules are in a non-overlapping configuration. We next refine the solution to obtain an even distribution (Step 4). We run the rigid body simulation once more, now applying redistribution forces. These forces attract each module towards the center of its Voronoi cell, computed from the module centers every iteration. During this step, the Voronoi diagram is computed on the entire triangle *before* trimming, allowing modules to reach and align with the trimmed triangle boundary. This allows for a better distribution over the final object, where it helps hide gaps between hinges. We use 800 iterations (experimentally determined).

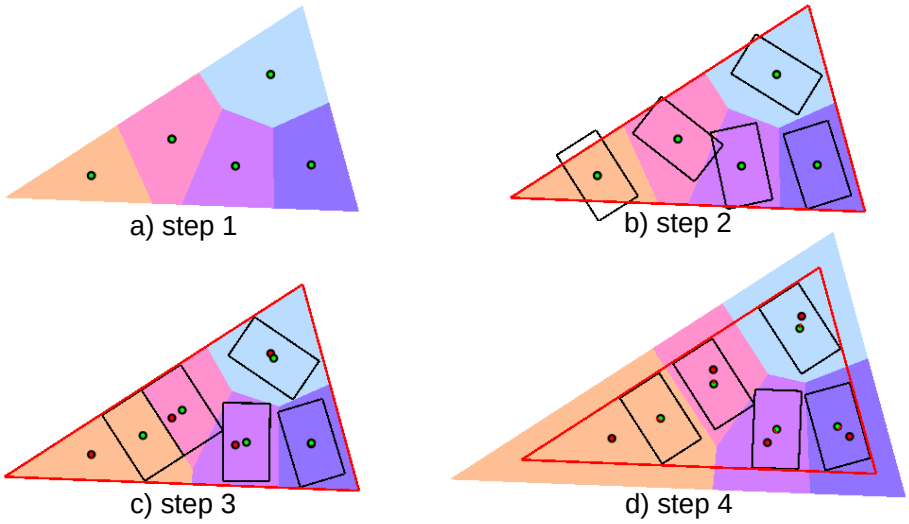
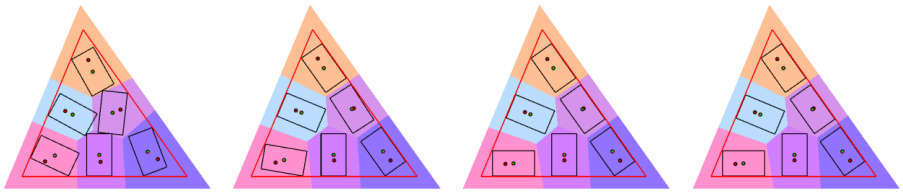


Figure 3.17: Placement algorithm. Step 1: Placing points. Red points specify the Voronoi cell centers and green points specify the center point of modules. Step 2: Spawning rigid bodies. Step 3: Collision resolution. Step 4: Even distribution with untrimmed boundaries.

The entire placement process is illustrated in Figure 3.17. Simulation parameters were adjusted manually as a time-quality trade-off.

Placement loop. The first time the placer runs on a triangle it attempts to place as many modules as possible. The initial value for the target number M is the triangle area divided by the module area, times a factor of 0.55 (experimentally determined), rounded down. If a valid placement for M modules

is found in less than 4 (experimentally determined) tries of step 3 (collision resolution) M is incremented by one. If all tries fail M is decremented by one. This process stops once we find a valid placement for M and fail to pack $M+1$ modules. Later on the placer might be called again for the same triangle in cases where routing or verification fails (Section 3.4.3.4). The number of modules to place is then given to the placer.



(a) Iteration no. 0 (b) Iteration no. 400 (c) Iteration no. 800 (d) Iteration no. 1200

Figure 3.18: Impact of the number of iterations spent on step 4 of the placer.

Placement quality and timing. Timings are given in Table 3.2 for different models. Step 4 of the placement algorithm dominates the timing. Based on the desired placements quality, the user may choose to end this step earlier thereby reducing the placement time. We show the effect of the number of iterations in step 4 on the result in Figure 3.18. We always use 800 iterations in our results, which gives a good compromise between execution time and placement quality.

Note that the choice of the placement parameters (0.55 packing density, 4 attempts per module number) have a significant impact on the performance of the placement step. Indeed, if the initial guess is poor, we might need to increment or decrement the number of modules many times. Since each failed attempt requires multiple tentative placements, this can quickly snowball. More sophisticated approaches could be devised by experimentally studying packing density as a function of the triangle size and shape. Then the number of modules could be modified by a variable step depending on the current packing density and the empirically derived one.

User-defined LED density. Optionally, the user can specify a target average distance between the modules within a triangle, thereby controlling the LED density and hence the overall appearance of the model. High LED densities produce gaps in the final distribution corresponding to the hinges, while cut edges in the unfolding meet seamlessly. This results in noticeable hinge gaps (see Figure 3.19, left). With a proper choice of target spacing, the user can control the density so that the gaps are no longer noticeable (see Figure 3.19, right).

3.4.3.2 Circuit routing

The router operates on a trimmed triangle with modules positioned by the placer and the sub-chain inputs and outputs determined during the global-to-local step. It outputs a set of data traces implementing the desired connectivity. The router does not need to consider collisions between traces and modules,

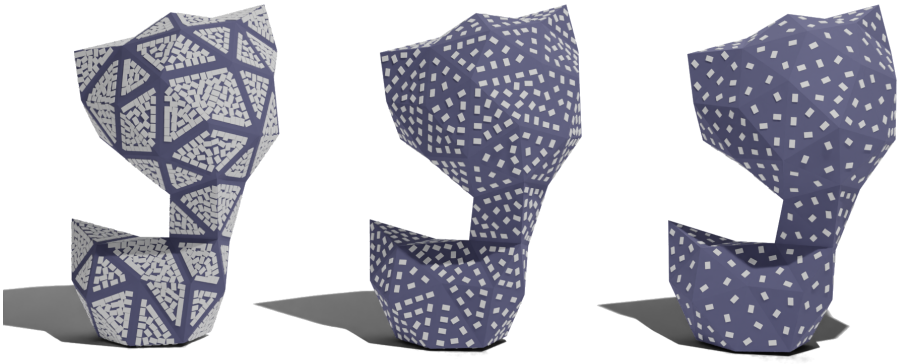
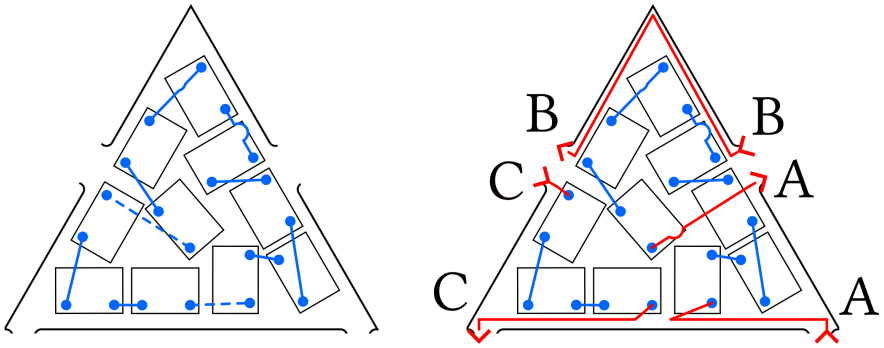


Figure 3.19: The user controls the density of LEDs on the result by specifying a target spacing: (left) 5.0 mm (middle) 7.5 mm (right) 10 mm. The results show the renderings of *cat* model with different LED densities.

since these lie on different layers of the PCB (bottom and top respectively, see Figure 3.13). Vias are later placed to connect the module pins to the data traces.



(a) Module cycle. (b) Hinge cuts.

Figure 3.20: Overview of the ordering and routing within a triangle. Left: Module cycle, dotted lines where traces are cut. Right: Hinge cuts connect the cycle to the hinges to obtain the desired global topology, labels with the same letter are ends of the same sub-chain, arrows show their direction.

Our router functions in roughly two steps, illustrated in Figure 3.20:

1. Connect all modules to form a cycle, adding non-overlapping data segments between their `data-in`/`data-out` pins.
2. Choose where to cut the cycle to connect the modules to the sub-chain inputs and outputs, resulting in the expected circuit topology.

We keep the router fast and simple by performing only intersection and spacing checks between traces, ignoring vias and possible disconnects of the `gnd` plane. Therefore, the verification step that follows (Section 3.4.3.3) may detect issues and re-run the process with a different choice of parameters (Section 3.4.3.4). We now describe each step in more details.

Connecting modules. We form a cycle passing through every module without self-intersections, adding segments between each successive `data-out`/`data-in` pair. At first sight, obtaining such a cycle seems to amount to finding a shortest Hamiltonian cycle in the complete graph with the modules as nodes. However, the problem is made more difficult by the fact that the distance between any two modules depends on their specific orientation. Since the `data-out`/`data-in` pins can be anywhere in a module (depending on the footprint), flipping a module can have a large impact in the quality of the solution. The original design of the `5050` module footprint was symmetrical to avoid this issue, but was significantly larger.

We propose a dedicated approach based on a divide-and-conquer strategy. We note that for small instances of the problem (e.g. less than 8 modules), a brute-force exploration of all module orders and orientations is possible since intersection checking is very fast. We exploit this property as follows. For larger instances of the problem we start by computing a Hamiltonian cycle through the module centers using a Traveling Salesman Problem (TSP) solver [App+01]. Recall that a shortest Hamiltonian cycle under Euclidean distance cannot contain intersections. From this cycle, we decide the orientation of each module. The orientation of a module is binary: it is either flipped or not. This is due to the rotation of the module being fixed during the placement step.

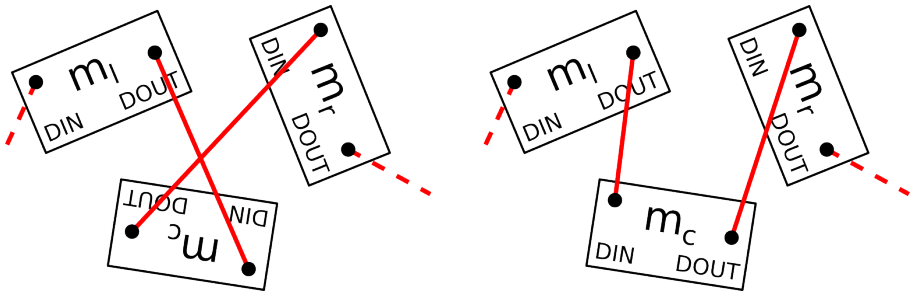


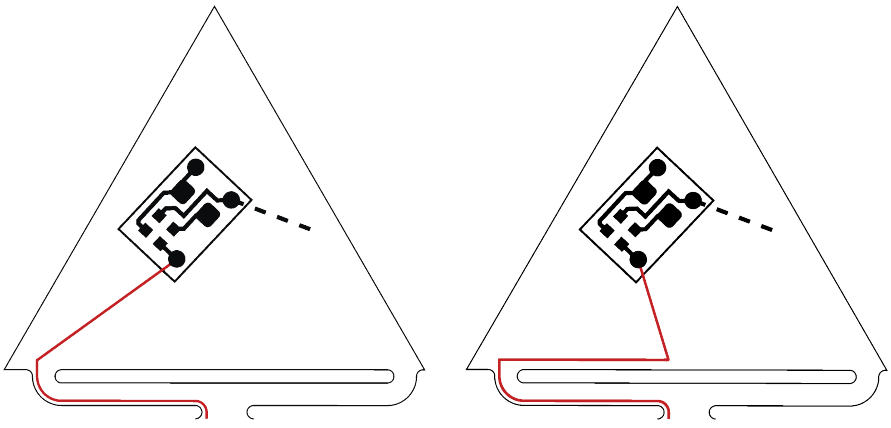
Figure 3.21: Both orientations of m_c for given orientations of m_l and m_r. A bad orientation leads to an intersection (left) while a good orientation produces no intersection (right).

The problem has a specific structure we can exploit: the best possible orientation of some modules can be easily determined. Consider three modules m_l, m_c, m_r consecutive in the cycle. Given a choice of orientations for m_l and m_r, we can compute which orientation for m_c avoids intersections between traces connecting the modules and minimizes their length. This can be seen in Figure 3.21. If m_c has the same best orientation for all possible orientations of m_l and m_r, then we say it is a *stable module*. The orientation of stable modules is independent from the orientation of the other modules and can be fixed from the start.

Interestingly, in our placement results, stable modules tend to be spread out along the TSP sequence, with only short unstable sequences in-between. Longest unstable subsequences rarely contain more than 8 modules on average, even in triangles with tens of modules. We therefore split the cycle into these unstable subsequences of unknown orientation, and run a brute-force search for each one. The length of the longest unstable module sequence in a triangle reaches a maximum of 18 value in our examples. This happens for the model *sqtorus* with `1515` modules in a triangle with 64 total modules. All other

examples have lengths of at most 14. Note that the brute-force search can stop as soon as a valid solution for the sequence is found, without exploring all possibilities.

Hinge cuts. Now that we have a cycle defined by the order and the orientation of the modules, we need to split it into module sub-chains connected to the inputs and outputs of the triangle, as shown in Figure 3.20. The cycle needs to be cut in a number of places equal to the number of hinges incident to the triangle. We name *hinge cuts* the places where the cycle is split and connected to the hinges. Sub-chains are connected to inputs and outputs as to preserve the desired global circuit topology (Section 3.4.2). A hinge cut replaces a trace between two modules with a set of traces, connecting the output of the module before the cut to a triangle output, and the input of the module after the cut to a triangle input. A hinge cut is valid if the newly added traces do not intersect any existing traces, and have enough space between them to avoid disconnecting the `gnd` plane. We find a valid set of hinge cuts by enumerating all possibilities without intersections, choosing the set maximizing spacing between traces. This is the main performance bottleneck of the routing step as the number of modules goes above 50 in a triangle with 3 hinges. This only happens in our *sqtorus* model with `1515` LEDs, which contains relatively large triangles. This problem could easily be addressed by replacing the exhaustive search with a incremental search starting with traces closest to the hinges. If no solution without intersections is found, routing fails and a new placement is generated. Note that a resulting sub-chain can contain zero modules, in which case the added traces go around the edge of the triangle (see sub-chain B in Figure 3.20). If a valid set of hinge cuts is found, we proceed to the layout phase that performs additional verifications.



(a) Direct connection. (b) Narrowing connection.

Figure 3.22: Different types of connection between a module and a half hinge (wide part). Only the left connection is shown here for simplicity, but both connections can also happen for the right connection.

Half hinge parametric connections. Contrary to full hinges, the wide part of a half hinge makes the connection points for data traces be very far apart

from one another (see Figure 3.4). This makes it more likely that direct traces from a module to this type of connection point will cause intersections with other traces. To circumvent this, we introduce a *narrowing connection* that goes first to the center of the hinge and then to the corresponding connection point, see Figure 3.22. These take more space along the border of the triangle but are less likely to cause intersections. Both direct and narrowing connections are tested when solving for valid hinge cuts and the best one is kept. We also limit difficult cases by orienting half hinges such that at most one wide side points towards any triangle. This is done during the global-to-local phase as described in Section 3.4.2.

3.4.3.3 Local layout and circuit verification

This step takes the module placement and routing information and generates the corresponding geometric layout for the triangle. The `vcc` plane is created on the top layer, modules are carved out, and the corresponding footprints are inserted. The `gnd` plane is created on the bottom layer, traces and vias are added and carved out from it. Traces that intersect vias are modified to avoid them. These steps are performed as boolean operations on the polygons forming the traces and `gnd`/`vcc` planes. Routing does not consider these final steps. For this reason, the generated layouts can sometimes be invalid. We thus have to run additional verifications on the generated geometry.

Verifications are done on the bottom layer to ensure that the `gnd` plane contains all `gnd` vias in the triangle and is properly connected to the hinges. In practice, this means checking that there are no total disconnects of the plane, or narrowings below a specified length threshold. This is done with a morphological opening of the `gnd` plane, using the minimal trace width (0.2 mm) as the opening diameter. We also check that the modifications to the data traces do not create intersecting geometries. No verifications need to be done on the top layer since the module footprints contain an empty border that guarantees that they cannot fully disconnect the `vcc` plane. If any of these tests fails, the layout is discarded and the triangle is sent back to the placer.

3.4.3.4 Placement-routing loop

The routing process may fail either during routing itself (Section 3.4.3.2) or verification (Section 3.4.3.3). Both cases send the triangle back to the placer, starting the loop again. It can be challenging to find a layout for very dense packings of modules. Because of this, after a number of failed attempts for a given number of modules, we decrease the number of placed modules by one, until a valid layout is found. This number is a parameter given to the pipeline. We found that a single try is enough to obtain good results.

3.4.4 Fabrication-ready schematics

Once a valid local layout is computed for each triangle, we stitch them back together to obtain the global PCB blueprints. Our software outputs a set of SVG files representing the PCB outline, layers, traces and components which we load into *KiCAD 6* using the *import from SVG* feature. From *KiCAD 6* we

generate the Gerber fabrication files and upload them to the PCB fabrication service together with the bill of materials (list of components).

3.5 Results

We demonstrate our approach on physical models and virtual examples. We first discuss the fabrication and assembly process (Section 3.5.1), then explain how we design the lighting effects along the surfaces (Section 3.5.2) and showcase our results on physical models (Section 3.5.3). We use our pipeline to produce additional virtual examples (Section 3.5.4). We conclude by analyzing the performance of the pipeline on our results (Section 3.5.5). The dimensions of all models are listed in Table 3.1. We provide in the submission data linked in the introduction the circuit blueprints for all results (SVG files and Gerber fabrication files). The submission video showcases the physical and virtual results with animated light patterns.

Fabricated results showcased in the figures do not use the maximal density allowed by the system unless specified otherwise. The fabrication of the results was done in two batches each with a different version of the placer, due to time constraints for publication. The first batch was sent for fabrication with a purely CVT-based placer, whereas the second one used the placing algorithm described in Section 3.4.3.1. A comparison of the two batches for *cat* can be seen in Figure 3.23. Additionally, Figures 3.24 and 3.25 show the max-density *star* and *icosa* PCBs. SVGs illustrating the maximal density can also be found in the submission data.

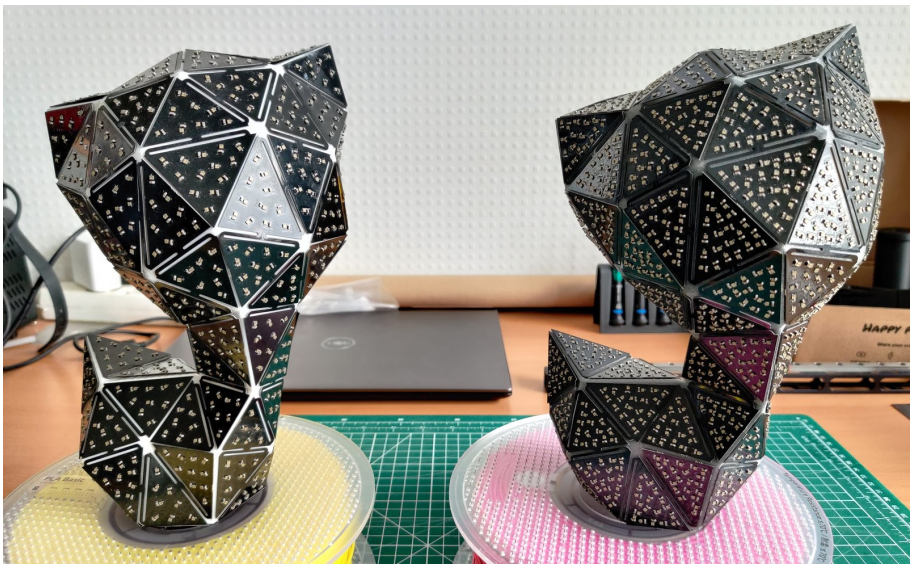


Figure 3.23: Side-by-side comparison of the two fabricated *cat* models with 1515 LEDs. Left: low-density batch with 979 LEDs. Right: max-density batch with 1881 LEDs.

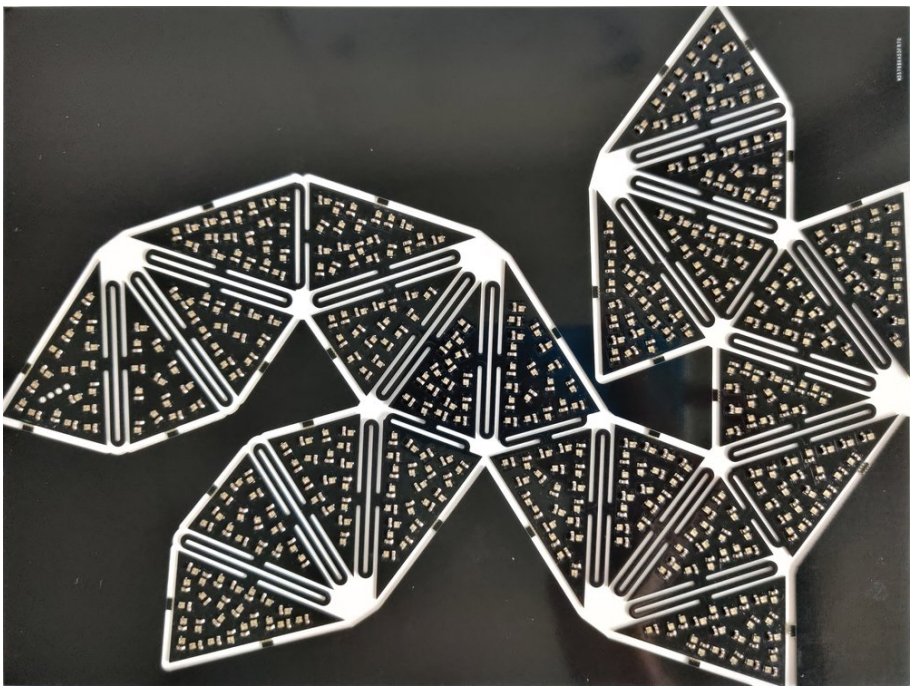
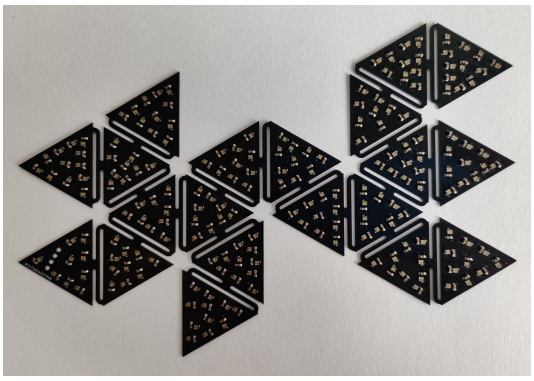
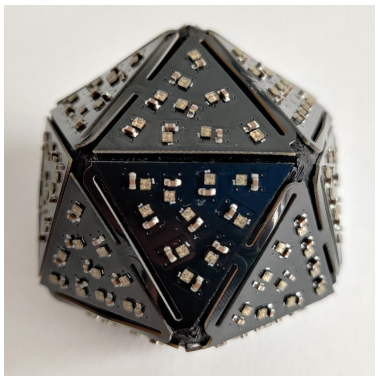


Figure 3.24: Max-density *star* PCB laying flat. The extra material on the outside is attached to the PCB by narrow, holed connections. This material is kept to stabilize the PCB during fabrication and shipping.



(a) Flat *icosa*.



(b) Assembled *icosa*.

Figure 3.25: Showcase of the max-density *icosa* PCB (images at a different scale).

3.5.1 Fabrication and assembly

We fabricated four models: *icosa*, *star*, *sqtorus* and *cat*. For PCB manufacturing and component soldering we used *PCBway*, a popular online service. The unfolding contour is eroded everywhere by 0.5 mm (1.0 mm for *icosa*) due to fabrication tolerances. PCBs are manufactured in double-layer 0.6 mm thick FR4 material. All LEDs and capacitors come already soldered, with only the through-hole connector remaining for us to add. We 3D printed the support meshes in-house, using PLA filament on a Creality CR-10S Pro V2 FDM printer.

Table 3.1: Dimensions of axis-aligned bounding boxes of the models and their unfoldings. Models measured as featured in the figures, unfoldings measured from fabrication files. The unfolding of *archi* has two patches, hence the two bounding boxes.

Name	3D model (mm^3)	Unfolding (mm^2)
icosa	$64 \times 64 \times 53$	172×150
star	$118 \times 103 \times 84$	187×257
sqtorus	$195 \times 195 \times 130$	292×505
archi	$200 \times 224 \times 55$	$154 \times 202 + 291 \times 234$
dome	$255 \times 255 \times 93$	388×407
batman	$187 \times 229 \times 120$	396×346
cat	$144 \times 138 \times 240$	548×466

Folding the PCB onto the support is done by bending and gluing. The process typically starts by identifying a first triangle between the PCB and the surface, and then progressively folding the board around the shape. This is easily achieved by one person on small results (*icosa*, *star*), but requires a second person for larger models (*cat*, *sqtorus*). The whole process took from 15 minutes (*icosa*, *star*) to 70 minutes (*cat*), please refer to the accompanying video for assembly sequences. The main difficulty is holding the folded board in place while the glue settles, but that is otherwise a simple process. Several options could be considered to avoid the glue, such as 3D printed snap-fit mechanisms or the use of thread forming screws for plastic.

3.5.1.1 Optional light diffuser

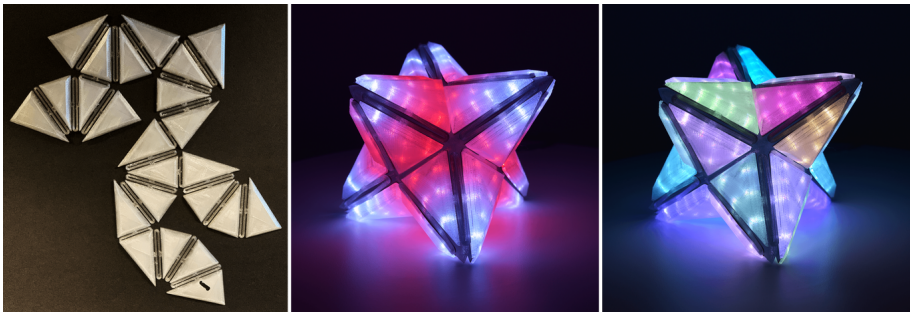


Figure 3.26: 3D printed light diffuser for the *star* model (left). Light effects on the design: inside-out pulse (middle) and per triangle colors (right).

LEDs appear as bright point light sources. For some lighting applications, it is desirable to use an optional light diffuser. We produce diffusers as a 3D printed foldable part reproducing the PCB outline, with individual rectangular housings for the LEDs. It easily folds onto the PCB with the LED housings snapping in place onto the soldered components. A diffuser is shown before and after assembly in Figure 3.26, and can also be seen in Figure 3.27.

Table 3.2: Statistics gathered when running the pipeline on our models. Run on a computer equipped with an AMD Ryzen 9 5900X CPU, a GeForce RTX 3080 GPU, and 32 GB of RAM. Ran with one try before removing an LED upon router/verifier failure, 1.2 mm minimum spacing between input/output traces connecting to hinges, and 0.2 mm minimum spacing between data traces connecting modules.

Model information				Module	LED Modules				Failures			Execution times (s)			
Name	Faces	Patches	Hinges	type	Max	Final	%loss	Avg/face	Total	Router	Verifier	Total	Placer	Router	Verifier
icosa	20	1	0F, 19H	5050	43	41	4.5%	2	4	1	3	548	548	< 1	< 1
				1515	204	201	1.5%	10	3	0	3	789	787	1	1
star	24	1	18F, 5H	5050	111	109	1.8%	5	2	1	1	724	724	< 1	< 1
				1515	458	447	2.4%	19	10	7	3	1168	1162	4	2
sqtorus	48	1	0F, 47H	5050	552	545	1.3%	11	7	5	2	1774	1763	8	2
				1515	2057	2011	2.2%	42	42	25	17	4468	3334	1122	12
archi	80	2	1F, 77H	5050	193	190	1.5%	2	17	6	11	2280	2279	< 1	1
				1515	860	852	0.9%	11	8	1	7	2915	2909	2	4
dome	81	1	0F, 80H	5050	461	458	0.6%	6	3	1	2	2874	2871	2	2
				1515	1837	1819	1.0%	23	17	3	14	4293	4202	82	9
batman	92	1	3F, 88H	5050	290	289	0.3%	3	1	0	1	2826	2824	< 1	1
				1515	1296	1278	1.4%	14	18	9	9	3841	3817	18	6
cat	102	1	0F, 101H	5050	460	460	0.0%	5	0	0	0	3355	3352	1	2
				1515	1908	1881	1.4%	19	27	12	15	4833	4776	48	9

3.5.2 Lighting up the shape

The ability to display colors at many locations along the surface raises the interesting question of how to create lighting effects on our fabricated object. We implemented a small shader-like program, assigning an RGB color to each LED over time. A function is called on each LED, taking as input the current frame time and the LED identifier for which to compute the color. It returns the color as a 24-bit RGB triple directly sent to the LED. The shader has access to the entire mesh information (vertices, normals, topology). This enables on-surface effects such as a geodesic propagation, moving and winding lights. The LEDs are very bright, and therefore the light effects cover a high dynamic brightness range. This is best seen in our accompanying video. We implement the shader as a script in the Lua language, dynamically reloaded upon save with immediate effect on the surface. A simple shader is shown in Listing 3.1. The data is streamed through UART (1 MBd) to an external LED controller implemented on an FPGA (Lattice ECP5). The controller runs at 100 MHz and can drive several boards in parallel with up to 2048 LEDs per board in its current implementation.

3.5.3 Fabricated and lit results

We fabricated four designs, two small ones (*icosa*, *star*) and three larger ones (*cat*, *sqtorus*, *batman*). The price of a single PCB (without components and assembly) ranges from USD$10 per unit (*icosa*) to USD$63 per unit (*cat*). On all models, the cost of the PCB is actually less than 20% of the total, the rest of the price being the cost of components (LEDs and capacitors) and the assembly service.

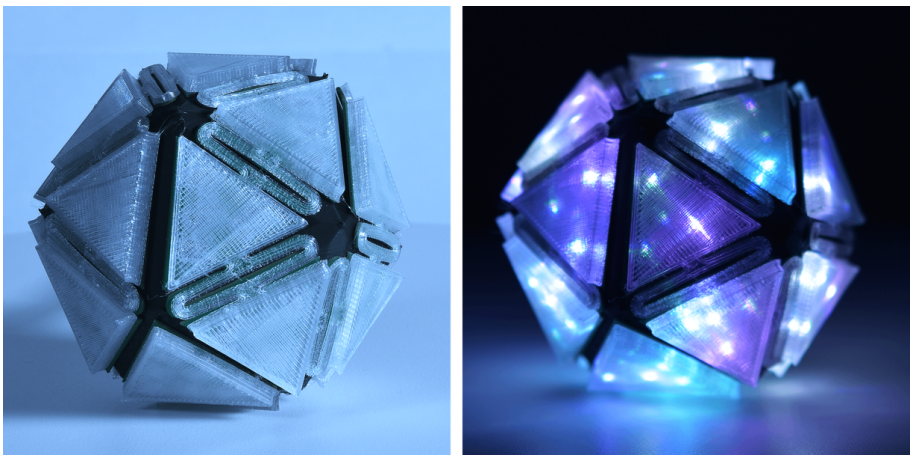


Figure 3.27: *icosa* model with its diffuser (left), lit by a sparkling effect (right).

icosa is a small sphere-like shape shown in Figure 3.27. It only requires half hinges and is quick and simple to assemble, while already enabling complex directional light effects. Our fabricated version features 86 LEDs.

star is a challenging model in terms of angles as it features six spikes with sharp angles between some of its 24 faces. It is shown in Figures 3.2 and 3.26. Our unfolder produces a single patch with mostly full hinges, featuring 166

Listing 3.1: Example shader computing a diffuse reflection with a varying light direction. The shader supports basic mathematical functions and basic vector operations.

```
function diffuse(id, pos, normal)
    -- Compute time elapsed since the start
    t = (time - start_time) * 0.01;

    -- Define darkest and lightest colors
    dark  = Vec3(0.002, 0.002, 0.001);
    light = Vec3(0.5, 0.5, 1.0);
    color = dark;

    -- Define initial light direction
    -- and time-varying angle
    ld = Vec3(1, -1, 2); ld:normalize();
    a = 0.04 * pi * t;

    -- Rotate the light direction
    ld = Vec3(cos(a) * ld.x + sin(a) * ld.y,
              -sin(a) * ld.x + cos(a) * ld.y,
               ld.z)

    k = max(normal:dot(ld), 0.0);

    -- Return a color based on the dot product between
    -- the light direction and the LED normal vector
    color = dark:mult(1-k):add(light:mult(k));
    return Vec3(0, 0, 0) + color * 32.0
end

function main(id, pos, normal, tri, mapi)
    return diffuse(id, pos, normal);
end
```

LEDs. The *star* model is a good example of how using a diffuser allows the faces to become fully lit. Once animated, the shape produces intricate effects between occlusions, moving light patterns and reflections around.

cat is the model featured in Figure 3.1. It has 102 faces supporting 979 LEDs. The unfolder produces a single patch using only half hinges. This is the model most delicate to fold due to its more complex shape. The final result is a sculpture that can emit light in all directions, and self-illuminate. Effects exploiting surface properties are particularly noteworthy on this model, producing an uncanny effect as the surface becomes alive. We also fabricated it with maximal density and 460 5050 LEDs, as shown in Figure 3.31 (right).

sqtorus is a model of a quarter of a torus. It has 192 faces and features 678 LEDs. We fabricate a full torus using four assembled pieces to obtain a torus screen with 2712 LEDs in total, see Figure 3.28. For fast refresh the four boards are driven in parallel by the controller, but for convenience it appears as a single sequence of 2712 LEDs to the shader interface. The torus enables strong

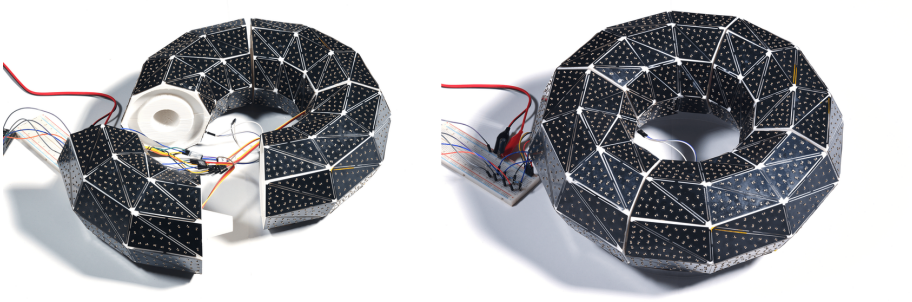


Figure 3.28: A torus assembled from four *sqtorus* tiles. Note that different assemblies from *sqtorus* tiles are possible. See Appendix 3.A for another virtual configuration.

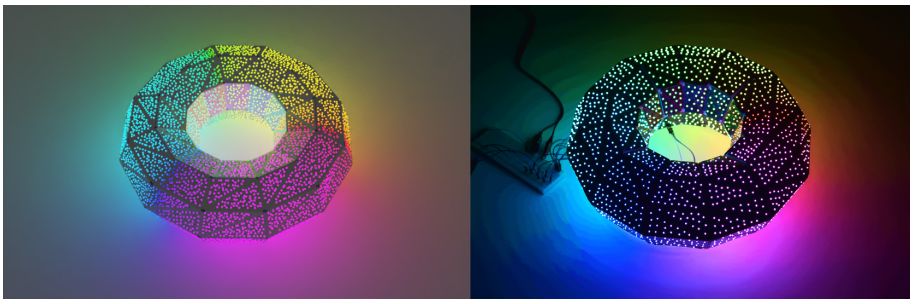


Figure 3.29: Left: virtual preview at maximal density. Right: physical model at a lower density.

directional effects and is sufficiently powerful to light a space, drawing upwards of 3 A on full power. Lighting effects are shown in Figure 3.30. The tileable pieces can be arranged in different and possibly larger shapes, an example appears in the accompanying video and Appendix 3.A. Figure 3.29 shows a preview of the final result for torus alongside the physical model. Note that the LED density of the virtual example is higher than the real one, matching the second batch of fabricated objects.

batman is a model of a face mask, with 92 faces featuring 289 `5050` LEDs. It mixes full and half hinges (3 and 88 respectively), and is an open mesh. The fabricated object can be seen in Figure 3.31 (left).

3.5.4 Virtual examples

We run our pipeline on additional models illustrated in Figure 3.32, for which statistics are given in Table 3.2. *batman* is a mask covered with LEDs. In this case the input mesh is an open surface, which is supported by our pipeline without any modification. *dome* is another open surface, showcasing a potential light dome built from our system. Both of these models unfold as a single patch. *archi* is a surface inspired by architectural results resulting in two patches. Our method allows to choose between different LED modules and to preview the expected outlook after fabrication. Figure 3.33 shows the use of our pipeline on `5050` and `1515` LED modules. The user can also test different LED patterns on the simulated results to get a design better catering to their choice. The LED density can also be adjusted, see Figure 3.19. The preview of the final appearance is faster than running the whole pipeline to generate

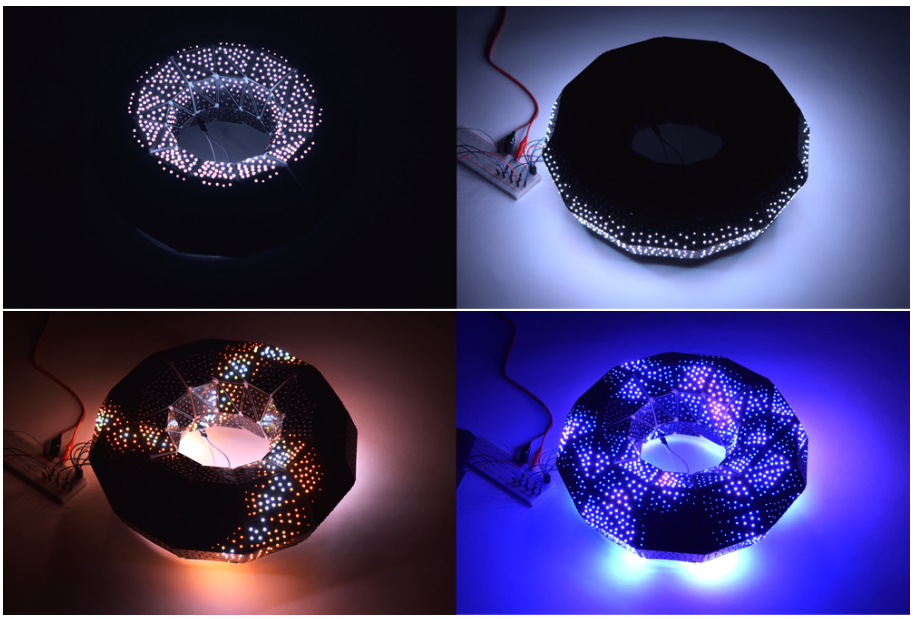


Figure 3.30: The torus produces directional light patterns that strongly reflect on its surroundings. These effects are best visualized in the accompanying video.

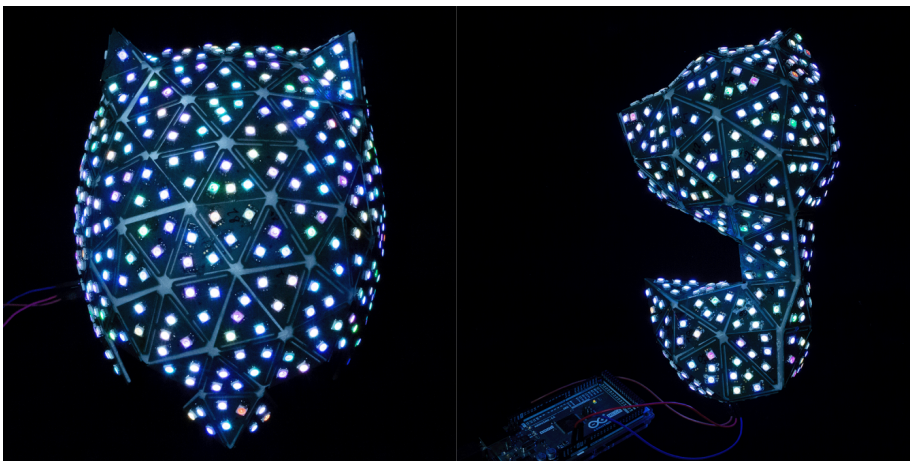
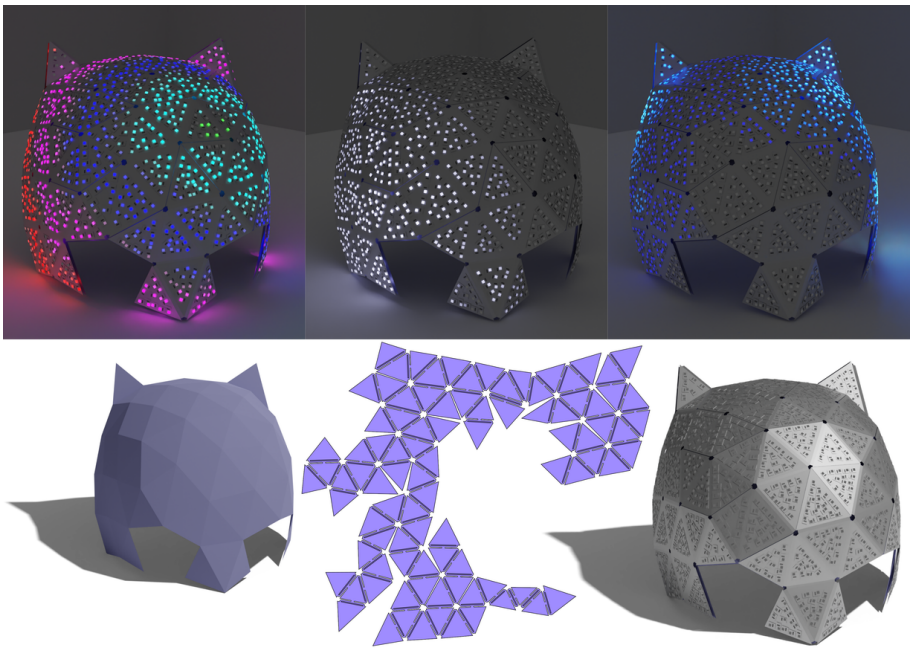


Figure 3.31: *batman* and *cat* model fabricated with big LEDs using the maximal density of our placement algorithm.

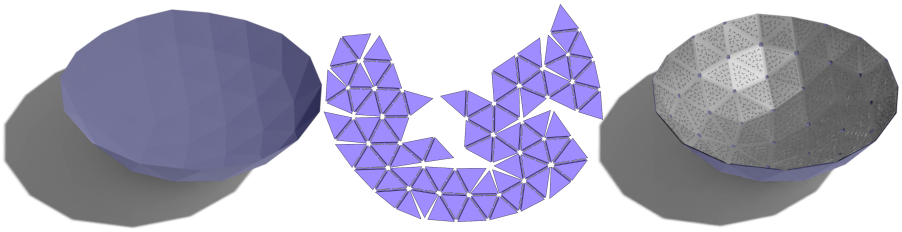
the real blueprints, since we can skip the routing and verification steps. More virtual results with maximal LED density are shown in Appendix 3.A.

3.5.5 Statistics on different models

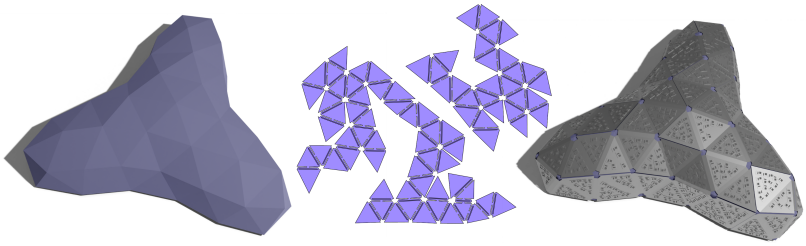
Key statistics regarding the performance of our pipeline are given in Table 3.2 for all test models and two different sizes of LED. When discussing performance we report timings for a single thread. However, placement and routing are performed per-triangle and could be trivially run on different cores or machines. In CPU time the full process takes from 9 minutes for *icosa* to 81 minutes for



(a) *batman*.



(b) *dome*.



(c) *archi*.

Figure 3.32: Showcase of different virtual results. Pictured from left to right are the original mesh, the unfolded mesh, and the folded PCB. The topmost example additionally features simulated lighting effects.

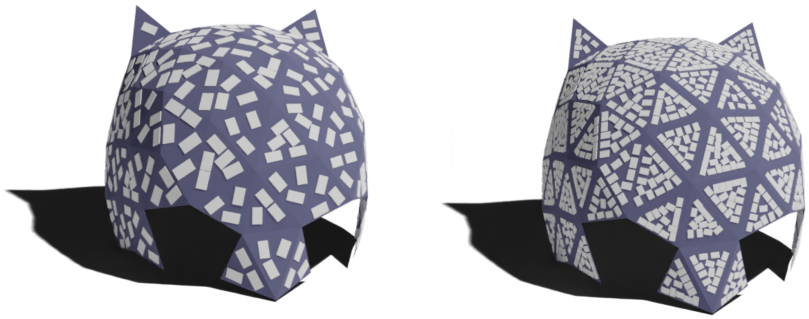


Figure 3.33: *batman* model covered with with 5050 LED modules (left) and 1515 LED modules (right).

cat. In all models but *sqtorus* the time is dominated by placement. For *sqtorus* routing represents 25% of the total runtime. This is due to the larger number of LEDs per triangle in the model, 42 on average. Routing may fail for a given triangle. Table 3.2 compares the maximum number of LED modules placed at any time – including failed attempts – against the final number of LEDs in the blueprint. In all cases less than 2.5% of the modules are lost, except on *icosa* (5050 LED) which is at 4.5%. The unfolder runs in a few seconds for all models, producing single-patch unfoldings for all models except *archi*. We use the minimum perimeter heuristic to obtain fewer patches on *star* and *archi*.

3.6 Discussion and conclusion

Starting from just a 3D model of a surface, our approach enables the creation of curved displays composed of individually addressable RGB pixels, covering the entire object. By relying on standard PCBs we make fabrication scalable, reliable and cost-efficient. This allows us to experiment with a wide array of lighting effects using a shader-like language. As we show with the torus example, tileable designs can be created to produce larger shapes and enable reuse through reconfiguration.

Currently, our method does not support incompatible meshes as discussed in Section 3.3.4. On incompatible meshes, a standard CVT remesher will greatly improve the triangle quality and make it suitable for our technique. In general, a specialized mesher that converts a highly-detailed 3D models into lower-resolution approximations adapted to our pipeline is an interesting direction for future work. This would require exploring tradeoffs between triangle sizes and dihedral angles.

As the number of LEDs per-triangle increases the cost of the enumeration-based ordering and routing becomes problematic. This, of course, could be attacked by subdividing a triangle into a divide and conquer approach, another possibility being to resort on stochastic exploration. Additionally, every part of our pipeline conceptually generalizes to meshes with convex polygons as faces. Supporting polygonal meshes would improve results such as the torus, where hinges between coplanar triangles are not necessary. Finally, even higher LED

densities can be reached by reducing the number of decoupling capacitors in the circuit (currently one per LED).

Beyond lighting effects, we believe our work could inspire general extensions of standard electronic manufacturing workflows with computational tools allowing to transform flat designs to 3D surfaces, thereby enabling a plethora of exciting new applications. In particular, we believe that this approach could also be extended to more complex electronic circuits, integrating other types of components such as accelerometers or touch sensors, that would enable interactive experiences. This system is straightforward to generalize to more complex chained units that fit into the triangles. Placement and routing within a triangle can be handled in a similar manner or with an out-of-the-box placer and router. Global routing can be carried out in the same way as for LEDs. If the units are not connected in a chain, this requires more consideration. Indeed, the kinds of information that can be transmitted between units is directly limited by the track width in the hinge design (w_t in Figure 3.4), which is likely to impact the bending capability of the hinge. Further bending experiments should be carried out to evaluate the impact on the bending angle of track width and other parameters such as PCB thickness. By enlarging the track width on paths with heavily interconnected units, it would be possible to embed more complex circuits onto the foldable PCBs.

3.A Additional virtual examples

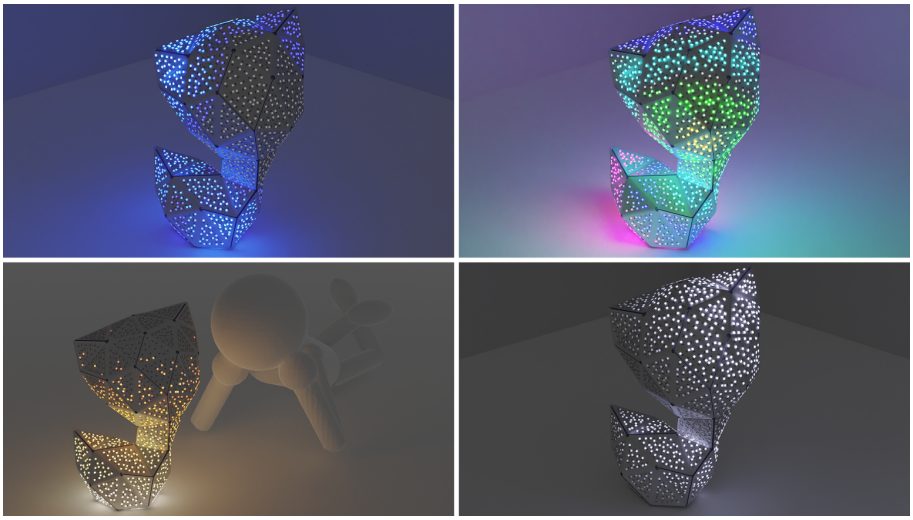


Figure 3.34: Virtual results of our *cat* model.

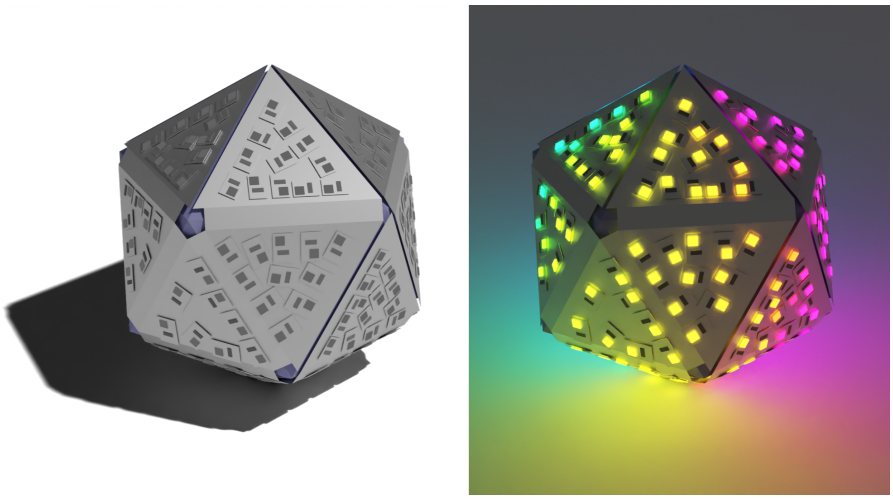


Figure 3.35: Virtual results of our *icosa* model.

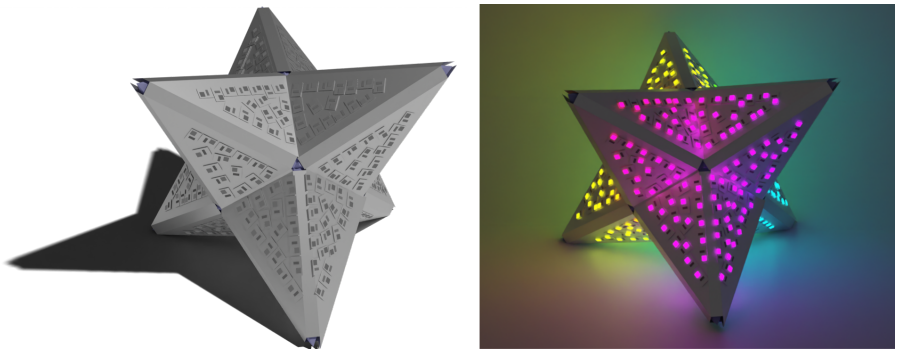


Figure 3.36: Virtual results of our *star* model.

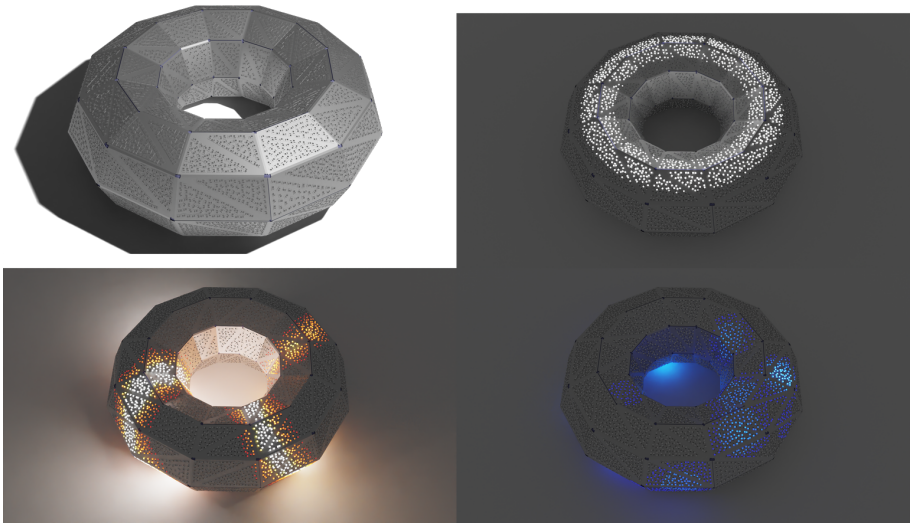


Figure 3.37: Virtual results of our *sqtorus* model.

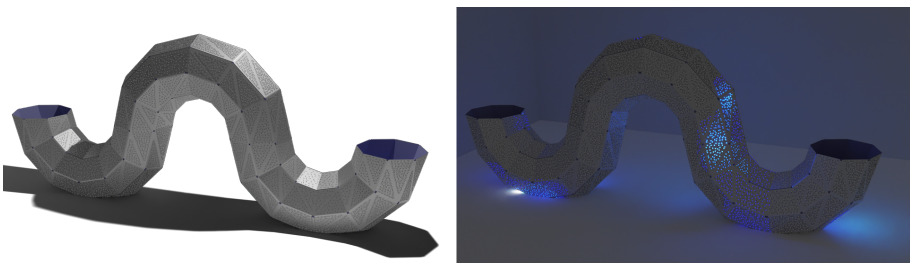


Figure 3.38: Virtual results of a squiggly model composed of 6 *sqtorus* parts.

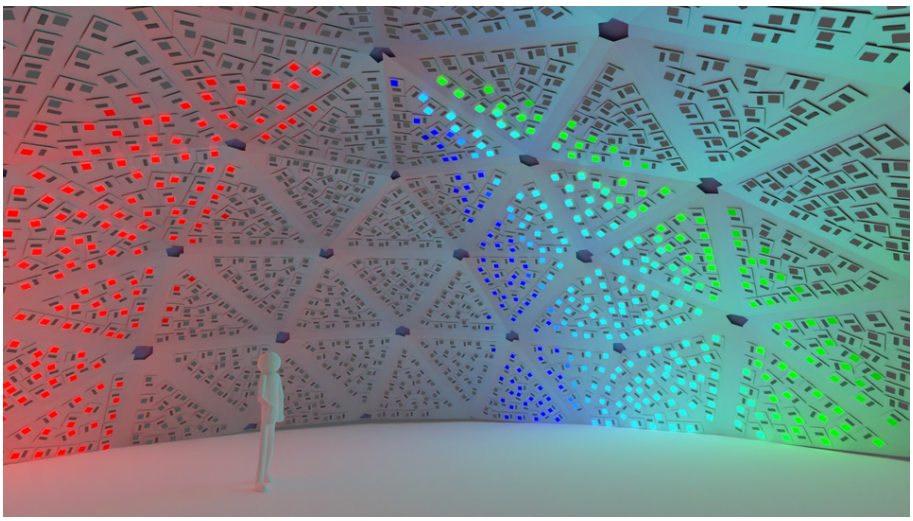
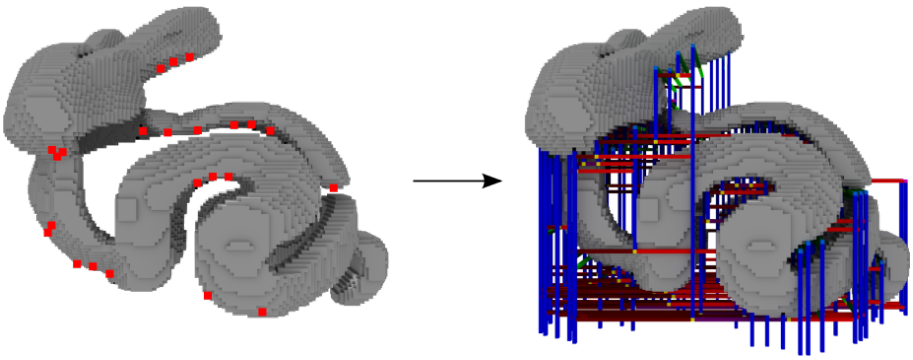


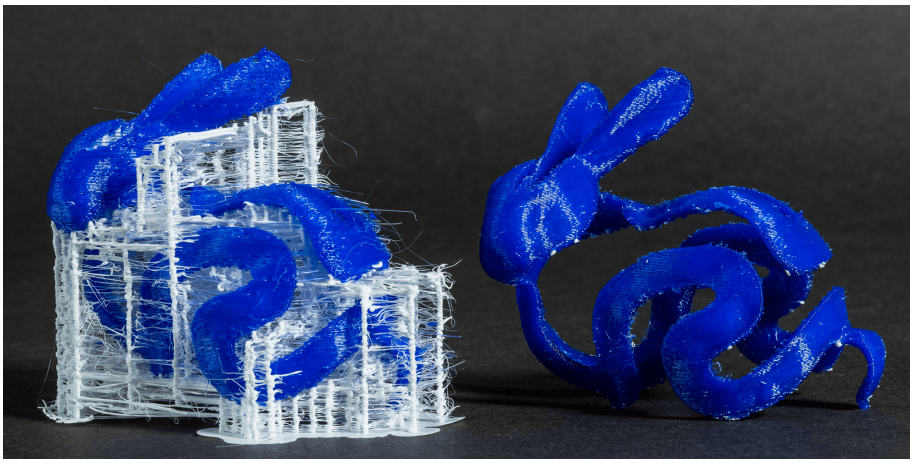
Figure 3.39: Futuristic example: person standing under a *dome*.

Chapter 4

Procedural generation of 3D printing supports



(a) Support generation for the *Bunny peel* model.



(b) Printed *Bunny peel* before and after support removal.

Figure 4.1: Our algorithm generates bridges-and-pillars supports in a voxel grid surrounding an object, with complexity independent from the number of points to support. The support graph is converted into geometry, sliced and 3D printed. Our supports carefully avoid the part, leaving no unnecessary scars (model from [SU14]). Thin white threads are due to optimizing away retraction moves, but none touch the part.

This chapter tackles the problem of support generation for 3D printing from an original point of view. Additive manufacturing requires support structures to fabricate parts with overhangs. Here, we revisit a known support structure based on bridges-and-pillars [DHL14] (see Figure 4.1). These support structures are made of vertical pillars supporting horizontal bridges. Their scaffolding structure makes them stable and reliable to print. However, the original algorithm's heuristic search does not scale well and is prone to produce contacts with the parts, leaving scars after removal.

Our guiding principle is to cast the problem as a constrained layout problem. From this point of view, the problem is amenable to techniques such as Model Synthesis (MS) and Wave Function Collapse (WFC). Originally, these techniques are popular for procedural content generation, where they are useful to synthesize 2D textures or 3D models replicating neighborhoods from a provided example. To this end, they operate in a grid where cells are assigned a label representing some kind of structure. These labels are related to each other through adjacency or neighborhood constraints, i.e. rules stating what label

can be adjacent to or near each other. These are the essential elements of a layout problem as defined in Chapter 2, i.e. a space, objects and a set of constraints. Unfortunately, MS and WFC often run into inconsistencies, requiring the algorithm to backtrack or restart from scratch. By carefully co-designing a set of constraints and specializing the method for support generation, we manage to always generate a support structure without trial and error. We also particularly focus on avoiding unnecessary contacts with the part as much as possible, to avoid damage to the part during support removal.

This work was carried out during the first year of my PhD through the multiple COVID-19 lockdowns. It was a joint work with Samuel Hornus, Salim Perchy and my advisor Sylvain Lefebvre, with a lot of technical assistance from Pierre-Alexandre Hugron and Pierre Bedell. It resulted in a publication [Fre+22] in the short paper track of the Eurographics 2022 conference. This approach is also implemented in a beta version of the MFX research team's slicer *IceSL* [INR13].

4.1 Introduction

3D printing allows the creation of tangible objects from a 3D model, depositing material layer after layer to form a physical counterpart of the digital model. Several printing technologies, in particular Fused Deposition Modeling (FDM), can only stack a new layer on top of an already fabricated surface. Printing overhanging features thus requires disposable support structures, cleaned after fabrication. They must be easy to remove and most importantly, touch the object only where strictly necessary to avoid scarring. There is a vast body of work in the support structure literature with techniques attempting to strike a delicate balance between reducing support material usage, printing reliably, and minimally impacting part quality.

In this work we revisit the generation algorithm of the bridges-and-pillars support structures of [DHL14], which is available in the *IceSL* [INR13] slicing software. This technique relies on the bridging capability of FDM printers to produce a scaffolding geometry that is stable and prints reliably. While effective, the 'next bridge' heuristic search proposed in [DHL14] suffers two main drawbacks. First, in cramped geometries the algorithm struggles to detect collision-free bridges. Most notably, it cannot detect narrow passages to go through. Besides, collision checking is expensive and not implemented in the publicly available version in the *IceSL* [INR13] software. Second, it scales with the fourth power of the number of points to support, making it impractical for large models. We propose a novel algorithm addressing the aforementioned drawbacks. We build upon the *example-based model synthesis* technique [Mer07; Gum16] that generates geometry from a given example. Model synthesis draws inspiration from general *constraint satisfaction problems* [KS17] algorithms such as AC3 [Rus+22].

4.2 Related work

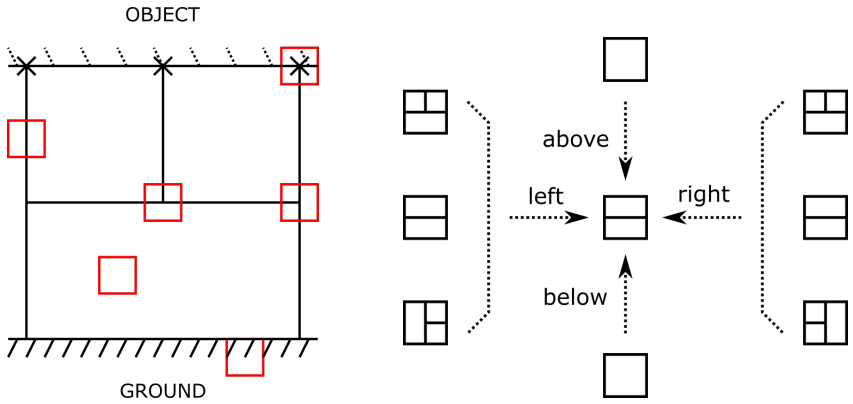
Support structures are necessary with most additive manufacturing technologies, typically to support regions in overhang with respect to the build direction. The supports we consider here were designed for filament extrusion. As we

specifically focus on the approach of [DHL14], we only provide a brief overview of the field for context. We compare our results to the original in Section 4.9. For more in-depth reviews on support techniques please refer to [Liv+17; Liu+18; JXS18].

Support generation starts with choosing the orientation in which the object will be printed. Choosing a proper orientation can significantly reduce the amount of supports needed. Designers often choose a specific orientation for aesthetic or structural purposes, e.g. avoiding support removal scars on specific regions of the object or ensuring that the print is stable throughout printing. These considerations are highly important, but orthogonal to the problem at hand. Given an object under a specific orientation, a first problem in conceiving a support technique is to identify which points should be supported. This can be done from the triangle mesh [AAD98], with boolean differences between subsequent slices [All+88; CJR95; Hua+09a], or by considering the deposition trajectories directly [DHL14]. This step outputs a set of surfaces or points to be supported, and is also orthogonal to the problem of support generation. Here we start from a list of points to be supported, and make no assumption on how it was obtained.

After the points to support are determined a support structure is generated. Early approaches extrude a large volume beneath the overhanghing surfaces. This is then printed with a weak infill pattern or soluble materials. While this prints very reliably, the volume can be quite large leading to an increased print time, an increased material usage and a difficult cleanup. As a consequence several works investigate how to limit material usage, typically reducing the support's geometric complexity far away from the object [Hei11; Hua+09b]. A key development in recent support technologies are the optimization of branching tree structures [SU14; VGB14] which were pioneered in the *MeshMixer* software [Sch13]. This inspired subsequent research, including the technique we are focusing on.

4.3 Model synthesis



(a) Side view of a support, red squares represent different labels.

(b) Side view, possible adjacent labels of a bridge element label.

Figure 4.2: Label geometries and adjacency constraints.

This section continues the preliminary introduction to the basic MS algorithm given in Section 2.3.2.2. This allows to pinpoint the specific aspects of the algorithm that we modify for support generation. MS works in a discrete 3D voxel grid, where each voxel contains a set of possible labels. Labels give geometric meaning to the voxel they are attached to, and serve to represent different parts of the bridges-and-pillars support structure. For example, some labels represent parts of pillar (vertical) elements of the structure, while others represent parts of bridges (horizontal). The possible set of output support structures is encoded by adjacency constraints (see Figure 4.2) specifying what labels can be adjacent to each other in a given direction, e.g. pillar labels can be adjacent vertically, but not horizontally, and conversely for bridges.

Input : `obj`: object to print and points needing support
L, AC: set of labels, adjacency constraints
Data: $M(v) \in L$: label at voxel v; if unassigned, $M(v) = \perp$
$A(v) \subseteq L$: labels allowed for voxel v
Function `synthesize(obj)`
  `initModel(obj, M)`
  $U \leftarrow$ set of unassigned voxels in M
  while U *is not empty* **do**
    choose $v \in U$, choose $l \in A(v)$
    $M(v) \leftarrow l$
    `// update allowed labels`
    `propagateConstraints(A, v, AC)`
  end

Algorithm 1: Model synthesis algorithm

The algorithm itself consists of a few steps outlined in Algorithm 1, the most essential being *constraint propagation*. Constraint propagation is based on the concept of *arc consistency* [Rus+22; Lec09], borrowed from Constraint Satisfaction Problem (CSP). In the CSP framework, voxels are *variables* taking values in a *domain*, here the set of labels. *Constraints* restrict the allowable combinations of values assigned to related variables. These correspond to our adjacency constraints, specifying what combinations of labels are allowed in adjacent voxels. A variable is *arc-consistent* with respect to another if for every possible value of the former, some value for the latter satisfies the constraints. The problem is arc-consistent if all variables are arc-consistent with respect to all other variables. This can be used to prune the possible values for a variable. Indeed, arc-consistency is a *necessary condition* for a solvable CSP. By removing labels that prevent the model from being arc-consistent, no useful information is lost, helping make progress in the algorithm.

Note that arc-consistency is not sufficient for solvability, unfortunate label assignments can lead to an inconsistent state further down the line, where no label is longer possible for some voxels. More details on this are given in Paul Merrell's thesis [Mer09], specifically Theorem 3.3.2 and Section 3.3.4. Stronger notions of consistency such as k-consistency need to be considered to deterministically obtain a solution to a general CSP, but require exponential time and space to be computed [Rus+22]. In our method, constraint propagation uses the `AC-3` algorithm [Mac77], with a worst-case complexity of $O(nk^3)$, n

being the total number of voxels and k the total number of labels. The worst-case complexity is rarely attained after initialization since the updates to the model become significantly more localized.

With constraint propagation explained, the rest of the base algorithm is relatively straightforward. However, an important part of our work is determining the set of rules. We are not creating a generic solver, but instead co-designing a solver and the static set of rules it operates within.

We use *model* to represent the working state of the algorithm. First, the model is initialized based on the object to print, represented by the `initModel` function. This initialization assigns the *object* label to all voxels within the object and the *anchor* label to the voxels that need to be supported for printing. The rest of voxels are unassigned, i.e. all possible labels are allowed. Then a constraint propagation step is done to ensure arc-consistency. The main loop of the algorithm boils down to choosing an unassigned voxel, giving it a label among its possibles, and propagating the constraints to obtain an arc-consistent model once again. This stops once all voxels are assigned or an inconsistent state is reached, i.e. there is a voxel for which all labels violate some constraint. Inconsistencies are typically handled in WFC restarting from scratch or backtracking, and in MS by starting with a simple initial solution and regenerating the final structure in smaller blocks. The algorithm has an overall worst-case complexity of $O(n^2k^3)$, with the notation defined above.

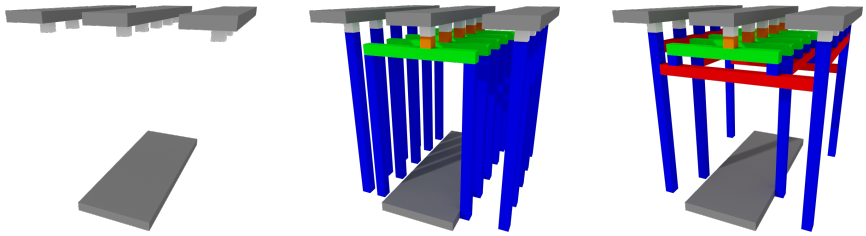
Our main contribution is the design of a specially crafted set of constraints and a custom next voxel and label selection, that together synthesize an initial structure minimizing contact with the part *without trial-and-error*. This is in stark contrast to MS and WFC which often encounter inconsistencies where no label is allowed for a voxel, and either backtrack or have to restart. The set of constraints is encoded as a voxel-based *template*. The main elements from MS we modify are highlighted in blue in Algorithm 1. These are the constraints, which unassigned voxel is processed next, and how its label is chosen based on adjacency constraints.

The rest of the chapter is structured as follows. First, Section 4.4 gives an overall perspective of the algorithm by explaining in detail the role of the two phases of the algorithm. Then, Section 4.5 goes in-depth into the voxel choice and label assignment heuristics crucial to inconsistency avoidance. Next, once an overall view of the algorithm has been established, Section 4.6 enters into the technical details of our template design, revealing how it was built in practice, and what the design challenges are. Following in that vein, Section 4.7 lists the label priority rules and the rationale behind them. Section 4.8 exposes the limitations of the example-based approach to adjacency constraint design, specifically when it comes to allowing or disallowing structure contiguity. Finally, the chapter concludes with a gallery of results in Section 4.9 and a critical discussion of the approach.

4.4 Two-phase algorithm

This section introduces the two-phase structure of the algorithm, explaining the role of each phase and the differences between the two from a general point of view. The technical details are explained later in Section 4.6, where I explain

how the specific goals of each phase are achieved in practice through template design and custom label choice rules.



(a) Voxelized object. (b) Phase one. (c) Phase two.

Figure 4.3: Illustration of our two-phase approach. Pillars in blue, tables in green, bridges in red, object in gray, anchors in white. The bottom object works as an obstacle for the supports to avoid.

Our algorithm operates in two phases, illustrated in Figure 4.3 on a toy example. Both phases follow the synthesis process outlined in Algorithm 1, with different sets of labels and adjacency constraints. The first one builds tables, i.e. bridges resting atop two pillars at its ends, and isolated pillars, i.e. pillars going straight from an anchor to the ground. Its goal is to generate the simplest type of structure to support every anchor while *minimizing* the number of pillars standing on the object. The second phase uses the result of the first phase as a starting point and optimizes the generated structure by building bridges between isolated pillars, thus reducing its overall length. The type of structures generated by the algorithm is described by two templates, one for each phase. These models define the adjacency constraints: if two labels are not adjacent in the example, they cannot be adjacent in the output. Both templates are designed in such a way to allow bridge-and-pillar support structures to be generated. The two phases use different — albeit strongly related — templates.

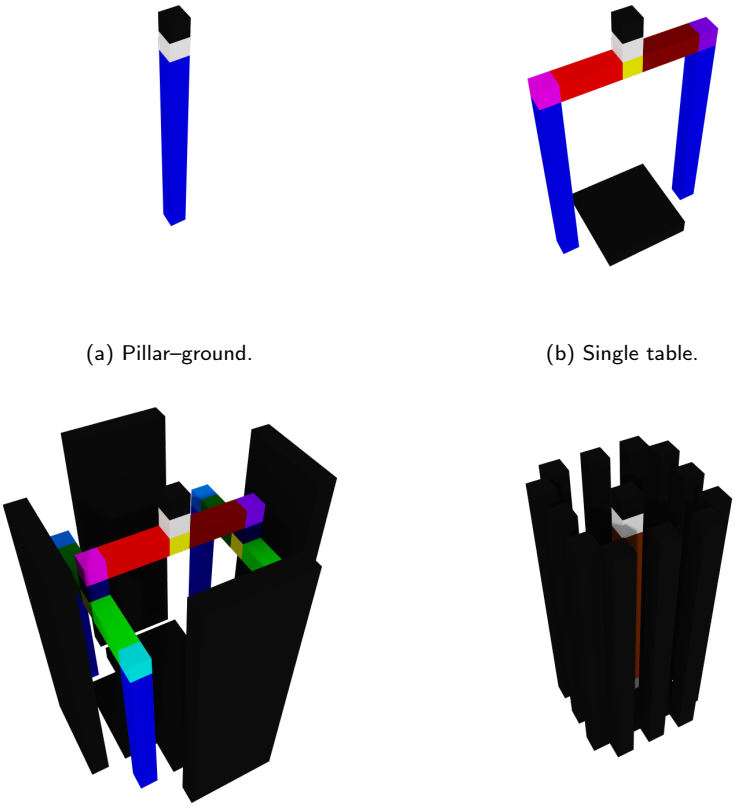
4.4.1 First phase: object avoidance

Anchors (i.e. points needing support) with an unobstructed vertical line of sight to the ground can be easily supported with a single vertical pillar. Of course many anchors are not visible in this way, since the object itself can obstruct the line of sight. Our template leads to the synthesis of different structures avoiding the object, illustrated in Figure 4.4 for simple scenes.

These are, by decreasing priority:

(a) an isolated pillar standing on ground;
(b) a single table;
(c) a double table;
(d) an isolated pillar standing on the object.

The template for the first phase is shown in Figure 4.5. Different colors correspond to different labels and different pieces of the support geometry (see also Figure 4.2). There are in fact more labels than there are visible colors due to the limited palette. The principles behind the design of this template are explored further in Section 4.6.2. The hierarchical nature of the structures, as well as the use of different labels at each of their levels causes many labels to be removed during each constraint propagation step. Due to the design of the template, the initial constraint propagation step after the object and anchors



(a) Pillar–ground.

(b) Single table.

(c) Double table.

(d) Pillar–object.

Figure 4.4: Structures (color) generated by the first phase from an anchor (white) around the object (black). In each case, the object is artificially constructed to force the algorithm to generate the showcased structure.

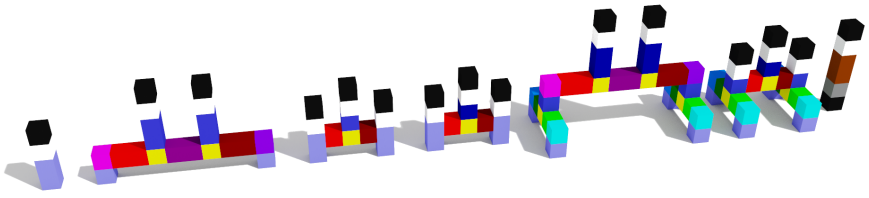


Figure 4.5: First phase template. Different voxel colors represent different labels, but a single color may represent multiple labels.

are loaded (`initModel` in Algorithm 1) is actually sufficient to determine how to support each anchor. Indeed, after propagation, the allowed labels below an anchor already indicate the simplest structures that can support it. This property is specific to this template.

This is mainly a consequence of using different sets of labels for the four first phase structures in the template. Constraint propagation then essentially checks

the feasibility of all four types independently but simultaneously. Reusing the same labels for all structures would make the adjacency constraints much more permissive, losing the desired property. Thus, choosing a label below an anchor fully determines the structure supporting it, the rest of it having been already determined by constraint propagation.

4.4.2 Second phase: support optimization

The first phase is designed to generate an initial structure avoiding contact with the object, and does not attempt to reduce the support size. In particular, it always generates isolated pillars under unobstructed anchors. Phase two improves the structure by connecting multiple aligned pillars with bridges. These always support at least one pillar, making them more efficient than isolated pillars in most cases. However, if the bridge is longer than its height multiplied by the number of supported pillars, this is results in a larger support than the isolated pillars. We do not explicitly check for performance considerations. Additionally, bridges make supports more stable by connecting previously disconnected sections of the structure. This is done by removing all isolated pillars from the result, and then regenerating the structure with the synthesis algorithm, using bridges wherever possible. The constraint propagation step after isolated pillar removal immediately regenerates some removed pillars: those that cannot be connected with bridges. No new first phase structures are generated by the second phase: the relevant labels are disallowed during synthesis. Compared to the first phase, the template includes additional structures to make these improvements possible (see Section 4.6.3).

4.5 Choice and assignment heuristics

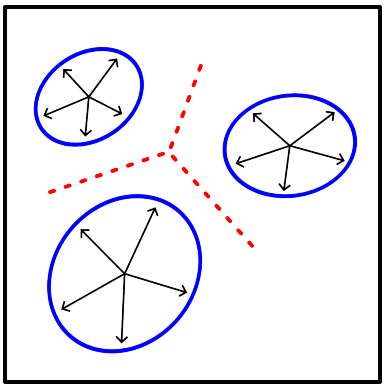


Figure 4.6: Schematic 2D illustration of random voxel choice heuristic. Multiple constraint propagation fronts in blue meet along the dotted red lines and are likely to create inconsistencies.

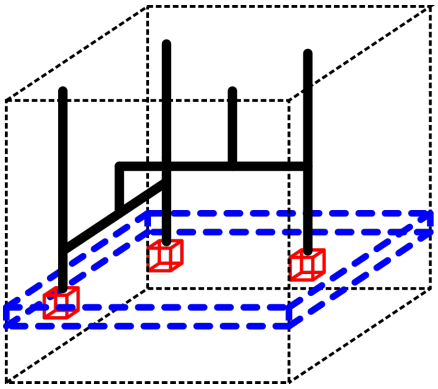


Figure 4.7: Support propagation heuristic. Already generated support in black, current layer being generated in blue, set of current layer seeds in red. The current layer is generated starting from the seeds and propagating outward. Once generated, the layer immediately below is next.

The order in which unassigned voxels are chosen is crucial to ensure that the algorithm avoids inconsistencies. In an incomplete model, let us choose a

voxel far from the already assigned voxels. That voxel would spawn a second structure in that region of the model. If done multiple times, we create many local structures that will all have to meet at some point — and are very likely to disagree. This is illustrated schematically in Figure 4.6.

To solve this, we use a context-dependent order. The algorithm operates slice by slice, from top to bottom (see Figure 4.7). Within a slice (in blue), voxels whose 'upstairs neighbors' are part of the support structure are tagged as seeds (in red). Voxels in the current slice are selected by increasing distance to the set of seeds. Once all voxels in a slice have been assigned, the algorithm goes to the next one until the bottom of the model is reached. By doing this, we always select unassigned voxels that are adjacent to already assigned voxels. This drastically decreases the probability of making a choice that will lead to an inconsistency further down the line.

Label choice for an unassigned voxel is done according to a set of priority rules. The possible labels can be ordered in order of decreasing priority, and a random label among the highest priority ones is chosen. Understanding the priority rules and their effect requires a more in-depth explanation of the construction of the templates and the adjacency rules. The controlled randomness of label choice allows the generation of similar but slightly different supports, some more material-efficient than others. By running multiple synthesis processes in parallel, we can choose any of the resulting structures depending on user-defined criteria.

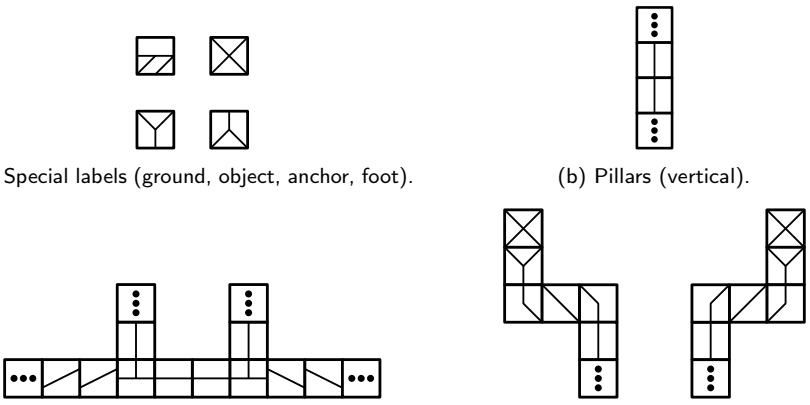
4.6 Template design

With the principle behind the method explained, it is now easier to understand the process and rationale behind the design of the template models and priority rules. The equivalent to our templates in Merrell's thesis [Mer09] were originally named *exemplars*, since they served as a model for the desired output of the method. I decided to rename them here to better convey the fact that ours are fixed, and serve to encode the specific structure of our supports. For both phases, the templates consist of a two identical sets of two-dimensional structures, one along the x axis and the other along the y axis. Labels that are part of a horizontal bar have an x and y variant, while labels that are part of pillars have a single variant. All illustrations in this section will be in two dimensions. Recall that the adjacency constraints are extracted simply by allowing any combination of labels occurring in a given direction if it occurs in the template. Specific limitations of the approach of using a template to define the set of adjacency constraints are exposed later in Section 4.8.

4.6.1 Building blocks

In this section we explain how the pillars and the bars used in our supports are built in the template. All of the structures are shown in Figure 4.8.

Special labels. In our template we use a set of special labels that have a particular meaning. From left to right and bottom to top, they are the *ground* label, the *object* label, the *anchor* label and the *foot* label. The ground label represents the printing bed, the object label is assigned during model



(a) Special labels (ground, object, anchor, foot).

(b) Pillars (vertical).

(c) Bars (horizontal).

(d) Offsets.

Figure 4.8: Main building blocks constituting the supports.

initialization (`initModel` in Algorithm 1) to voxels within the object, the anchor label represent object points needing support, and the foot label represents a point of contact between the object and a pillar standing on the object. Object, ground and anchor labels cannot be assigned during synthesis, instead they are assigned at initialization based on the input model. Foot labels are generated during the first phase exclusively. Not pictured here is the *empty* label, which is also generated by the algorithm to denote the absence of supports in a voxel.

Pillars. Vertical pillars of arbitrary height between the ground and an anchor can be defined very concisely. By vertically stacking two voxels with the same label, we allow the label to be adjacent to itself in the upwards and downwards direction. This authorizes vertical pillars of any height in the output. By starting and ending the pillar with the labels for ground and anchor respectively, we force pillars in the output to only happen between those labels. Since there are multiple labels that can start or end a pillar, we denote them by an ellipsis here. These are also allowed to start with a foot label which is allowed to make contact with the object. This possibility is only allowed to avoid inconsistencies in the case that the support structure cannot totally avoid the object, but these types of contact are actively discouraged and rare (see Section 4.9).

Bars. The horizontal bars in our model are slightly more complicated than pillars. They are of arbitrary length, but they also need to be able to support an arbitrary number of pillars. Bars consist of three sections: a beginning, a repeating middle section and an end. Pillars attach to the bar atop junction labels separating the different sections of the bar. The middle section can be repeated any number of times, since a junction label can have a beginning, end or middle section label at both of its sides. Each section is of arbitrary length for the same reason pillars are. The horizontal ellipsis here denote labels that can start or end a bar, and the vertical one represents the rest of the vertical pillar.

Offsets. Anchors cannot be supported by the same structure if they are not aligned in either the x or y direction. To give the algorithm additional freedom,

we allow these offset junctions between anchors and pillars in the output. Also, a given anchor might be possible to support by a simpler structure if it is offset. These are not discussed further in the rest of the chapter, as they do not fundamentally change the principle of the algorithm.

4.6.2 First phase: tables

The goal of the first phase is to find the simplest possible structure to support all anchors. Ideally, isolated pillars suffice, but complex geometries require supports that can avoid the object. To this end, we use *table* structures that can support the anchors while avoiding the object. These are different from bridges in that they rest atop two pillars instead of being subtended between them.

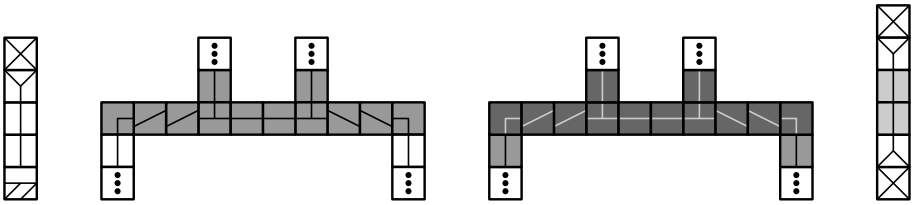


Figure 4.9: First phase template (simplified)

In order of priority, the structures allowed in this pass are straight pillars standing on the ground, single tables, double tables, and straight pillars standing on the object. These structures are represented from left to right in Figure 4.9. Labels that only differ in their fill color have the same function but are distinct. Pillars standing on ground and pillars standing on the object are straightforward. The model contains two different types of tables, one for the first level of table and one for the second level. A first level table stands on the ground and can either support a second level table or connect to an anchor on top. A second level table always stands on a first level table and connects to an anchor on top. Using distinct labels for each level is necessary to determine the simplest support structure for an anchor from constraint propagation as explained in Section 4.4.1.

Higher stacks of tables could be defined by creating new labels, but we observed that three levels or more were rarely needed to support any anchor. Additionally, the complexity of the algorithm depends on the number of labels, thus making it more expensive to allow more stacks. Tables could also be defined with a single type of label, instead of using distinct labels for each level. This would make it so they could stack on top of each other an arbitrary number of times, but would make constraint propagation significantly less effective, and would hugely expand the solution space.

Not pictured in this figure are *hybrid* structures that allow to horizontally connect a table to existing pillars instead of spawning new ones to support it. This simple addition to the allowed structures requires careful consideration and the addition of a priority rule to ensure that synthesis avois inconsistencies. This is discussed further in Section 4.7.

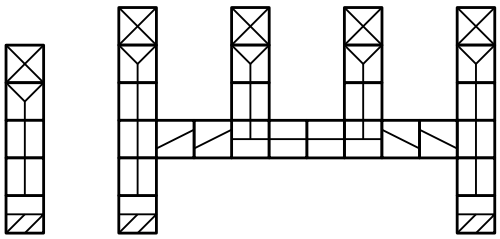


Figure 4.10: Second phase template (simplified), first phase structures not shown.

4.6.3 Second phase: bridges

The second phase regenerates the isolated pillars created in the first phase while allowing bridges to be built between them. This is done by simply extending the first phase template with the structures in Figure 4.10. Note that, contrary to tables, bridges can stack indefinitely instead of being limited to two levels. This is due to the same pillar labels being used both below and above the bridge in the template.

4.7 Priority rules

Previous methods choose labels either randomly or according to a probability distribution so that the distribution of labels in the output model matches the one in the template. Instead, we observe that the criteria used to assign labels to unassigned voxels can be adapted to our problem to minimize the failure rate, while giving us control on the properties of the final structure. This section explains the priority list system used in the algorithm.

We choose labels according to a priority list, i.e. a list of disjoint priority rules each defined by a set of labels. For an unassigned voxel, if no rule contains any possible label for that voxel, then the label is chosen randomly among the possible ones for the voxel. Otherwise, the first such rule is taken and the label is chosen randomly among the priority rule labels that are possible for the voxel.

The rules for the first phase are the following:

(1) empty label

(2) **if below anchor**, pillar on ground

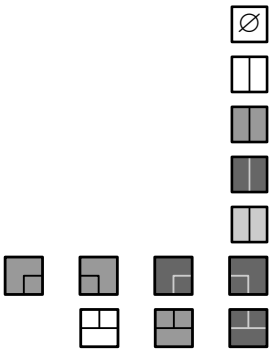
(3) **if below anchor**, single table

(4) **if below anchor**, double table

(5) **if below anchor**, pillar on object

(6) end of table

(7) bar junction



Rules (2) through (5) are only used on unassigned voxels right below anchors. They ensure that the simplest structure is chosen to support these anchors. Rule (6) ends tables as soon as possible to avoid having unnecessarily long tables, making them just long enough to avoid the object instead. Rule (7) starts horizontal bars to support a pillar wherever possible. This favors bars supporting multiple pillars and ensures that thye are as high up as possible.

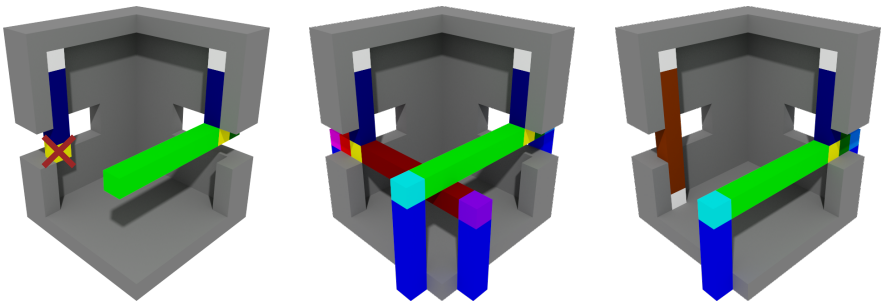
The second phase only uses rules (1) and (7) (only for bridge junctions), since the structures involved in the other rules have already been generated.

The first rule requires more attention to properly understand. It essentially states that if the current voxel can be empty, then it is made empty. This rule closely works with our assignment heuristic (see Section 4.5) to make it so labels locally decide the directions the structure will grow in. When an unassigned voxel is chosen, it is adjacent to at least one already assigned voxel due to our assignment order. If the voxel we chose is along the direction in which the structure needs to grow, then the empty label will not be possible for that voxel. If not, then the voxel it is not part of the structure so it can be empty. This also explains why hybrid structures will not appear except where strictly needed, i.e. they are always generated from a pillar-bridge junction and not from pillars. In particular, a case made impossible by the combination of assignment heuristic and rule (1) is a structure 'spawning' others parallel to it. This would create additional constraint propagation fronts, leading to possible inconsistencies (see Figure 4.6) Here, all structures must originate from an anchor, thus avoiding inconsistencies.

It is important to note that priority rules are applied only during synthesis, and do not play any role in constraint propagation whatsoever. Excessively permissive adjacency constraints cannot be patched *a posteriori* with priority rules.

4.8 Structure contiguity

Using a template to define the adjacency constraints has some limitations. The empty label has a special role when constructing a template in a voxel editor, since it is the default value for a voxel. The only way to forbid an empty label beside another label is to fill that side with a non-empty voxel for every single instance of that label in the template. For this reason, templates work well to define labels that can be adjacent to a small number of other labels. If we want a label to be adjacent to a large number of labels, we need to build every possible combination of labels by hand in the template.



(a) Forbidding contiguity. (b) Allowing contiguity. (c) Even-odd setup.

Figure 4.11: Comparison of different contiguity setups in a highly constained model. Left: The algorithm fails to generate a structure, no labels are possible at the crossed-out voxel. Middle: The algorithm generates two tables with one bar on top of another. Right: The algorithm generates a single table and a pillar on the object, since bars can only exist at an even z coordinate.

In particular, this difficulty arises when considering the contiguity of different

Table 4.1: Performance for various models, 0.5mm voxels. Computed on an AMD Ryzen 5600X with 16GB of DDR4 RAM. The last column (#feet) refers to the number of newly created contact points between the support structure and the object.

Models	Grid size	#voxels	Time (s)	#anchors	#feet
Gymnast	$81 \times 28 \times 99$	224 532	1	37	2
Hilbert cube	$70 \times 70 \times 65$	318 500	2	347	0
Knot	$96 \times 101 \times 72$	698 112	4	199	33
Bunny peel	$119 \times 92 \times 104$	1 138 592	9	235	1
Minotaur	$126 \times 94 \times 203$	2 404 332	19	392	16
Enterprise	$318 \times 149 \times 72$	3 411 504	28	1475	0
Fox	$137 \times 166 \times 218$	4 957 756	34	73	0
Thigh left	$138 \times 390 \times 167$	9 987 940	78	3498	43
Cellular	$294 \times 286 \times 248$	20 852 832	185	2369	33

parts of the structure, i.e. unrelated parts of the support being adjacent with each other. We will use *contiguity* to describe this situation to avoid confusion with *adjacency* constraints. Parts of the support should be able to be contiguous to other parts of itself so they do not hinder each other's growth in cramped spaces. Without contiguity, some of these cases can lead to inconsistencies, which we want to avoid as much as possible. Indeed, without it we no longer have enough information to determine the simplest structure that can support an anchor just by looking at the possible labels below the anchor (see Figure 4.11a). Allowing contiguity to the template would require adding an enormous number of label pairs to it. We found two alternative approaches to this problem.

First, contiguity can be added programmatically to the set of constraints. Given any two non-empty labels that can each be adjacent to the empty label in opposite directions (up-down, left-right, front-back), we simply add the label pair to the allowed combinations in that direction. This provides the most freedom in the output (see Figure 4.11b), the downside being that our set of constraints becomes significantly less restrictive, in that most labels can now be adjacent to many labels that were not possible before. The algorithm still works but struggles in finding efficient solutions.

Our second approach is to only construct supports in voxels with even coordinates, enforcing an empty voxel interspace between any two components of the structure. Pillars can only be built in voxels with even x and y coordinates, and bars can only be built in voxels with an even z coordinate (see Figure 4.11c). This effectively halves the resolution for the support structure, thus reducing computation time while keeping the same adjacency constraints. As a downside, some anchors may only able to be supported by pillars standing on the object due to the reduced support resolution. This can be mitigated by increasing the overall voxel resolution to better capture the free space around the object.

We use the second approach in our implementation, since we found that the increase in the number of pillars standing on the object was not substantial, but the gain in computation time and structure efficiency were. Figure 4.11 shows a comparison of the different contiguity setups.

4.9 Results

Unless otherwise specified we use a voxel size of 0.5 mm. We printed these models using PLA filament on low-cost Ender3 and CR10 filament printers.

4.9.1 Implementation

Our algorithm takes a voxel grid as an input and returns a list of segments. The voxels represent the object and the points needing support are explicitly labeled as anchors. The resulting list of segments describes the computed support structure. Our processing pipeline is given a 3D model and outputs GCode for a physical print. It consists of the following steps:

(1) a voxelizer and support point detector;
(2) a support structure generator (our method);
(3) a support geometry creator;
(4) slicing, trajectories and GCode output.

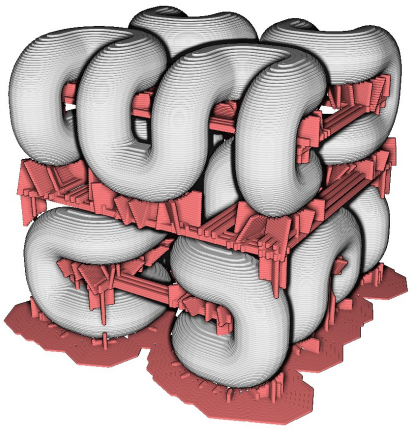
The final step is using a standard slicer, while the other steps are tailored to our method. (1), (3) and (4) are independent from our method and could be done by other means.

4.9.2 Comparison with the original algorithm

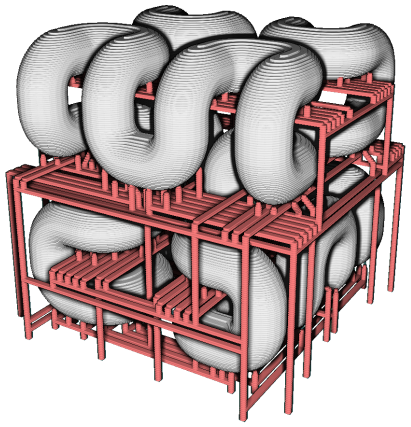
We propose a different algorithm to generate structures similar to [DHL14]. Figure 4.12 compares both methods on the *Hilbert cube*, *Enterprise* and *Sergeant* model. Note how the supports of the original algorithm intersect the object in many places on the *Hilbert cube* and *Sergeant*, while ours avoid the object. The support size, measured using the total filament length for fabrication, is similar for *Hilbert cube*, but is typically larger for our method as seen for the other two models. For a similar number of anchors the execution time of our method is faster both for the *Hilbert cube* and the *Enterprise*. The complexity of our approach is independent from the number of anchors. Instead, it increases proportionally to the square of the number of voxels giving it an advantage on larger, more complex models. However, the original can be faster on small models with few anchors, as showcased by the *Sergeant* model. Note that for this model, our algorithm generates 'precarious' supports for the sword, consisting of very tall tables with extremely long bars. These isolated structures tend to happen when anchors are sparse, preventing support structures from connecting to others, thus becoming less stable. This issue is discussed further at the end of the chapter (see Section 4.10).Collision avoidance in [DHL14] is expensive, as each candidate bridge has to be explicitly checked against the model, and the algorithm still often fails to find collision-free solutions. Due to this, collision avoidance is not even implemented in the publicly available version of [DHL14].

4.9.3 Result gallery

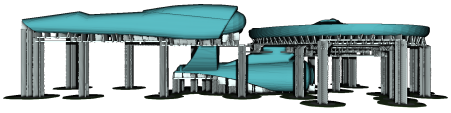
Table 4.1 lists execution times for models with varying voxel grid sizes. Also shown is the number of anchors and how many contact points are created by the algorithm failing to avoid the object. For all models but *Knot*, the

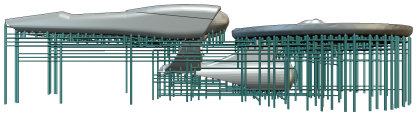


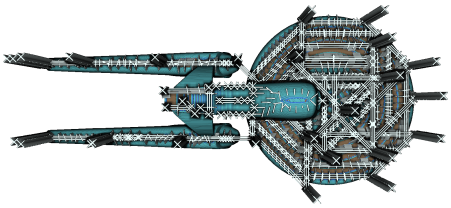
(a) 333 anchors, 12 s, ∼2800 mm of supports.

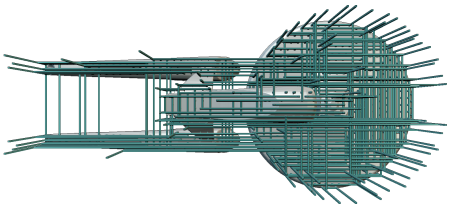


(b) 347 anchors, 2 s, ∼2800 mm of supports.



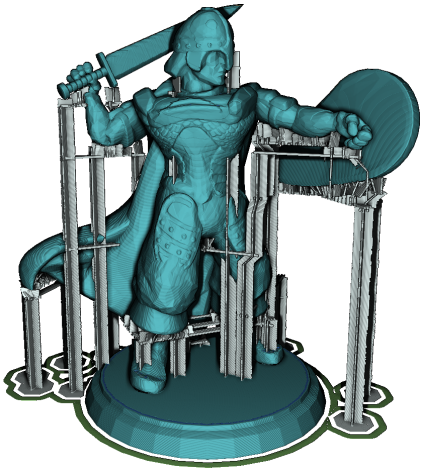




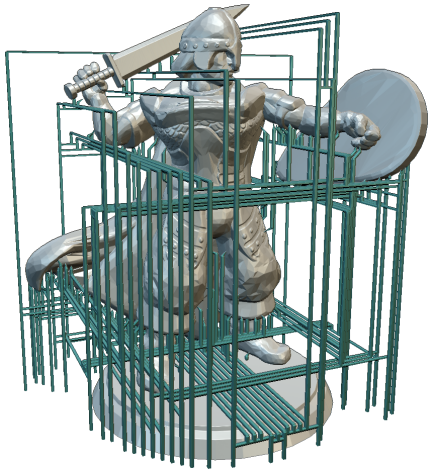


(c) 742 anchors, 97 s, 4268 mm of supports.

(d) 741 anchors, 32 s, 12843 mm of supports.

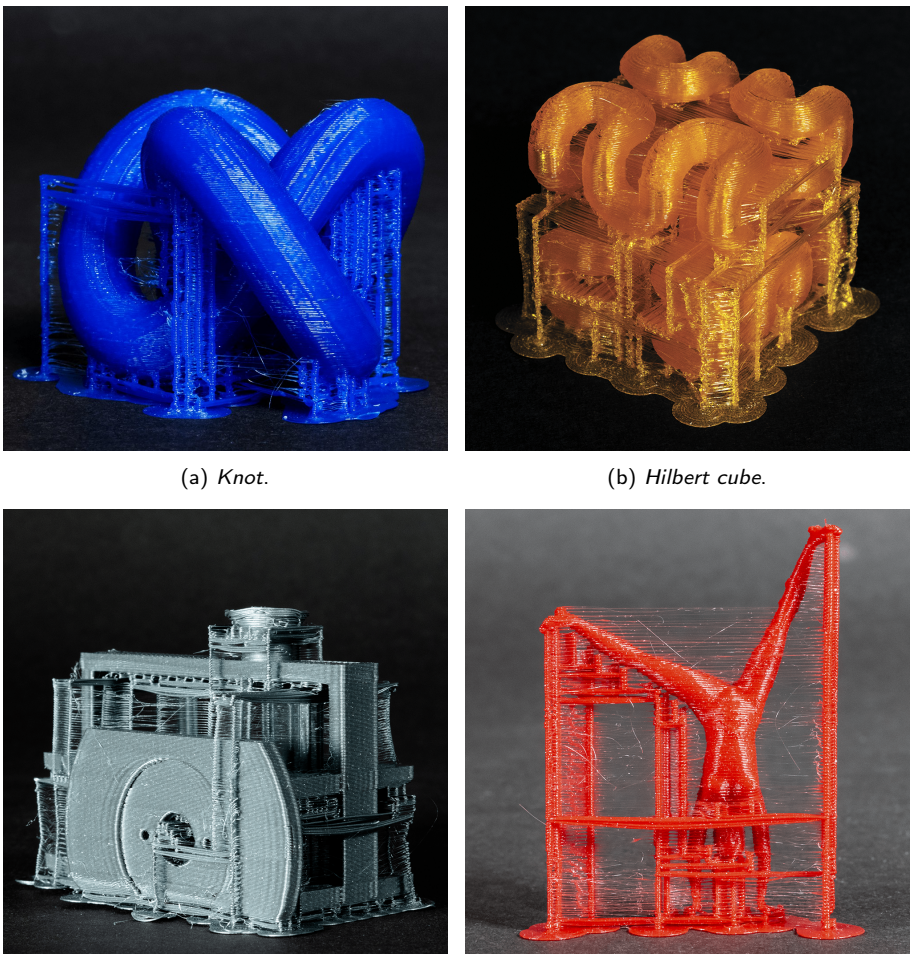


(e) 234 anchors, 8 s, 2139 mm of supports.



(f) 239 anchors, 30 s, 8359 mm of supports.

Figure 4.12: Visual comparison of [DHL14] (left column) to our method (right column) on the *Hilbert cube* model (top row), the *Enterprise* model (middle row) and the *Sergeant* model (bottom row).



(a) *Knot*. (b) *Hilbert cube*.

(c) *Servo support*. (d) *Gymnast*.

Figure 4.13: Models printed with our technique.

number of created contact points is below 6% of the total number. *Knot* has a cramped geometry and the three-fold rotational symmetry makes it difficult to generate axis-aligned supports that avoid the object. In Figure 4.13, note how the supports avoid touching the part. For instance, despite the intricate shape of the *Hilbert cube* (yellow), no pillars are contacting downwards with the print. The same is true of the *Servo support* (gray), where multiple horizontal bridges can be seen going through the lateral hole. This avoidance comes at no extra cost, contrary to the previous algorithm. In all these results a significant amount of stringing can be seen. This is due to the way our slicer optimizes away filament retraction between support pillars. However, none of these thin plastic threads actually connect to the part, making cleaning very easy. Figure 4.1 shows the *Bunny peel* model after support removal, where two different materials were used for the part and the supports. Note how despite using very contrasted white and blue filaments, there are no significant white smears on the object surface outside of the downwards support anchors.

4.9.4 Robustness

4.9.4.1 In the wild

We processed a batch of 900 models extracted from the *Thingi10K* [ZJ16] database with voxel size 0.5mm. The algorithm successfully generated a support structure for every model in the dataset.

4.9.4.2 Azimuth angle

A downside of using a voxel-based method is that our supports can only go in the x and y directions. Diagonal horizontal bars cannot be easily encoded in our template. For this reason, the orientation of the part with respect to the z axis has an impact on the total length of the supports and the number of pillars standing on the object. We measured the impact of the orientation by generating supports for the same part at rotation angles around z between 0 and 90 degrees with a step of 5 degrees.

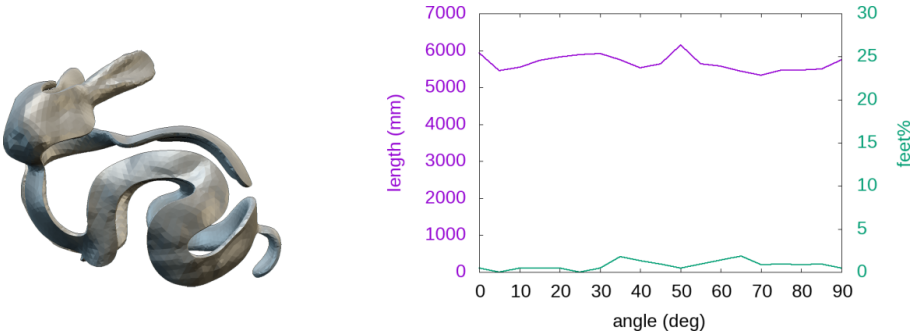
We see in the resulting plots shown in Figure 4.14 that on some objects such as *Servo support*, orientation has a dramatic impact on the length of the support and the number of pillars standing on the object. The best and worst cases for this specific model are shown in Figure 4.15. Also, there is often a specific angle that minimizes both of these quantities. The algorithm finds efficient solutions when the gaps in the object are aligned with the x or y axes, and struggles for other orientations, since it cannot create bars through these gaps.

4.10 Discussion and conclusion

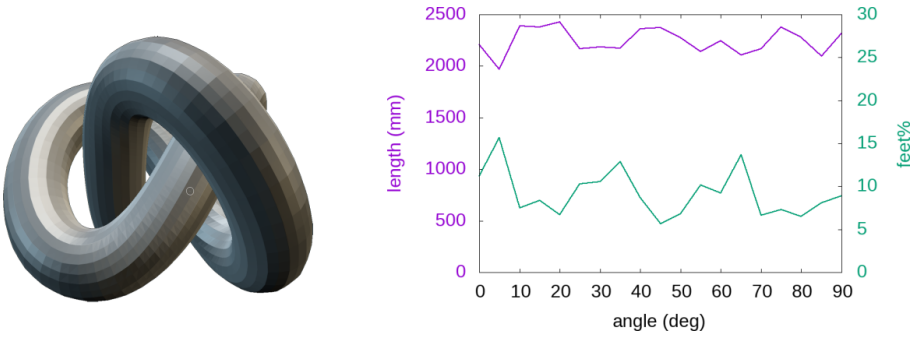
This section discusses the limitations, specificities of our approach, explores future work and concludes the chapter with a summary of the contribution.

Infallibility. An important future work is to prove theoretically that our approach cannot encouter inconsistencies. Our empirical observation on hundreds of models gives us confidence in that claim, but cannot replace a rigorous proof. A challenging aspect of this proof is considering all possible labels combinations in every direction. Despite the restrictiveness of our set of constraints, this still makes for many possible combinations due to the sheer number of total labels ($\sim$40 for the simplest possible version of the constraints). Additionally, the global effects of constraint propagation are hard to reason about. A proof assistant would probably be a good option for the derivation of this proof.

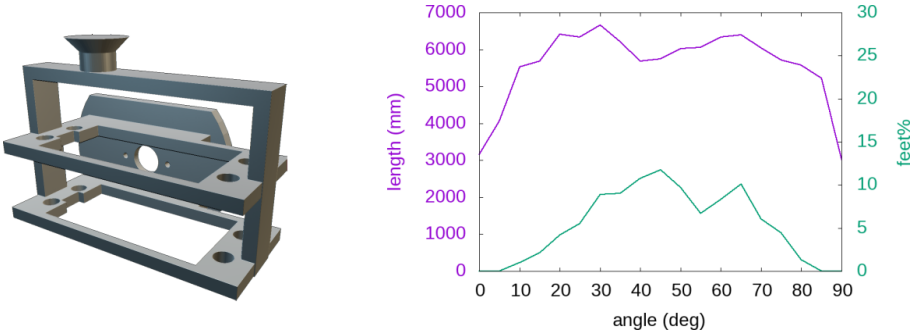
Original model synthesis. In his thesis [Mer09] on MS, Merrell realizes that larger models tend to run into inconsistencies more often. He proposes starting with a trivial solution, and regenerating it in smaller chunks in a scanline manner, thus reducing the likelyhood of any chunk being inconsistent. When generating a chunk, only the voxels within are modified, and have to satisfy the constraints on its border. The drawback of this approach is that structures with features larger than the dimensions of the block in multiple directions cannot be generated. This effectively removes potentially desirable outputs from the attainable results space. Our choice and assignment heuristics and our set of



(a) *Bunny peel*.



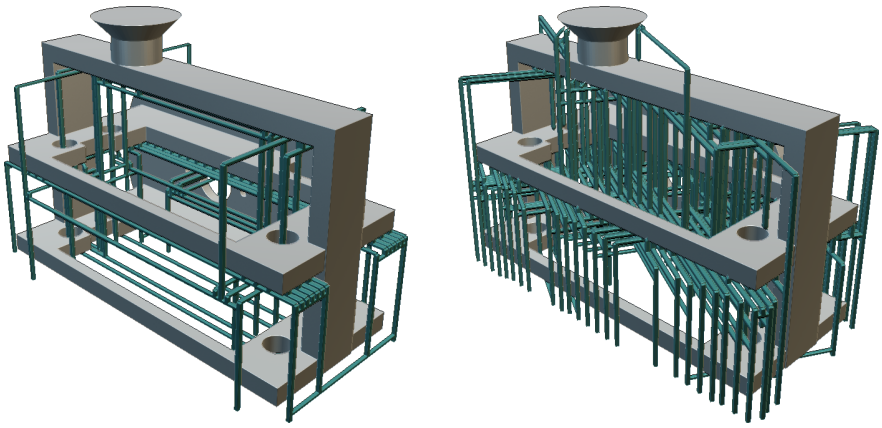
(b) *Knot*.



(c) *Servo support*.

Figure 4.14: Support length (mm) and percentage of anchors supported by a pillar standing on the object (feet%) as a function of the angle of rotation of the part around the z axis in degrees.

priority rules also limits the space of outputs that can be generated by the algorithm. A core difference between these approaches is that these elements were specifically crafted to exclude undesirable results that are often inconsistent. Additionally, we manage to avoid inconsistencies altogether, whereas block-based synthesis simply reduces their likelihood. The main reason explaining our results is that we target very specific types of structures, bridges-and-pillars supports. This allows us to design a coherent system, where every element works together. Originally, MS targets arbitrary sets of adjacency constraints, meaning that such a degree of customization is impossible.



(a) Azimuth: 0 degrees. (b) Azimuth: 30 degrees.

Figure 4.15: Best and worst orientations for *Servo support*.

Bridge length. One particularity of the supports generated by this algorithm is that horizontal bridges can be arbitrarily long (see *Enterprise* in Figure 4.12). Indeed, the construction of bars in the templates have no 'knowledge' of their own length (see Section 4.6.1). In practice, this did not pose a problem when printing our results, as filament printers can robustly print unsupported horizontal bridges between two points [DHL14], and sagging does not pose a significant problem since supports are later removed. Nevertheless, this could pose problems for larger objects or more precarious bars such as the ones in *Sergeant* from Figure 4.12. Two possible solutions come to mind. Firstly, a maximum bar length L can be encoded in the template at the cost of having L distinct bar labels instead of a constant number independently of the length. This is better done programatically than in a voxel editor, since it would require adding many possible label combinations to the constraints. Additionally, execution times would increase even for small maximal lengths, since complexity is cubic in the numbrer of labels in the worst case. A second approach would be to generate the support structure by xy-blocks spanning the whole height of the model, similarly to Merrell's block-based synthesis. This would prevent bridges from spanning multiple blocks without changes to the templates, and would make execution faster, since the blocks could be generated separately and complexity scales with the squared number of voxels in the region. Of course, one should be careful to ensure that inconsistencies are still prevented with this approach.

Voxel grid and diagonal features. Our algorithm works in a 3D voxel grid, but MS and WFC work in any general graph, as they are solvers for CSP with binary constraints. One limitation of working on a 3D orthogonal grid is the difficulty of defining diagonal features, which requires staircase patterns in the templates, requiring many labels and taking up valuable space in the voxel grid. An alternative approach would be to consider diagonal neighbors, allowing to define diagonal features in a straightforward manner. This makes the underlying graph heavily connected, which makes crafting the adjacency constraints harder, since more directions have to be accounted for when constructing the template.

Also, the approach could be recast in a hierarchical and or distorted grid, preoptimized to better capture the model's features.

Post-optimization. Our method outputs a list of vertical and horizontal segments abstractly representing the support structure. These are later transformed into geometric primitives that are then sliced and printed. Before this step, this list of segments could be optimized through linear programming to minimize support length, raise and merge bridges, and many more. By continuously optimizing the positions of segment endpoints, we could 'liberate' them from the voxel grid, thus enabling diagonal bridges and pillars and other types of features that the synthesis algorithm cannot generate.

Conclusion. Our technique generates simple and reliable support structures inspired from [DHL14] that avoid touching the part when possible in a reasonable time. The cost is independent of the object complexity and instead only scales with voxel grid size. In practice a solution is always found if it exists.

Beyond the current results, our approach shows that example-based model synthesis is a viable option for structure synthesis seen as a layout problem, if correctly specialized. Defining a structure based on local properties and synthesizing it while targeting specific global properties is an interesting challenge. We believe that there are other contexts in shape synthesis where this approach is applicable, and hope to see future work inspired by our technique.

Chapter 5

Conclusion

END OF LINE.

MASTER CONTROL PROGRAM

This thesis started with a very broad object of study: *layout problems for generative design and shape modeling*. The underlying goal was to take inspiration from standard techniques in computer graphics and apply them to layout problems linked to computational fabrication. During the span of this thesis, we slowly evolved towards using local descriptions and topological considerations to simplify the layout problems, which eventually became the main focus of this work. Computational fabrication is an inherently multi-disciplinary field. As stated early in the introduction, designing algorithms and systems for fabrication requires knowledge of the underlying techniques and mechanisms. Of course this is common to all forms of abstraction, but is particularly important when building a very high-level interface. This is necessary here to make these technologies as accessible as possible to as many people as possible. This thesis has allowed me to spend time understanding these fields, and contribute to the effort towards making these technologies more accessible, versatile and expressive.

In Chapter 3, we used Printed Circuit Board (PCB) *kerfing* to design an end-to-end system for designing 3D Light-Emitting Diode (LED)-based foldable displays, that we nicknamed *PCBend*. The system takes a suitable input mesh, unfolds it, embeds a fully functioning LED chain within it, and finally outputs fabricable blueprints, ready to be sent to a PCB manufacturing service. After assembly, these displays can be used to create beautiful lighting effects. The LED we use are often used in commercial LED strips or amateur projects, and can be controlled easily. The key elements of this work are the following. First, we follow an atypical approach to circuit design, i.e. not starting from traditional schematics but instead using a local description of the circuit. This in some sense provides a much more flexible representation of the circuit, which can be much more easily adapted to a constrained space, such as our 'hinged' PCB. The level of granularity allowed by our circuits being LED chains is unmatched, but will be coarser for more complex circuits, which will present new challenges that will need addressing. Second is the general idea of PCB kerfing for low-cost flexible electronics. This approach has been occasionally explored by hobbyists [cy319], but does not seem to be particularly widespread.

Traditional flexible or rigid-flex electronics are significantly more expensive and harder to design than regular PCB. We show that PCB kerfing can be successfully used to bend thin complex PCB without damage to the board. While this type of PCB might not be suited for incredibly tight spaces such as the inside of a smartphone, or might not tolerate repeated bending, I believe that it can be particularly fruitful for prototyping and enable many creative applications (see the accompanying video). I personally hope that we will see more people considering this option in the future.

In Chapter 4, we introduce a procedural approach to 3D printing support generation (*ProcSupGen*), inspired from Model Synthesis (MS) and Wave Function Collapse (WFC), two example-based procedural synthesis techniques. We customized these algorithms to generate the bridges-and-pillars supports used in [DHL14], with a complexity independent on the number of points to support, and while avoiding the object as much as possible. For this, we first crafted a custom set of exemplars defining the structures that can be generated. We also developed voxel and label choice heuristics guaranteeing that synthesis succeeds without trial-and-error, which is atypical for MS. This method was integrated into the *IceSL* [INR13] slicer developed by the MFX research team I worked in during this thesis. Most future work on this technique was already outlined in the conclusion of the corresponding chapter. The main takeaway is the flexibility of this type of procedural algorithm, which can be adapted to generate an endless variety of structures. Despite the structure descriptions being local, one can manage to guarantee or optimize global properties of the synthesized output. There is still potential in computer graphics and shape synthesis interpretations of Constraint Satisfaction Problem (CSP) resolution algorithms. Embedding abstract objects into different kinds of geometric space tends to produce interesting results.

In my opinion, the main contribution of this thesis is showing that the layout point of view can help breaking down problems and solve them efficiently. Analyzing structures and deriving local properties are great initiatives when confronted to challenging situations. Additionally, due to the ubiquity of layout problems across many different domains, as showcased in Chapters 1 and 2, a huge variety of methods have already been successfully implemented for solving them. Drawing on this existing state of the art is useful when tackling new problems. The main remaining challenge is that there is currently little crossover between different fields dealing with layout problems: industrial facility layouts, architectural floorplans, electronic circuit layouts... The similarities tend to stop at the basic algorithmic level, each work quickly diverging into the specificities of their domain, instead of building taller abstractions (of which I am also guilty). I feel that there is unexploited potential that could be tapped into by drawing bridges between these different fields.

I want to end this conclusion with a few personal thoughts, which can be skipped without harm to the scientific contribution of this document. Research is a very human activity, influenced by our moods, thoughts and emotions. I hope that this shines some light on the day-to-day reality of working on a research project, and maybe help some stray PhD student stumbling upon this thesis. I chose to present our work on foldable circuit boards for 3D LED-based screens before our work on supports for 3D printing, despite them happening in the opposite order chronologically. *PCBend* presented a more traditional flavor of computational fabrication, and is a better introduction to the work

presented here and the richness of the related research fields. This project had a significantly wider scope than *ProcSupGen*, and it went through uncountable iterations before reaching its current status. It was both incredibly fulfilling and terribly frustrating. Computational fabrication consists in part of handling a relentless series of unforeseen problems, due to the seemingly infinite complexity of the task at hand. Design flaws, fabrication delays, components vanishing in magic smoke, malfunctioning prototypes..., everything can go wrong and will go wrong as soon as you turn your back for a few seconds. But then at some point, either by sheer luck or hard work (depending on your mood), everything decides to work together. These moments of satisfaction are unmatched, and make you temporarily forget all of the hardships. Instead, I like to think of *ProcSupGen* as a more atypical and whimsical project. We tried to apply an existing technique that we found interesting to a seemingly unrelated real-life problem. This type of approach sometimes ends in failure, which can be hard to deal given the expectations on PhD students, but always results in learning interesting things that can be later reused. This project happened during the multiple COVID lockdowns at the beginning of my thesis, and most of the time was spent perplexedly looking at brightly colored voxels on a screen. Taming model synthesis and developing an intuition for a seemingly chaotic system has been one of my favorite moments of this thesis. Revisiting all of this to expose the fine details of our techniques has made me realize my own evolution throughout these years, and has been extremely gratifying.

Bibliography

[AI 22] AI and Games, director. *How Townscaper Works: A Story Four Games in the Making | AI and Games #65*. Mar. 28, 2022 (cit. on p. 34).

[AAD98] Paul Alexander, Seth Allen, and Debasish Dutta. "Part Orientation and Build Cost Determination in Layered Manufacturing". In: *Computer-Aided Design* 30.5 (1998), pp. 343–356. DOI: 10/bmxst3 (cit. on p. 80).

[Ali+16] Daniel G. Aliaga et al. "Inverse Procedural Modeling of 3D Models for Virtual Worlds". In: *ACM SIGGRAPH 2016 Courses*. SIGGRAPH '16: Special Interest Group on Computer Graphics and Interactive Techniques Conference. Anaheim California: ACM, July 24, 2016, pp. 1–316. ISBN: 978-1-4503-4289-6. DOI: 10.1145/2897826.2927323 (cit. on p. 35).

[All23] All Flex Solutions. *Why Do Rigid Flex PCB's Cost More Than Other PCB's?* Sept. 18, 2023. URL: https://web.archive.org/web/20231025082807/https://www.allflexinc.com/blog/why-do-rigid-flex-pcbs-cost-more-than-other-pcbs/ (cit. on pp. 5, 44).

[All+02] A. Allan et al. "2001 Technology Roadmap for Semiconductors". In: *Computer* 35.1 (Jan. 2002), pp. 42–53. ISSN: 1558-0814. DOI: 10.1109/2.976918 (cit. on p. 28).

[All+88] Joseph W. Allison et al. "Boolean Layer Comparison Slice". U.S. pat. Patent 5854748. 1988 (cit. on p. 80).

[Alt05] Altium. *Altium Designer*. 2005 (cit. on p. 43).

[Alt14] Altium. *Designing a Rigid-Flex PCB in Altium Designer*. 2014. URL: https://www.altium.com/documentation/altium-designer/designing-rigid-flex-pcb (cit. on p. 44).

[An+18] Byoungkwon An et al. "Thermorph: Democratizing 4D Printing of Self-Folding Materials and Interfaces". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18: CHI Conference on Human Factors in Computing Systems. Montreal QC Canada: ACM, Apr. 21, 2018, pp. 1–12. ISBN: 978-1-4503-5620-6. DOI: 10/gg3zv9 (cit. on p. 49).

[And18] Andreas Olofsson. "Silicon Compilers - Version 2.0". International Symposium on Physical Design (Monterey, CA). Mar. 25–28, 2018 (cit. on p. 28).

[Ang+18] Kristin Angel et al. "Selective Electroplating of 3D Printed Parts". In: *Additive Manufacturing* 20 (Mar. 2018), pp. 164–172. ISSN: 22148604. DOI: 10.1016/j.addma.2018.01.006 (cit. on p. 44).

[App+01] David Applegate et al. *Concorde TSP*. 2001 (cit. on p. 60).

[Bar93] Ya. M. Barzdin. "On the Realization of Networks in Three-Dimensional Space". In: *Selected Works of A. N. Kolmogorov: Volume III: Information Theory and the Theory of Algorithms*. Ed. by A. N. Shiryayev. Mathematics and Its Applications. Dordrecht: Springer Netherlands, 1993, pp. 194–202. ISBN: 978-94-017-2973-4. DOI: 10.1007/978-94-017-2973-4_11 (cit. on p. 14).

[BFR17] Amit H. Bermano, Thomas Funkhouser, and Szymon Rusinkiewicz. "State of the Art in Methods and Representations for Fabrication-Aware Design". In: *Computer Graphics Forum* 36.2 (May 2017), pp. 509–535. ISSN: 01677055. DOI: 10.1111/cgf.13146 (cit. on pp. 3, 36, 126).

[BWL18] Xiaojun Bian, Li-Yi Wei, and Sylvain Lefebvre. "Tile-Based Pattern Design with Topology Control". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.1 (July 25, 2018), pp. 1–15. ISSN: 25776193. DOI: 10/ghgdxf (cit. on p. 32).

[Boh07] Mark Bohr. "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper". In: *IEEE Solid-State Circuits Newsletter* 12.1 (Win. 2007), pp. 11–13. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2007.4785534 (cit. on p. 21).

[BFM06] Drago Bokal, Gasper Fijavz, and Bojan Mohar. "The Minor Crossing Number". In: *SIAM Journal on Discrete Mathematics* 20.2 (Jan. 2006), pp. 344–356. ISSN: 0895-4801, 1095-7146. DOI: 10.1137/05062706X (cit. on p. 15).

[BWS10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. "A Connection between Partial Symmetry and Inverse Procedural Modeling". In: *ACM Transactions on Graphics* 29.4 (July 26, 2010), 104:1–104:10. ISSN: 0730-0301. DOI: 10.1145/1778765.1778841 (cit. on p. 36).

[Bok+12] Martin Bokeloh et al. "An Algebraic Model for Parameterized Shape Editing". In: *ACM Transactions on Graphics* 31.4 (July 1, 2012), 78:1–78:10. ISSN: 0730-0301. DOI: 10.1145/2185520.2185574 (cit. on p. 36).

[Boy04] Chuck Boyer. *The 360 Revolution*. International Business Machines Corporation (IBM), Apr. 2004 (cit. on p. 22).

[Bra11] David Braben. *Classic Game Postmortem - ELITE*. 2011. URL: https://www.gdcvault.com/play/1014628/ (cit. on p. 29).

[Bra21] H. R. Brahana. "Systems of Circuits on Two-Dimensional Manifolds". In: *Annals of Mathematics* 23.2 (1921), pp. 144–168. ISSN: 0003-486X. DOI: 10.2307/1968030. JSTOR: 1968030 (cit. on p. 14).

[BC47] Cledo Brunetti and Roger W Curtis. *Circular of the Bureau of Standards No. 468:: Printed Circuit Techniques*. NBS CIRC 468. Gaithersburg, MD: National Bureau of Standards, 1947, NBS CIRC 468. DOI: 10.6028/NBS.CIRC.468 (cit. on p. 19).

[CM12] Sergio Cabello and Bojan Mohar. *Adding One Edge to Planar Graphs Makes Crossing Number and 1-Planarity Hard*. Mar. 27, 2012. DOI: 10.48550/arXiv.1203.5944. arXiv: 1203.5944 [cs, math]. URL: http://arxiv.org/abs/1203.5944. Pre-published (cit. on p. 14).

[Cad22] Cadence PCB Solutions. *Does Autorouting Fit in a Standard PCB Design Workflow?* Dec. 12, 2022. URL: https://resources.pcb.cadence.com/blog/does-autorouting-fit-in-a-standard-pcb-design-workflow (cit. on p. 28).

[Cad88] Cadsoft Computer GbmH. *EAGLE*. Autodesk, 1988 (cit. on pp. 5, 43).

[CSL20] Inês Caetano, Luís Santos, and António Leitão. "Computational Design in Architecture: Defining Parametric, Generative, and Algorithmic Design". In: *Frontiers of Architectural Research* 9.2 (June 2020), pp. 287–300. ISSN: 20952635. DOI: 10.1016/j.foar.2019.12.008 (cit. on pp. 29, 127).

[CZ18] Sebastien J. P. Callens and Amir A. Zadpoor. "From Flat Sheets to Curved Geometries: Origami and Kirigami Approaches". In: *Materials Today* 21.3 (Apr. 1, 2018), pp. 241–264. ISSN: 1369-7021. DOI: 10.1016/j.mattod.2017.10.004 (cit. on p. 45).

[Cat24] Erin Catto. *Box2d*. Mar. 7, 2024 (cit. on p. 57).

[CJR95] Kumar Chalasani, Larry Jones, and Larry Roscoe. "Support Generation for Fused Deposition Modeling". In: *Solid Freeform Fabrication Symposium*. 1995, pp. 229–241 (cit. on p. 80).

[CK92] Jean-Pierre Charras and KiCad Development Team. *KiCad*. 1992 (cit. on pp. 5, 43, 129).

[Che+16] Weikai Chen et al. "Synthesis of Filigrees for Digital Fabrication". In: *ACM Transactions on Graphics* 35.4 (July 11, 2016), pp. 1–13. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/2897824.2925911 (cit. on p. 46).

[Che+17] Weikai Chen et al. "Fabricable Tile Decors". In: *ACM Transactions on Graphics* 36.6 (Dec. 31, 2017), pp. 1–15. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3130800.3130817 (cit. on p. 46).

[CG07] Markus Chimani and Carsten Gutwenger. "Algorithms for the Hypergraph and the Minor Crossing Number Problems". In: *Algorithms and Computation: 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007. Proceedings*. Berlin, Heidelberg: Springer-Verlag, Dec. 2007, pp. 184–195. ISBN: 978-3-540-77118-0. DOI: 10.1007/978-3-540-77120-3_18 (cit. on p. 15).

[CKS77] Y. E. Cho, A. J. Korenjak, and D. E. Stockton. "Floss: An Approach to Automated Layout for High-Volume Designs". In: *Proceedings of the 14th Design Automation Conference*. DAC '77. IEEE Press, Jan. 1, 1977, pp. 138–141 (cit. on pp. 22, 24).

[Cho56] N. Chomsky. "Three Models for the Description of Language". In: *IRE Transactions on Information Theory* 2.3 (Sept. 1956), pp. 113–124. ISSN: 2168-2712. DOI: 10.1109/TIT.1956.1056813 (cit. on p. 32).

[Coh+03] Michael F. Cohen et al. "Wang Tiles for Image and Texture Generation". In: *ACM Transactions on Graphics* 22.3 (July 1, 2003), pp. 287–294. ISSN: 0730-0301. DOI: 10.1145/882262.882265 (cit. on p. 31).

[Com+99] Committee on Innovations in Computing and Communications: Lessons From History et al. *Funding a Revolution: Government Support for Computing Research*. Washington, D.C.: National Academies Press, Jan. 11, 1999, p. 6323. ISBN: 978-0-309-06278-7. DOI: 10.17226/6323 (cit. on pp. 17, 23).

[Com24a] Computer History Museum. *1948: The European Transistor Invention | The Silicon Engine | Computer History Museum*. computerhistory.org. 2024. URL: https://www.computerhistory.org/siliconengine/the-european-transistor-invention/ (cit. on p. 17).

[Com24b] Computer History Museum. *1967: Application Specific Integrated Circuits Employ Computer-Aided Design | The Silicon Engine | Computer History Museum*. computerhistory.org. 2024. URL: https://www.computerhistory.org/siliconengine/application-specific-integrated-circuits-employ-computer-aided-design/ (cit. on p. 23).

[Com88] Association for Computing Machinery, ed. *25 Years of Electronic Design Automation: A Compendium of Papers from the Design Automation Conference*. New York, N.Y. : Baltimore, MD: Association for Computing Machinery ; May be ordered from ACM Order Dept, 1988. 626 pp. ISBN: 978-0-89791-267-9 (cit. on p. 21).

[Con12] Lynn Conway. "Reminiscences of the VLSI Revolution: How a Series of Failures Triggered a Paradigm Shift in Digital Design". In: *IEEE Solid-State Circuits Magazine* 4.4 (Dec. 2012), pp. 8–31. ISSN: 1943-0590. DOI: 10.1109/MSSC.2012.2215752 (cit. on p. 23).

[Cra21] David L. Craddock. *Dungeon Hacks: How Nethack, Angband, and Other Rougelikes Changed the Course of Video Games*. 1st edition. Boca Raton: CRC Press, 2021. ISBN: 978-1-03-205240-3 978-1-03-205154-3 (cit. on p. 29).

[cy319] cy384. *Flex without the Flex*. 2019. URL: http://www.cy384.com/blog/flex-pcbs.html (cit. on p. 99).

[Dac70] M. F. Dacey. "The Syntax of a Triangle and Some Other Figures". In: *Pattern Recognition* 2.1 (Jan. 1, 1970), pp. 11–31. ISSN: 0031-3203. DOI: 10.1016/0031-3203(70)90038-5 (cit. on p. 33).

[Dal+06] Ketan Dalal et al. "A Spectral Approach to NPR Packing". In: *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*. NPAR06: The 4th International Symposium on Non-Photorealistic Animation. Annecy France: ACM, June 5, 2006, pp. 71–78. ISBN: 978-1-59593-357-7. DOI: 10.1145/1124728.1124741 (cit. on p. 46).

[Dan05] G. Dan Hutcheson. "Moore's Law: The History and Economics of an Observation That Changed the World". In: *The Electrochemical Society Interface* 14.1 (Mar. 1, 2005), pp. 17–21. ISSN: 1064-8208, 1944-8783. DOI: 10.1149/2.F04051IF (cit. on p. 20).

[DM22] Anthony C. Davies and Franco Maloberti. *A Short History of Circuits and Systems*. 1st ed. New York: River Publishers, Sept. 1, 2022. ISBN: 978-1-00-333693-8. DOI: 10.1201/9781003336938 (cit. on p. 16).

[dBoi22] Aurelie de Boissieu. "Introduction to Computational Design: Subsets, Challenges in Practice and Emerging Roles". In: *Industry 4.0 for the Built Environment: Methodologies, Technologies and Skills*. Ed. by Marzia Bolpagni, Rui Gavina, and Diogo Ribeiro. Cham: Springer International Publishing, 2022, pp. 55–75. ISBN: 978-3-030-82430-3. DOI: 10.1007/978-3-030-82430-3_3 (cit. on pp. 2, 126).

[DH07] M. Dehn and P. Heegaard. "III AB.3 Analysis situs." In: *Encyklopädie der mathematischen Wissenschaften mit Einschluss ihrer Anwendungen*. Vol. 3. B.G. Teubner Verlag, 1907, pp. 153–220 (cit. on p. 14).

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge ; New York: Cambridge University Press, 2007. 472 pp. ISBN: 978-0-521-85757-4 (cit. on p. 45).

[Den+74] R.H. Dennard et al. "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. ISSN: 1558-173X. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 20).

[Dor04] Armand Van Dormael. "The 'French' Transistor". In: *Proceedings of the 2004 IEEE Conference on the History of Electronics* (June 2004) (cit. on p. 17).

[DHL14] Jérémie Dumas, Jean Hergel, and Sylvain Lefebvre. "Bridging the Gap: Automated Steady Scaffoldings for 3D Printing". In: *ACM Transactions on Graphics* 33.4 (July 2014), pp. 1–10. ISSN: 0730-0301, 1557-7368. DOI: 10/ghpm83 (cit. on pp. vi, vii, 78–80, 92, 93, 97, 98, 100, 130).

[Dun80] A. E. Dunlop. "SLIM-the Translation of Symbolic Layouts into Mask Data". In: *Proceedings of the 17th Design Automation Conference*. DAC '80. New York, NY, USA: Association for Computing Machinery, June 23, 1980, pp. 595–602. ISBN: 978-0-89791-020-0. DOI: 10.1145/800139.804592 (cit. on p. 22).

[ESW96] Peter Eades, Charles Stirk, and Sue Whitesides. "The Techniques of Komolgorov and Bardzin for Three-Dimensional Orthogonal Graph Drawings". In: *Information Processing Letters* 60.2 (Oct. 28, 1996), pp. 97–103. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(96)00133-0 (cit. on p. 14).

[Ebe03] David S. Ebert. *Texturing & Modeling: A Procedural Approach*. 3rd ed. Amsterdam Boston: Academic Press, 2003. ISBN: 978-1-55860-848-1 (cit. on p. 30).

[Ein85] Norman G. Einspruch, ed. *VLSI Handbook*. Handbooks in Science and Technology. Orlando, Fla: Academic Press, 1985. 902 pp. ISBN: 978-0-12-234100-7 (cit. on p. 22).

[EW89] Paul Eisler and Mari E. W. Williams. *My Life with the Printed Circuit*. Bethlehem: Lehigh University Press, 1989. 170 pp. ISBN: 978-0-934223-04-1 (cit. on p. 18).

[Eur09] European Commission. Joint Research Center. Institute for Prospective and Technological Studies. *The Future of Semiconductor Intellectual Property Architectural Blocks in Europe*. LU: Publications Office, 2009 (cit. on p. 24).

[Fag09] Federico Faggin. "The Making of the First Microprocessor". In: *IEEE Solid-State Circuits Magazine* 1.1 (2009), pp. 8–21. ISSN: 1943-0590. DOI: 10.1109/MSSC.2008.930938 (cit. on p. 22).

[Fan+22] Filippo Andrea Fanni et al. "PAVEL: Decorative Patterns with Packed Volumetric Elements". In: *ACM Transactions on Graphics* 41.2 (Apr. 30, 2022), pp. 1–15. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3502802 (cit. on p. 46).

[Fat21] Robert W. Fathauer. *Tessellations: Mathematics, Art, and Recreation*. Boca Raton: AK Peters/CRC Press, 2021. 1 p. ISBN: 978-0-429-19712-3 (cit. on p. 12).

[Fey60] Richard P. Feynman. "There's Plenty of Room at the Bottom". In: *Enigineering and Science* 23.5 (Feb. 1960), pp. 22–36. ISSN: 0013-7812 (cit. on p. 18).

[Flo+17] Patrick F. Flowers et al. "3D Printing Electronic Components and Circuits with Conductive Thermoplastic Filament". In: *Additive Manufacturing* 18 (Dec. 2017), pp. 156–163. ISSN: 22148604. DOI: 10.1016/j.addma.2017.10.002 (cit. on p. 44).

[FPT81] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. "Optimal Packing and Covering in the Plane Are NP-complete". In: *Information Processing Letters* 12.3 (June 13, 1981), pp. 133–137. ISSN: 0020-0190. DOI: 10.1016/0020-0190(81)90111-3 (cit. on p. 46).

[Fre+22] Marco Freire et al. "Procedural Bridges-and-Pillars Support Generation". In: *Eurographics 2022 - 43rd annual conference of the european association for computer graphics* (Apr. 2022), 4 pages. DOI: 10.2312/EGS.20221025 (cit. on pp. 8, 79).

[Fre+23] Marco Freire et al. "PCBend: Light up Your 3D Shapes with Foldable Circuit Boards". In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–16. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3592411 (cit. on pp. 8, 38).

[Gam16] Game Maker's Toolkit, director. *How (and Why) Spelunky Makes Its Own Levels*. Apr. 12, 2016 (cit. on pp. 4, 128).

[Gao+21] Yue Gao et al. "Hypergraph Learning: Methods and Practices". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.3039374 (cit. on p. 15).

[GJ12] Rodrigo García Alvarado and Jaime Jofre Muñoz. "The Control Of Shape: Origins Of Parametric Design In Architecture In Xenakis, Gehry And Grimshaw". In: *METU JOURNAL OF THE FACULTY OF ARCHITECTURE* (June 1, 2012). ISSN: 02585316. DOI: 10.4305/METU.JFA.2012.1.6 (cit. on pp. 2, 3, 127).

[GJ83] M. R. Garey and D. S. Johnson. "Crossing Number Is NP-Complete". In: *SIAM Journal on Algebraic Discrete Methods* 4.3 (Sept. 1983), pp. 312–316. ISSN: 0196-5212. DOI: 10.1137/0604033 (cit. on p. 14).

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. A Series of Books in the Mathematical Sciences. New York: W. H. Freeman, 1979. ISBN: 978-0-7167-1045-5 (cit. on p. 12).

[GN76] Dave Gibson and Scott Nance. "SLIC - Symbolic Layout of Integrated Circuits". In: *Proceedings of the 13th Design Automation Conference*. DAC '76. New York, NY, USA: Association for Computing Machinery, June 28, 1976, pp. 434–440. ISBN: 978-1-4503-7448-4. DOI: 10.1145/800146.804844 (cit. on p. 22).

[GS18] Daniel Groeger and Jürgen Steimle. "ObjectSkin: Augmenting Everyday Objects with Hydroprinted Touch Sensors and Displays". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.4 (Jan. 8, 2018), pp. 1–23. ISSN: 2474-9567. DOI: 10.1145/3161165 (cit. on p. 44).

[GS19] Daniel Groeger and Jürgen Steimle. "LASEC: Instant Fabrication of Stretchable Circuits Using a Laser Cutter". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19: CHI Conference on Human Factors in Computing Systems. Glasgow Scotland Uk: ACM, May 2, 2019, pp. 1–14. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300929 (cit. on p. 44).

[GG12] Misha Gromov and Larry Guth. "Generalizations of the Kolmogorov-Barzdin Embedding Estimates". In: *Duke Mathematical Journal* 161.13 (Oct. 1, 2012). ISSN: 0012-7094. DOI: 10.1215/00127094-1812840. arXiv: 1103.3423 [math] (cit. on p. 14).

[GS87] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. A Series of Books in the Mathematical Sciences. New York: W.H. Freeman, 1987. 700 pp. ISBN: 978-0-7167-1193-3 (cit. on p. 12).

[Gua12] Massimo Guarnieri. "The Age of Vacuum Tubes: Early Devices and the Rise of Radio Communications [Historical]". In: *IEEE Industrial Electronics Magazine* 6.1 (Mar. 2012), pp. 41–43. ISSN: 1932-4529. DOI: 10.1109/MIE.2012.2182822 (cit. on p. 16).

[Gum16] Maxim Gumin. *Wave Function Collapse*. 2016 (cit. on pp. 33, 79).

[HC21] Thomas Haigh and Paul E. Ceruzzi. *A New History of Modern Computing*. History of Computing. Cambridge [Massachusetts]: The MIT Press, 2021. ISBN: 978-0-262-54290-6 (cit. on p. 19).

[Ham20] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing, 2020. ISBN: 978-3-031-00460-5 978-3-031-01588-5. DOI: 10.1007/978-3-031-01588-5 (cit. on p. 15).

[Han+20] Ollie Hanton et al. "ProtoSpray: Combining 3D Printing and Spraying to Create Interactive Displays with Arbitrary Shapes". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20: CHI Conference on Human Factors in Computing Systems. Honolulu HI USA: ACM, Apr. 21, 2020, pp. 1–13. ISBN: 978-1-4503-6708-0. DOI: 10.1145/3313831.3376543 (cit. on p. 45).

[Har03] Charles A. Harper, ed. *Electronic Materials and Processes Handbook*. 3rd ed. McGraw-Hill Handbooks. New York: McGraw-Hill, 2003. 1 p. ISBN: 978-0-07-140214-9 (cit. on p. 18).

[HS71] Akihiro Hashimoto and James Stevens. "Wire Routing by Optimizing Channel Assignment within Large Apertures". In: *Proceedings of the 8th Design Automation Workshop*. DAC '71. New York, NY, USA: Association for Computing Machinery, June 28, 1971, pp. 155–169. ISBN: 978-1-4503-7465-1. DOI: 10.1145/800158.805069 (cit. on p. 22).

[Hau01] Alejo Hausner. "Simulating Decorative Mosaics". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH01: The 28th International Conference on Computer Graphics and Interactive Techniques. ACM, Aug. 2001, pp. 573–580. ISBN: 978-1-58113-374-5. DOI: 10.1145/383259.383327 (cit. on p. 46).

[He+21] Liang He et al. "ModElec: A Design Tool for Prototyping Physical Computing Devices Using Conductive 3D Printing". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5.4 (Dec. 27, 2021), pp. 1–20. ISSN: 2474-9567. DOI: 10.1145/3495000 (cit. on p. 44).

[Hei11] E.K. Heide. "Method for Generating and Building Support Structures with Deposition-Based Digital Manufacturing Systems". Pat. July 2011 (cit. on p. 80).

[HP19] John L. Hennessy and David A. Patterson. "A New Golden Age for Computer Architecture". In: *Communications of the ACM* 62.2 (Jan. 28, 2019), pp. 48–60. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3282307 (cit. on p. 21).

[Hig69] David W. Hightower. "A Solution to Line-Routing Problems on the Continuous Plane". In: *Proceedings of the 6th Annual Design Automation Conference*. DAC '69. New York, NY, USA: Association for Computing Machinery, Jan. 1, 1969, pp. 1–24. ISBN: 978-1-4503-7929-8. DOI: 10.1145/800260.809014 (cit. on p. 22).

[Hod+14] Steve Hodges et al. "Circuit Stickers: Peel-and-Stick Construction of Interactive Electronic Prototypes". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14: CHI Conference on Human Factors in Computing Systems. Toronto Ontario Canada: ACM, Apr. 26, 2014, pp. 1743–1746. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557150 (cit. on p. 44).

[HMB21] Freddie Hong, Connor Myant, and David Boyle. "Thermoformed Circuit Boards: Fabrication of Highly Conductive Freeform 3D Printed Circuit Boards with Heat Bending". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. May 6, 2021, pp. 1–10. DOI: 10.1145/3411764.3445469. arXiv: 2011.06473 [cs] (cit. on p. 44).

[HT74] John Hopcroft and Robert Tarjan. "Efficient Planarity Testing". In: *Journal of the ACM* 21.4 (Oct. 1, 1974), pp. 549–568. ISSN: 0004-5411. DOI: 10.1145/321850.321852 (cit. on p. 13).

[Hu+16] Wenchao Hu et al. "Surface Mosaic Synthesis with Irregular Tiles". In: *IEEE Transactions on Visualization and Computer Graphics* 22.3 (Mar. 1, 2016), pp. 1302–1313. ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2498620 (cit. on p. 46).

[Hua+09a] Xiaomao Huang et al. "Slice Data Based Support Generation Algorithm for Fused Deposition Modeling". In: *Tsinghua Science and Technology* 14.S1 (2009), pp. 223–228. DOI: 10/dd2cb6 (cit. on p. 80).

[Hua+09b] Xiaomao Huang et al. "Sloping Wall Structure Support Generation for Fused Deposition Modeling". In: *The International Journal of Advanced Manufacturing Technology* 42.11-12 (2009), pp. 1074–1081. DOI: 10/bn3txh (cit. on p. 80).

[INR13] INRIA. *IceSL Modeler and Slicer*. 2013 (cit. on pp. 8, 79, 100).

[Jan03] Dirk Jansen, ed. *The Electronic Design Automation Handbook*. Boston, MA: Springer US, 2003. ISBN: 978-1-4419-5369-8 978-0-387-73543-6. DOI: 10.1007/978-0-387-73543-6 (cit. on p. 22).

[JSC15] Naveen Noah Jason, Wei Shen, and Wenlong Cheng. "Copper Nanowires as Conductive Ink for Low-Cost Draw-On Electronics". In: *ACS Applied Materials & Interfaces* 7.30 (Aug. 5, 2015), pp. 16760–16766. ISSN: 1944-8244. DOI: 10.1021/acsami.5b04522 (cit. on p. 44).

[JR21] Emmanuel Jeandel and Michaël Rao. "An Aperiodic Set of 11 Wang Tiles". In: *Advances in Combinatorics* (Jan. 11, 2021). ISSN: 25175599. DOI: 10.19086/aic.18614 (cit. on p. 31).

[JXS18] Jingchao Jiang, Xun Xu, and Jonathan Stringer. "Support Structures for Additive Manufacturing: A Review". In: *Journal of Manufacturing and Materials Processing* 2.4 (Sept. 2018), p. 64. ISSN: 2504-4494. DOI: 10/gf6fcx (cit. on pp. 7, 80).

[Joh79] D. Johannsen. "Bristle Blocks: A Silicon Compiler". In: *16th Design Automation Conference*. 16th Design Automation Conference. June 1979, pp. 310–313. DOI: 10.1109/DAC.1979.1600125 (cit. on p. 22).

[Kah+11] Andrew B. Kahng et al. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Netherlands, 2011. ISBN: 978-90-481-9590-9. DOI: 10.1007/978-90-481-9591-6 (cit. on pp. 15, 25).

[Kal20] Andriani-Melina Kalama. "Kerf Bending: A Genealogy of Cutting Patterns for Single and Double Curvature". In: moNGeometrija 2020 (7th International Scientific Conference on Geometry and Graphics). 2020, p. 17 (cit. on p. 40).

[Kal+12] Javor Kalojanov et al. "Microtiles: Extracting Building Blocks from Correspondences". In: *Computer Graphics Forum* 31.5 (2012), pp. 1597–1606. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2012.03165.x (cit. on p. 36).

[KS17] Isaac Karth and Adam M. Smith. "WaveFunctionCollapse Is Constraint Solving in the Wild". In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. Hyannis Massachusetts: ACM, Aug. 2017, pp. 1–10. ISBN: 978-1-4503-5319-9. DOI: 10/gfsb97 (cit. on p. 79).

[Kaw+13] Yoshihiro Kawahara et al. "Instant Inkjet Circuits: Lab-Based Inkjet Printing to Support Rapid Prototyping of UbiComp Devices". In: *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '13: The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing. Zurich Switzerland: ACM, Sept. 8, 2013, pp. 363–372. ISBN: 978-1-4503-1770-2. DOI: 10.1145/2493432.2493486 (cit. on p. 44).

[KMR08] Ken-ichi Kawarabayashi, Bojan Mohar, and Bruce Reed. "A Simpler Linear Time Algorithm for Embedding Graphs into an Arbitrary Surface and the Genus of Graphs of Bounded Tree-Width". In: *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. 2008 IEEE 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS). Philadelphia, PA, USA: IEEE, Oct. 2008, pp. 771–780. ISBN: 978-0-7695-3436-7. DOI: 10.1109/FOCS.2008.53 (cit. on p. 14).

[KSP73] B. W. Kernighan, D. G. Schweikert, and G. Persky. "An Optimum Channel-Routing Algorithm for Polycell Layouts of Integrated Circuits". In: *Proceedings of the 10th Design Automation Workshop*. DAC '73. IEEE Press, June 25, 1973, pp. 50–59 (cit. on p. 22).

[El-+09] Dania El-Khechen et al. “Packing 2 × 2 Unit Squares into Grid Polygons Is NP-complete”. In: *Canadian Conference on Computational Geometry* (2009) (cit. on p. 46).

[Kim+19] Myung Jun Kim et al. “One-Step Electrodeposition of Copper on Conductive 3D Printed Objects”. In: *Additive Manufacturing* 27 (May 2019), pp. 318–326. ISSN: 22148604. DOI: 10.1016/j.addma.2019.03.016 (cit. on p. 44).

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (May 13, 1983), pp. 671–680. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.220.4598.671 (cit. on p. 22).

[Kir64] Russell A. Kirsch. “Computer Interpretation of English Text and Picture Patterns”. In: *IEEE Transactions on Electronic Computers* EC-13.4 (Aug. 1964), pp. 363–376. ISSN: 0367-7508. DOI: 10.1109/PGEC.1964.263816 (cit. on p. 33).

[KCG58] M. Kloomok, P. W. Case, and H. H. Graff. “The Recording, Checking, and Printing of Logic Diagrams”. In: *Papers and Discussions Presented at the December 3-5, 1958, Eastern Joint Computer Conference: Modern Computers: Objectives, Designs, Applications on XX - AIEE-ACM-IRE ’58 (Eastern)*. Papers and Discussions Presented at the December 3-5, 1958, Eastern Joint Computer Conference: Modern Computers: Objectives, Designs, Applications. Philadelphia, Pennsylvania: ACM Press, 1958, pp. 108–118. DOI: 10.1145/1458043.1458067 (cit. on pp. 21, 22).

[KMB06] Ivana Kolingerová, Petr März, and Bedrich Beneš. “Tensor Product Surfaces as Rewriting Process”. In: *Proceedings of the 22nd Spring Conference on Computer Graphics*. SCCG06: Spring Conference on Computer Graphics. Casta-Papiernicka Slovakia: ACM, Apr. 20, 2006, pp. 107–112. ISBN: 978-1-4503-2829-6. DOI: 10.1145/2602161.2602173 (cit. on p. 33).

[Kon03] Konrad Polthier. *Imaging Maths - Unfolding Polyhedra*. Plus Maths. Jan. 11, 2003. URL: https://plus.maths.org/content/imaging-maths-unfolding-polyhedra (cit. on p. 45).

[Kor+20] Thorsten Korpitsch et al. “Simulated Annealing to Unfold 3D Meshes and Assign Glue Tabs”. In: *Journal of WSCG* 28.1-2 (2020), pp. 47–56. ISSN: 12136972. DOI: 10.24132/JWSCG.2020.28.6 (cit. on p. 45).

[KNG15] Vasilis Kostakis, Vasilis Niaros, and Christos Giotitsas. “Open Source 3D Printing as a Means of Learning: An Educational Experiment in Two High Schools in Greece”. In: *Telematics and Informatics* 32.1 (Feb. 1, 2015), pp. 118–128. ISSN: 0736-5853. DOI: 10.1016/j.tele.2014.05.001 (cit. on p. 3).

[Lag+10] A. Lagae et al. “State of the Art in Procedural Noise Functions”. In: *Eurographics 2010 - State of the Art Reports* (2010), 19 pages. ISSN: 1017-4656. DOI: 10.2312/EGST.20101059 (cit. on p. 30).

[Lag07] Ares Lagae. "Tile-Based Methods in Computer Graphics ; Tegelgebaseerde Methodes in Computer Graphics". PhD thesis. Katholieke Universiteit Leuven, Belgium, 2007 (cit. on p. 31).

[LD06] Ares Lagae and Philip Dutré. "An Alternative for Wang Tiles: Colored Edges versus Colored Corners". In: *ACM Transactions on Graphics* 25.4 (Oct. 2006), pp. 1442–1459. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/1183287.1183296 (cit. on p. 31).

[Lag+08] Ares Lagae et al. "Tile-Based Methods for Interactive Applications". In: *ACM SIGGRAPH 2008 Classes*. SIGGRAPH '08. New York, NY, USA: Association for Computing Machinery, Aug. 11, 2008, pp. 1–267. ISBN: 978-1-4503-7845-1. DOI: 10.1145/1401132.1401254 (cit. on p. 31).

[Lag+09] Ares Lagae et al. "Procedural Noise Using Sparse Gabor Convolution". In: *ACM Transactions on Graphics* 28.3 (July 27, 2009), 54:1–54:10. ISSN: 0730-0301. DOI: 10.1145/1531326.1531360 (cit. on p. 30).

[Lec09] Christophe Lecoutre. *Constraint Networks: Techniques and Algorithms*. Hoboken, NJ: ISTE/John Wiley, 2009. 586 pp. ISBN: 978-1-84821-106-3 (cit. on p. 81).

[Lee+18] Dongchi Lee et al. "Origami Robots with Flexible Printed Circuit Sheets". In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*. UbiComp '18: The 2018 ACM International Joint Conference on Pervasive and Ubiquitous Computing. Singapore Singapore: ACM, Oct. 8, 2018, pp. 392–395. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267620 (cit. on p. 40).

[Lee+20] Yu-Ki Lee et al. "Computational Wrapping: A Universal Method to Wrap 3D-curved Surfaces with Nonstretchable Materials for Conformal Devices". In: *Science Advances* 6.15 (Apr. 2020), eaax6212. ISSN: 2375-2548. DOI: 10/gk42fh (cit. on p. 45).

[Lef14] Sylvain Lefebvre. "Synthèse de Textures Par l'exemple Pour Les Applications Interactives". Habilitation à Diriger des Recherches. Nancy: Université de Lorraine, June 30, 2014 (cit. on p. 30).

[Lév+02] Bruno Lévy et al. "Least Squares Conformal Maps for Automatic Texture Atlas Generation". In: *ACM Transactions on Graphics* 21.3 (July 1, 2002), pp. 362–371. ISSN: 0730-0301. DOI: 10.1145/566654.566590 (cit. on p. 12).

[LB17] Jens Lienig and Hans Bruemmer. *Fundamentals of Electronic Systems Design*. Springer International Publishing, 2017. ISBN: 978-3-319-55839-4. DOI: 10.1007/978-3-319-55840-0 (cit. on pp. 22, 27).

[LS20] Jens Lienig and Juergen Scheible. *Fundamentals of Layout Design for Electronic Circuits*. Springer International Publishing, 2020. ISBN: 978-3-030-39283-3. DOI: 10.1007/978-3-030-39284-0 (cit. on pp. 5, 21, 23, 24, 27, 28, 43, 53).

[Lil30] Edgar Julius Lilienfeld. "Method and Apparatus for Controlling Electric Currents". U.S. pat. 1745175A. Individual. Jan. 28, 1930 (cit. on p. 17).

[Lin68a] Aristid Lindenmayer. "Mathematical Models for Cellular Interactions in Development I. Filaments with One-Sided Inputs". In: *Journal of Theoretical Biology* 18.3 (Mar. 1968), pp. 280–299. ISSN: 00225193. DOI: 10.1016/0022-5193(68)90079-9 (cit. on p. 33).

[Lin68b] Aristid Lindenmayer. "Mathematical Models for Cellular Interactions in Development II. Simple and Branching Filaments with Two-Sided Inputs". In: *Journal of Theoretical Biology* 18.3 (Mar. 1968), pp. 300–315. ISSN: 00225193. DOI: 10.1016/0022-5193(68)90080-5 (cit. on p. 33).

[Liu+18] Jikai Liu et al. "Current and Future Trends in Topology Optimization for Additive Manufacturing". In: *Structural and Multidisciplinary Optimization* 57.6 (June 1, 2018), pp. 2457–2483. ISSN: 1615-1488. DOI: 10/gdrwvj (cit. on p. 80).

[Liv+17] Marco Livesu et al. "From 3D Models to 3D Prints: An Overview of the Processing Pipeline". In: *Computer Graphics Forum* 36.2 (May 2017), pp. 537–564. ISSN: 01677055. DOI: 10/gbmq9g (cit. on p. 80).

[Loj06] Bo Lojek. *History of Semiconductor Engineering*. New-York: Springer, 2006. ISBN: 978-3-540-34257-1 (cit. on p. 19).

[MWT11] Chongyang Ma, Li-Yi Wei, and Xin Tong. "Discrete Element Textures". In: *ACM SIGGRAPH 2011 Papers*. SIGGRAPH '11. New York, NY, USA: Association for Computing Machinery, July 25, 2011, pp. 1–10. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964957 (cit. on p. 30).

[Ma+14] Chongyang Ma et al. "Game Level Layout from Design Specification". In: *Computer Graphics Forum* 33.2 (May 2014), pp. 95–104. ISSN: 01677055. DOI: 10/ghgdws (cit. on pp. 4, 128).

[Mac77] Alan K. Mackworth. "Consistency in Networks of Relations". In: *Artificial Intelligence* 8.1 (Feb. 1, 1977), pp. 99–118. ISSN: 0004-3702. DOI: 10.1016/0004-3702(77)90007-8 (cit. on p. 81).

[Mak13] Tsugio Makimoto. "Implications of Makimoto's Wave". In: *Computer* 46.12 (Dec. 2013), pp. 32–37. ISSN: 0018-9162. DOI: 10.1109/MC.2013.294 (cit. on p. 24).

[Man83] Anthony Mansfield. "Determining the Thickness of Graphs Is NP-hard". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 93.1 (Jan. 1983), pp. 9–23. ISSN: 1469-8064, 0305-0041. DOI: 10.1017/S030500410006028X (cit. on p. 14).

[Mar17] David Marrachki. *Autorouting, or No Autorouting? A History of Failed Design Automation*. Altium. Feb. 21, 2017. URL: https://resources.altium.com/p/to-autoroute-or-not-to-autoroute-a-history-of-failed-design-automation (cit. on p. 28).

[Mar15] Lee Martin. "The Promise of the Maker Movement for Education". In: *Journal of Pre-College Engineering Education Research (J-PEER)* 5.1 (Apr. 29, 2015). ISSN: 2157-9288. DOI: 10.7771/2157-9288.1099 (cit. on p. 3).

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley Series in Computer Science. Reading, Mass.: Addison-Wesley, 1980. 396 pp. ISBN: 978-0-201-04358-7 (cit. on p. 23).

[Mer07] Paul Merrell. "Example-Based Model Synthesis". In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games - I3D '07*. The 2007 Symposium. Seattle, Washington: ACM Press, 2007, p. 105. ISBN: 978-1-59593-628-8. DOI: 10/cqqk94 (cit. on pp. 33, 79).

[Mer09] Paul Merrell. "Model Synthesis". Chapel Hill: University of North Carolina, 2009 (cit. on pp. vi, vii, 81, 86, 95, 130).

[Mer21] Paul Merrell. *Comparing Model Synthesis and Wave Function Collapse*. July 28, 2021 (cit. on p. 35).

[Mer23] Paul Merrell. "Example-Based Procedural Modeling Using Graph Grammars". In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–16. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3592119 (cit. on pp. 35, 36).

[MM08] Paul Merrell and Dinesh Manocha. "Continuous Model Synthesis". In: *ACM Transactions on Graphics* 27.5 (Dec. 1, 2008), 158:1–158:7. ISSN: 0730-0301. DOI: 10.1145/1409060.1409111 (cit. on p. 35).

[MM09] Paul Merrell and Dinesh Manocha. "Constraint-Based Model Synthesis". In: *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling on - SPM '09*. 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling. San Francisco, California: ACM Press, 2009, p. 101. ISBN: 978-1-60558-711-0. DOI: 10/d8tx7t (cit. on p. 35).

[MID92] Research Association Mechatronic Integrated Devices 3-D MID. *3D-MID Technology*. 1992. URL: https://www.3d-mid.de/en/technology/ (cit. on p. 44).

[Mit+14] Niloy J. Mitra et al. "Structure-Aware Shape Processing". In: *ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. New York, NY, USA: Association for Computing Machinery, July 27, 2014, pp. 1–21. ISBN: 978-1-4503-2962-0. DOI: 10.1145/2614028.2615401 (cit. on p. 32).

[Moh99] Bojan Mohar. "A Linear Time Algorithm for Embedding Graphs in an Arbitrary Surface". In: *SIAM Journal on Discrete Mathematics* 12.1 (Jan. 1999), pp. 6–26. ISSN: 0895-4801. DOI: 10.1137/S089548019529248X (cit. on p. 14).

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Baltimore, Md. [u.a.]: Johns Hopkins Univ. Press. Johns Hopkins Studies in the Mathematical Sciences. Baltimore: Johns Hopkins University Press, 2001. 291 pp. ISBN: 978-0-8018-6689-0 (cit. on p. 13).

[Mol12] Moleman, director. *Moleman 2 - Demoscene - The Art of the Algorithms (2012)*. Apr. 6, 2012 (cit. on p. 29).

[Moo65] Gordon E Moore. "Cramming More Components onto Integrated Circuits". In: *Electronics* 38.8 (1965) (cit. on p. 20).

[MS22] Samuel K. Moore and David Schneider. *The State of the Transistor in 3 Charts - IEEE Spectrum*. Nov. 26, 2022. URL: https://spectrum.ieee.org/transistor-density (cit. on p. 20).

[MP58] J. A. Morton and W. J. Pietenpol. "The Technological Impact of Transistors". In: *Proceedings of the IRE* (June 1958), pp. 955–959 (cit. on p. 18).

[Mox] Marc Moxon. *Fully Documented Source Code for Elite on the BBC Micro and NES - Elite on the BBC Micro and NES*. URL: https://www.bbcelite.com/ (cit. on p. 29).

[MMC19] Giovanni Gerardo Muscolo, Giacomo Moretti, and Giorgio Cannata. "SUAS: A Novel Soft Underwater Artificial Skin with Capacitive Transducers and Hyperelastic Membrane". In: *Robotica* 37.4 (Apr. 2019), pp. 756–777. ISSN: 0263-5747, 1469-8668. DOI: 10.1017/S0263574718001315 (cit. on p. 44).

[New82] A. R. Newton. "A Survey of Computer Aids for VLSI Layout". In: *1982 Symposium on VLSI Technology. Digest of Technical Papers*. 1982 Symposium on VLSI Technology. Digest of Technical Papers. Sept. 1982, pp. 72–75 (cit. on p. 22).

[NS87] A.R. Newton and A.L. Sangiovanni-Vincentelli. "CAD Tools for ASIC Design". In: *Proceedings of the IEEE* 75.6 (1987), pp. 765–776. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13798 (cit. on p. 22).

[Nob24] Nobel Prize Outreach AB 2024. *The Nobel Prize in Physics 1956*. NobelPrize.org. Jan. 15, 2024. URL: https://www.nobelprize.org/prizes/physics/1956/summary/ (cit. on p. 17).

[Noy61] Robert N. Noyce. "Semiconductor Device-and-Lead Structure". U.S. pat. 2981877A. Fairchild Semiconductor Corp. Apr. 25, 1961 (cit. on p. 20).

[NXP17] NXP Semiconductors. *SOT23 Package Information*. Jan. 9, 2017. URL: https://www.nxp.com/docs/en/package-information/SOT23.pdf (cit. on p. 19).

[Oh+18] Hyunjoo Oh et al. "PEP (3D Printed Electronic Papercrafts): An Integrated Approach for 3D Sculpting Paper-Based Electronic Devices". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18: CHI Conference on Human Factors in Computing Systems. Montreal QC Canada: ACM, Apr. 21, 2018, pp. 1–12. ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3174015 (cit. on p. 44).

[OWS14] Simon Olberding, Michael Wessely, and Jürgen Steimle. "PrintScreen: Fabricating Highly Customizable Thin-Film Touch-Displays". In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14: The 27th Annual ACM Symposium on User Interface Software and Technology. Honolulu Hawaii USA: ACM, Oct. 5, 2014, pp. 281–290. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647413 (cit. on p. 45).

[Olb+15] Simon Olberding et al. "Foldio: Digital Fabrication of Interactive and Shape-Changing Objects with Foldable Printed Electronics". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15: The 28th Annual ACM Symposium on User Interface Software and Technology. Charlotte NC USA: ACM, Nov. 5, 2015, pp. 223–232. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807494 (cit. on p. 44).

[PM07] David A Papa and Igor L. Markov. "Hypergraph Partitioning and Clustering". In: *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, May 1, 2007. ISBN: 978-1-58488-550-4 (cit. on p. 14).

[Par+21] Luis Paredes et al. "FabHandWear: An End-to-End Pipeline from Design to Fabrication of Customized Functional Hand Wearables". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5.2 (June 23, 2021), pp. 1–22. ISSN: 2474-9567. DOI: 10.1145/3463518 (cit. on p. 44).

[Pen79] R. Penrose. "Pentaplexity A Class of Non-Periodic Tilings of the Plane". In: *The Mathematical Intelligencer* 2.1 (Mar. 1, 1979), pp. 32–37. ISSN: 0343-6993. DOI: 10.1007/BF03024384 (cit. on p. 31).

[PMD21] Pablo Pérez-Gosende, Josefa Mula, and Manuel Díaz-Madroñero. "Facility Layout Planning. An Extended Literature Review". In: *International Journal of Production Research* 59.12 (June 18, 2021), pp. 3777–3816. ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2021.1897176 (cit. on p. 4).

[Per85] Ken Perlin. "An Image Synthesizer". In: *ACM SIGGRAPH Computer Graphics* 19.3 (July 1, 1985), pp. 287–296. ISSN: 0097-8930. DOI: 10.1145/325165.325247 (cit. on p. 30).

[PDS76] G. Persky, D. N. Deutsch, and D. G. Schweikert. "LTX - a System for the Directed Automatic Design of LSI Circuits". In: *The Proceedings of the Thirteenth Design Automation Conference on Design Automation - DAC '76, NO. 13*. The Proceedings of the Thirteenth Design Automation Conference. San Francisco, California, United States: ACM Press, 1976, pp. 399–407. DOI: 10.1145/800146.804840 (cit. on p. 22).

[Pfa72] John L. Pfaltz. "Web Grammars and Picture Description". In: *Computer Graphics and Image Processing* 1.2 (Aug. 1, 1972), pp. 193–220. ISSN: 0146-664X. DOI: 10.1016/S0146-664X(72)80015-7 (cit. on p. 33).

[Pis05] David Pisinger. "Where Are the Hard Knapsack Problems?" In: *Computers & Operations Research* 32.9 (Sept. 2005), pp. 2271–2284. ISSN: 03050548. DOI: 10.1016/j.cor.2004.03.002 (cit. on p. 12).

[Plo+16] Bart Plovie et al. "One-Time Deformable Thermoplastic Devices Based on Flexible Circuit Board Technology". In: *2016 11th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*. 2016 11th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT). Taipei, Taiwan: IEEE, Oct. 2016, pp. 125–128. ISBN: 978-1-5090-4769-7. DOI: 10.1109/IMPACT.2016.7799996 (cit. on p. 45).

[Plo+17] Bart Plovie et al. "Arbitrarily Shaped 2.5D Circuits Using Stretchable Interconnects Embedded in Thermoplastic Polymers". In: *Advanced Engineering Materials* 19.8 (Aug. 2017), p. 1700032. ISSN: 1438-1656, 1527-2648. DOI: 10.1002/adem.201700032 (cit. on pp. 44, 45).

[Pv79] B.T. Preas and W.M. vanCleemput. "Placement Algorithms for Arbitrarily Shaped Blocks". In: *16th Design Automation Conference*. 16th Design Automation Conference. June 1979, pp. 474–480. DOI: 10.1109/DAC.1979.1600152 (cit. on p. 22).

[Pru86] P Prusinkiewicz. "Graphical Applications of L-systems". In: *Proceedings on Graphics Interface '86/Vision Interface '86*. CAN: Canadian Information Processing Society, Aug. 1, 1986, pp. 247–253 (cit. on p. 33).

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. The Virtual Laboratory. New York: Springer-Verlag, 1990. 228 pp. ISBN: 978-0-387-97297-8 978-3-540-97297-6 (cit. on p. 33).

[PLF90] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and F. David Fracchia. "Synthesis of Space-Filling Curves on the Square Grid". In: *Fractals in the fundamental and applied sciences: proceedings of the First IFIP Conference on Fractals in the Fundamental and Applied Sciences, Lisbon, Portugal, 6 - 8 June, 1990* (1990) (cit. on p. 33).

[PSS10] Przemyslaw Prusinkiewicz, Mitra Shirmohammadi, and Faramarz Samavati. "L-Systems in Geometric Modeling". In: *Electronic Proceedings in Theoretical Computer Science* 31 (Aug. 7, 2010), pp. 3–14. ISSN: 2075-2180. DOI: 10.4204/EPTCS.31.3 (cit. on p. 33).

[Pru+03] Przemyslaw Prusinkiewicz et al. "L-System Description of Subdivision Curves". In: *International Journal of Shape Modeling* 09.01 (June 2003), pp. 41–59. ISSN: 0218-6543, 1793-639X. DOI: 10.1142/S0218654303000048 (cit. on p. 33).

[QB10] Jie Qi and Leah Buechley. "Electronic Popables: Exploring Paper-Based Computing through an Interactive Pop-up Book". In: *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '10. New York, NY, USA: Association for Computing Machinery, Jan. 24, 2010, pp. 121–128. ISBN: 978-1-60558-841-4. DOI: 10.1145/1709886.1709909 (cit. on p. 44).

[Qui75] Neil R. Quinn. "The Placement Problem as Viewed from the Physics of Classical Mechanics". In: *Proceedings of the 12th Design Automation Conference*. DAC '75. IEEE Press, Jan. 1, 1975, pp. 173–178 (cit. on p. 22).

[RL22] Yunqing Rao and Qiang Luo. *Intelligent Algorithms for Packing and Cutting Problem*. Vol. 10. Engineering Applications of Computational Methods. Singapore: Springer Nature Singapore, 2022. ISBN: 978-981-19591-5-8 978-981-19591-6-5. DOI: 10.1007/978-981-19-5916-5 (cit. on p. 12).

[Ric+21] Steven I. Rich et al. "Well-Rounded Devices: The Fabrication of Electronics on Curved Surfaces – a Review". In: *Materials Horizons* 8.7 (2021), pp. 1926–1958. ISSN: 2051-6347, 2051-6355. DOI: 10.1039/D1MH00143D (cit. on p. 43).

[RS04] Neil Robertson and P. D. Seymour. "Graph Minors. XX. Wagner's Conjecture". In: *Journal of Combinatorial Theory, Series B*. Special Issue Dedicated to Professor W.T. Tutte 92.2 (Nov. 1, 2004), pp. 325–357. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2004.08.001 (cit. on p. 14).

[RS09] Neil Robertson and P. D. Seymour. "Graph Minors I-XXIII." In: *Journal of Combinatorial Theory, Series B* (2009) (cit. on p. 14).

[Rod+24] Emmanuel Rodriguez et al. "Designing Bending-Active Freeform Surfaces". In: *Proceedings of the 9th ACM Symposium on Computational Fabrication*. SCF '24. New York, NY, USA: Association for Computing Machinery, July 7, 2024, pp. 1–11. ISBN: 9798400704963. DOI: 10.1145/3639473.3665793 (cit. on p. 44).

[Ros72] Azriel Rosenfeld. "Web Automata and Web Grammars". In: *Machine Intelligence* 7 (1972), pp. 307–324 (cit. on p. 33).

[Ros87] Azriel Rosenfeld. "Array Grammars". In: *Graph-Grammars and Their Application to Computer Science*. Ed. by Hartmut Ehrig et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 67–70. ISBN: 978-3-540-48178-2 (cit. on p. 33).

[RL] Jessica Rosenkrantz and Jesse Louis-Rosenberg. *Nervous System*. URL: https://n-e-r-v-o-u-s.com/about_us.php (cit. on pp. 3, 127).

[Rus+22] Stuart J. Russell et al. *Artificial Intelligence: A Modern Approach*. Fourth edition, global edition. Pearson Series in Artificial Intelligence. Harlow: Pearson, 2022. 1166 pp. ISBN: 978-1-292-40113-3 (cit. on pp. 79, 81).

[Rus+11] Analisa Russo et al. "Pen-on-Paper Flexible Electronics". In: *Advanced Materials* 23.30 (2011), pp. 3426–3430. ISSN: 1521-4095. DOI: 10.1002/adma.201101328 (cit. on pp. 44, 45).

[SB80] S. Sahni and A. Bhatt. "The Complexity of Design Automation Problems". In: *17th Design Automation Conference*. 17th Design Automation Conference. June 1980, pp. 402–411. DOI: 10.1145/800139.804562 (cit. on p. 26).

[SY01] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice*. Repr. Lecture Notes Series on Computing 6. Singapore: World Scientific, 2001. 482 pp. ISBN: 978-981-02-3883-4 (cit. on pp. 25, 26).

[San03] A. Sangiovanni-Vincentelli. "The Tides of EDA". In: *IEEE Design & Test of Computers* 20.6 (Nov. 2003), pp. 59–75. ISSN: 1558-1918. DOI: 10.1109/MDT.2003.1246165 (cit. on pp. 22, 24).

[Sat17] Sam Sattel. *Routing & Autorouting - PCB Layout Basics 2 | EAGLE | Blog*. Fusion Blog. May 3, 2017. URL: https://www.autodesk.com/products/fusion-360/blog/routing-autorouting-pcb-layout-basics-2/ (cit. on p. 28).

[Sch97] Wolfram Schlickenrieder. "Nets of Polyhedra". 1997 (cit. on p. 45).

[Sch13] Ryan Schmidt. *Support Structures (Blog Entry)*. 2013. URL: http://www.rms80.com/supports (cit. on p. 80).

[SU14] Ryan Schmidt and Nobuyuki Umetani. "Branching Support Structures for 3D Printing". In: *ACM SIGGRAPH 2014 Studio*. SIGGRAPH '14. New York, NY, USA: Association for Computing Machinery, 2014. ISBN: 978-1-4503-2977-4. DOI: 10/ghtq3g (cit. on pp. 78, 80).

[Sem97] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*. 1997 (cit. on p. 28).

[She75] G. C. Shephard. "Convex Polytopes with Convex Nets". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 78.3 (Nov. 1975), pp. 389–403. ISSN: 0305-0041, 1469-8064. DOI: 10.1017/S0305004100051860 (cit. on p. 45).

[She04] Naveed A. Sherwani. *Algorithms for VLSI Physical Design Automation*. 3. ed., 6. print. Boston, Mass.: Kluwer Academic, 2004. 572 pp. ISBN: 978-0-7923-8393-2 (cit. on p. 25).

[Sme+14] Ruben M. Smelik et al. "A Survey on Procedural Modelling for Virtual Worlds". In: *Computer Graphics Forum* 33.6 (Sept. 2014), pp. 31–50. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12276 (cit. on pp. 30, 33, 35).

[SPS04] Colin Smith, Przemyslaw Prusinkiewicz, and Faramarz Samavati. "Local Specification of Surface Subdivision Algorithms". In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by John L. Pfaltz, Manfred Nagl, and Boris Böhlen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 313–327. ISBN: 978-3-540-25959-6. DOI: 10.1007/978-3-540-25959-6_23 (cit. on p. 33).

[Smi+23a] David Smith et al. *A Chiral Aperiodic Monotile*. May 28, 2023. DOI: 10.48550/arXiv.2305.17743. arXiv: 2305.17743 [cs, math]. URL: http://arxiv.org/abs/2305.17743. Pre-published (cit. on p. 12).

[Smi+23b] David Smith et al. *An Aperiodic Monotile*. May 29, 2023. DOI: 10.48550/arXiv.2303.10798. arXiv: 2303.10798 [cs, math]. URL: http://arxiv.org/abs/2303.10798. Pre-published (cit. on p. 12).

[SLK05] Kaleigh Smith, Yunjun Liu, and Allison Klein. "Animosaics". In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA05: Symposium on Computer Animation. Los Angeles California: ACM, July 29, 2005, pp. 201–208. ISBN: 978-1-59593-198-6. DOI: 10.1145/1073368.1073397 (cit. on p. 46).

[SK24] Nils Speetzen and Leif Kobbelt. "Freeform Shape Fabrication by Kerfing Stiff Materials". In: (2024). ISSN: 1467-8659. DOI: 10.1111/cgf.15032 (cit. on p. 44).

[Sta13] Gary S. Stager. "Papert's Prison Fab Lab: Implications for the Maker Movement and Education Design". In: *Proceedings of the 12th International Conference on Interaction Design and Children*. IDC '13. New York, NY, USA: Association for Computing Machinery, June 24, 2013, pp. 487–490. ISBN: 978-1-4503-1918-8. DOI: 10.1145/2485760.2485811 (cit. on p. 3).

[Sta97] Jos Stam. *Aperiodic Texture Mapping*. R046. European Research Consortium for Informatics and Mathematics, Jan. 1997 (cit. on p. 31).

[SG71] G. Stiny and J. Gips. "Shape Grammars and the Generative Specification of Painting and Sculpture". In: IFIP Congress. 1971 (cit. on p. 33).

[Sti75] George Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars*. Basel: Birkhäuser Basel, 1975. ISBN: 978-3-7643-0803-2 978-3-0348-6879-2. DOI: 10.1007/978-3-0348-6879-2 (cit. on p. 33).

[SP11] Raphael Straub and Hartmut Prautzsch. *Creating Optimized Cut-out Sheets for Paper Models from Meshes*. 2011 (cit. on p. 45).

[Swa+19] Saiganesh Swaminathan et al. "FiberWire: Embedding Electronic Function into 3D Printed Mechanically Strong, Lightweight Carbon Fiber Composite Objects". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19: CHI Conference on Human Factors in Computing Systems. Glasgow Scotland Uk: ACM, May 2, 2019, pp. 1–11. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300797 (cit. on p. 44).

[Tak+11] Shigeo Takahashi et al. "Optimized Topological Surgery for Unfolding 3D Meshes". In: *Computer Graphics Forum* 30.7 (Sept. 2011), pp. 2077–2086. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/j.1467-8659.2011.02053.x (cit. on pp. 45, 49).

[Tam13] Roberto Tamassia. *Handbook of Graph Drawing and Visualization*. CRC Press, Aug. 19, 2013. 869 pp. ISBN: 978-1-58488-412-5. DOI: 10.1201/b15385. Google Books: lQBrAAAAQBAJ (cit. on p. 13).

[TBS18] Edna Tan, Angela Calabrese Barton, and Kathleen Schenkel. "Methods and Strategies: Equity and the Maker Movement". In: *Science and Children* 55.7 (Mar. 2018), pp. 76–81. ISSN: 0036-8148, 1943-4812. DOI: 10.2505/4/sc18_055_07_76 (cit. on p. 3).

[Tas04] Lassi Tasajärvi, ed. *Demoscene: The Art of Real-Time*. Trans. by Arttu Tolonen. Even Lake Studios, Katastro.fi, 2004. ISBN: 952-91-7022-X (cit. on p. 29).

[Ter03] Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science 55. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-39115-3 (cit. on p. 32).

[Tho89] Carsten Thomassen. "The Graph Genus Problem Is NP-complete". In: *Journal of Algorithms* 10.4 (Dec. 1, 1989), pp. 568–576. ISSN: 0196-6774. DOI: 10.1016/0196-6774(89)90006-0 (cit. on p. 14).

[Tok+88] T. Tokuda et al. "A Macrocell Approach for VLSI Processor Design". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7.12 (Dec. 1988), pp. 1272–1277. ISSN: 02780070. DOI: 10.1109/43.16805 (cit. on p. 23).

[Tor+13] J J Toriz-Garcia et al. "Fabrication of a 3D Electrically Small Antenna Using Holographic Photolithography". In: *Journal of Micromechanics and Microengineering* 23.5 (May 1, 2013), p. 055010. ISSN: 0960-1317, 1361-6439. DOI: 10.1088/0960-1317/23/5/055010 (cit. on p. 44).

[Tor+17] Cesar Torres et al. "Illumination Aesthetics: Light as a Creative Material within Computational Design". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17: CHI Conference on Human Factors in Computing Systems. Denver Colorado USA: ACM, May 2, 2017, pp. 6111–6122. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025466 (cit. on p. 45).

[TOG17] Csaba Tóth, Joseph O'Rourke, and Jacob E. Goodman, eds. *Handbook of Discrete and Computational Geometry*. Third edition. Boca Raton: CRC Press, 2017. ISBN: 978-1-4987-1139-5 (cit. on p. 12).

[TWZ22] Peihan Tu, Li-Yi Wei, and Matthias Zwicker. "Clustered Vector Textures". In: *ACM Transactions on Graphics* 41.4 (July 22, 2022), 159:1–159:23. ISSN: 0730-0301. DOI: 10.1145/3528223.3530062 (cit. on p. 30).

[US17] N. Umetani and R. Schmidt. "SurfCuit: Surface-Mounted Circuits on 3D Prints". In: *IEEE Computer Graphics and Applications* 37.3 (May 2017), pp. 52–60. ISSN: 1558-1756. DOI: 10/ghgdxc (cit. on p. 44).

[VGB14] J. Vanek, J. A. G. Galicia, and B. Benes. "Clever Support: Efficient Support Structure Generation for Digital Fabrication". In: *Computer Graphics Forum* 33.5 (2014), pp. 117–125. ISSN: 1467-8659. DOI: 10/f6f3rb (cit. on p. 80).

[Wag37] K. Wagner. "Über eine Eigenschaft der ebenen Komplexe". In: *Mathematische Annalen* 114.1 (Dec. 1937), pp. 570–590. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01594196 (cit. on p. 13).

[Wan+20] Guanyun Wang et al. "MorphingCircuit: An Integrated Design, Simulation, and Fabrication Workflow for Self-Morphing Electronics". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.4 (Dec. 17, 2020), pp. 1–26. ISSN: 2474-9567. DOI: 10.1145/3432232 (cit. on pp. 5, 44).

[Wan61] Hao Wang. "Proving Theorems by Pattern Recognition — II". In: *The Bell System Technical Journal* 40.1 (Jan. 1961), pp. 1–41. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1961.tb03975.x (cit. on p. 31).

[Wan65] Hao Wang. "Games, Logic and Computers". In: *Scientific American* 213.5 (Nov. 1965), pp. 98–106. ISSN: 0036-8733. DOI: 10.1038/scientificamerican1165-98 (cit. on p. 31).

[Wan+18] Tianyi Wang et al. "Plain2Fun: Augmenting Ordinary Objects with Surface Painted Circuits". In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18: CHI Conference on Human Factors in Computing Systems. Montreal QC Canada: ACM, Apr. 20, 2018, pp. 1–6. ISBN: 978-1-4503-5621-3. DOI: 10.1145/3170427.3188655 (cit. on p. 44).

[WHS07] Gerhard Wäscher, Heike Haußner, and Holger Schumann. "An Improved Typology of Cutting and Packing Problems". In: *European Journal of Operational Research* 183.3 (Dec. 2007), pp. 1109–1130. ISSN: 03772217. DOI: 10.1016/j.ejor.2005.12.047 (cit. on p. 12).

[WMR22] Ramon Elias Weber, Caitlin Mueller, and Christoph Reinhart. "Automated Floorplan Generation in Architectural Design: A Review of Methods and Applications". In: *Automation in Construction* 140 (Aug. 1, 2022), p. 104385. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2022.104385 (cit. on pp. 4, 127).

[Wei+09] Li-Yi Wei et al. "State of the Art in Example-based Texture Synthesis". In: *Eurographics 2009 - State of the Art Reports* (2009), 25 pages. ISSN: 1017-4656. DOI: 10.2312/EGST.20091063 (cit. on p. 30).

[Whi01] Arthur T. White. *Graphs of Groups on Surfaces: Interactions and Models*. 1st ed. North-Holland Mathematics Studies 188. Amsterdam ; New York: Elsevier, 2001. 363 pp. ISBN: 978-0-444-50075-5 (cit. on p. 13).

[Won+03] Peter Wonka et al. "Instant Architecture". In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. New York, NY, USA: Association for Computing Machinery, July 1, 2003, pp. 669–677. ISBN: 978-1-58113-709-5. DOI: 10.1145/1201775.882324 (cit. on p. 33).

[Wu+20] Hao Wu et al. "Fabrication Techniques for Curved Electronics on Arbitrary Surfaces". In: *Advanced Materials Technologies* 5.8 (Aug. 2020), p. 2000093. ISSN: 2365-709X, 2365-709X. DOI: 10.1002/admt.202000093 (cit. on p. 43).

[Wu+22] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Nature Singapore, 2022. ISBN: 9789811660535 9789811660542. DOI: 10.1007/978-981-16-6054-2 (cit. on p. 15).

[Xu+20] Hao Xu et al. "TilinGNN: Learning to Tile with Self-Supervised Graph Neural Network". In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392380 (cit. on pp. 46, 57).

[Yam+19] Junichi Yamaoka et al. "FoldTronics: Creating 3D Objects with Integrated Electronics Using Foldable Honeycomb Structures". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19: CHI Conference on Human Factors in Computing Systems. Glasgow Scotland Uk: ACM, May 2, 2019, pp. 1–14. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300858 (cit. on p. 44).

[Yan+22] Zeyu Yan et al. "Fibercuit: Prototyping High-Resolution Flexible and Kirigami Circuits with a Fiber Laser Engraver". In: *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. UIST '22: The 35th Annual ACM Symposium on User Interface Software and Technology. Bend OR USA: ACM, Oct. 29, 2022, pp. 1–13. ISBN: 978-1-4503-9320-1. DOI: 10.1145/3526113.3545652 (cit. on p. 44).

[Yu16] Derek Yu. *Spelunky*. Los Angeles, CA : Boss Fight Books, 2016. 230 pp. ISBN: 978-1-940535-11-1 978-1-940535-63-0 (cit. on pp. 4, 128).

[Zha+20] Allan Zhao et al. "RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design". In: *ACM Transactions on Graphics* 39.6 (Dec. 31, 2020), pp. 1–16. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3414685.3417831 (cit. on p. 33).

[ZJ16] Qingnan Zhou and Alec Jacobson. *Thingi10K: A Dataset of 10,000 3D-printing Models*. July 1, 2016. arXiv: 1605.04797 [cs]. URL: http://arxiv.org/abs/1605.04797. Pre-published (cit. on p. 95).

[ZJL14] Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. "Topology-Constrained Synthesis of Vector Patterns". In: *ACM Transactions on Graphics* 33.6 (Nov. 19, 2014), 215:1–215:11. ISSN: 0730-0301. DOI: 10.1145/2661229.2661238 (cit. on p. 32).

[Zhu+20a] Junyi Zhu et al. "CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20: CHI Conference on Human Factors in Computing Systems. Honolulu HI USA: ACM, Apr. 21, 2020, pp. 1–13. ISBN: 978-1-4503-6708-0. DOI: 10/ghhr54 (cit. on p. 44).

[Zhu+20b] Junyi Zhu et al. "MorphSensor: A 3D Electronic Design Tool for Reforming Sensor Modules". In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. UIST '20: The 33rd Annual ACM Symposium on User Interface Software and Technology. Virtual Event USA: ACM, Oct. 20, 2020, pp. 541–553. ISBN: 978-1-4503-7514-6. DOI: 10.1145/3379337.3415898 (cit. on p. 44).

Résumé long

Cette thèse traite sur des problèmes d'agencement à l'intersection de plusieurs domaines, chacun ayant une histoire riche et des applications diverses. En particulier, ces domaines sont la conception et la fabrication computationnelles et l'informatique graphique.

La démocratisation des techniques de fabrication auparavant exclusivement industrielles telles que l'impression 3D ou la fabrication de circuits électroniques ont rapproché ces procédés d'un public dépourvu d'une expertise spécialisée. Cette démocratisation résulte en partie de l'accessibilité des outils de fabrication, à travers par exemple l'apparition d'imprimantes 3D abordables, et de services en ligne de fabrication de circuits électroniques. Ce nouvel accès passe aussi par le développement d'outils numériques pour faciliter les tâches de conception et modélisation et ainsi baisser significativement la barrière d'entrée. Cet accès, porté par le développement d'outils abordables et de logiciels faciles d'utilisation, ont permis l'apparition et la croissance de communautés de passionnés et de *makers* (ou *faiseurs*). Celles-ci s'organisent en ligne et autour d'espaces communautaires proposant un accès à des ordinateurs, des outils d'atelier et des personnes compétentes en leur utilisation voulant partager leurs connaissances. Ces communautés contribuent aussi à rendre ces technologies toujours plus accessibles, et permettent à encore plus de personnes d'explorer leurs possibilités créatives. Cet intérêt croissant en ces technologies et la possibilité de construire des prototyes rapidement ont été accompagnées par des avancées importantes en informatique graphique et en fabrication computationnelle dans le domaine de la *Conception pour la Fabrication* [BFR17]. L'accessibilité nouvelle à ces technologies contraste cependant avec la complexité des processus eux-mêmes, qui doit être prise en compte par les chercheurs et développeurs des nouvelles méthodes dans ce domaine. Un élément clé du développement de ces outils est l'essor et la présence croissante des ordinateurs dans les dernières décennies. En effet, l'omniprésence de l'ordinateur et la digitalisation de tâches traditionellement analogiques ont provoqué des changements radicaux dans les méthodologies de conception. L'outil informatique permet l'automatisation de ces processus, les rendant plus efficientes en temps et en ressources. Grâce à leur puissance de calcul, il rend possible le traitement d'instances beaucoup plus grandes, qu'il aurait été inconcevable de traiter sans, ou qui auraient requis une quantité de main-d'œvre et d'effort disproportionnée. Le développement d'outils pour assister ces tâches est le but de la *Conception Assistée par Ordinateur*.

L'outil informatique permet aussi le développement de représentations spécifiques au milieu numérique. Par exemple, le paradigme de la *Conception Générative ou Paramétrique* [dBoi22] est irréalisable en absence d'ordinateurs. Cette approche consiste en la définition simultanée de familles entières de formes, définies à travers une représentation procédurale ou programmatique, où des formes

particulières peuvent être sélectionnées en donnant des valeurs particulières à des paramètres sémantiques. En d'autres mots, la forme est définie non pas directement en spécifiant sa géométrie exacte, mais à travers un assemblage de formes simples (ou primitives) définies par du code, prenant leur forme exacte à travers de paramètres. Ces paramètres définissent la forme exacte des primitives et comment elles s'assemblent, par exemple contrôlant la longueur d'un élément, ou le placement relatif de différents composants de la forme définie. Cette approche trouve des applications notamment en architecture [CSL20], où ces représentations flexibles permettent d'adapter des plans à des environnements complexes, ou bien de trouver des formes précises répondant au cahier de charges. Ce type de représentation a été utilisé par exemple dans la conception du Terminal International de Waterloo à Londres [GJ12], ou bien par le studio *Nervous System* [RL] pour générer une infinité de formes différentes à partir d'un seul concept artistique. Une partie essentielle de ce changement de méthodologie passe par le développement de nouvelles représentations de données et de techniques d'optimisation, capables de traiter de grandes quantités d'informations efficacement. L'*Informatique Graphique* est une branche de l'informatique qui, parmi d'autres, étudie le traitement, la simulation et la génération de données géométriques, physiques et visuelles telles que des maillages ou des textures pour la génération d'images de synthèse. Les données utilisées peuvent être gigantesques, de l'ordre de millions de triangles pour des maillages et des millions de pixels pour des images, demandant ainsi un travail poussé sur la représentation et le traitement efficaces de ces données. Il semble donc logique de réinvestir des algorithmes issus de l'informatique graphique pour la conception et la fabrication computationnelles.

Les problèmes d'agencement surviennent dans de nombreux contextes en ingénieurie et en informatique. Typiquement, la résolution d'un problème d'agencement consiste en l'organisation spatiale et l'interconnexion d'un ensemble d'éléments ou objets dans un espace. Cet espace et ces interconnexions peuvent être de complexité très variable, allant d'un simple rectangle où des objets sont reliés par des segments de droite à des emboîtements tridimensionnels d'objets de formes complexe. Un ensemble de contraintes et d'objectifs complémentent souvent la description du problème, tels que minimiser la longueur ou la surface des interconnexions ou fixer la position de certains éléments. Ces objectifs et contraintes sont souvent spécifiques à chacun des problèmes en question et sont hautement influencés par le domaine dans lequel le problème s'inscrit. De même, les enjeux sont très différents en fonction de l'application. La planification des étages en architecture, l'agencement d'installations industrielles, de circuits électroniques, ou la création de niveaux de jeux vidéo, sont tous des exemples de problèmes d'agencement. La planification d'étages en architecture [WMR22] regroupe la décision des éléments intérieurs d'un bâtiment étant donnée la forme extérieure de celui-ci. Ceci regroupe par exemple le placement de murs intérieurs et de portes afin de créer des pièces distinctes ou

l'attribution de rôles spécifiques à certaines pièces (pour une maison : cuisine, chambre, salle de bains, salon...), certaines devant être obligatoirement séparées ou connectées. Dans la planification de bâtiments publics, il peut y avoir en même temps des contraintes structurelles, desquelles dépend la stabilité du bâtiment et la sécurité des usagers ; et des objectifs visant à rendre facilement accessibles les zones hautement transitées. Un exemple à enjeux plus élevés est la planification des installations industrielles, où les différents éléments d'un système de production doivent être correctement placés dans l'installation afin d'optimiser l'utilisation de l'espace, le coût de construction ou bien le temps de transport entre éléments entre autres. De la solution obtenue dépend le bon fonctionnement du système, la méthode suivie peut donc faire la différence entre une installation industrielle productive et rentable et une installation inefficace. Un domaine d'application diamétralement opposé est la génération de niveaux de jeux vidéo, elle-même un sous-domaine de la *Génération Procédurale de Contenu*, souvent utilisée dans des jeux *rogue-like* tels que Spelunky [Yu16]. Ces techniques sont utilisés pour synthétiser des environnements complexes dans lesquels le joueur peut évoluer en facilitant la tâche du designer, requérant par exemple uniquement un graphe de connectivité et un ensemble possible de salles [Ma+14]. Les enjeux sont moins élevés, mais il est quand même difficile de trouver le bon équilibre entre la génération automatique et le contenu original créé par un designer. Trop de contenu synthétisé rend les niveaux répétitifs et peu originaux, mais trop de contenu original demande énormément de travail [Gam16]. Les contraintes et objectifs peuvent aussi être complexes. Par exemple, il est important d'éviter des sections totalement inaccessibles au joueur, ou d'assurer que la fin du niveau est accessible depuis sont début. Un designer peut vouloir établir des prérequis pour accéder à certaines parties du monde, pour par exemple forcer le joueur à visiter les niveaux dans un certain ordre.

Les contraintes topologiques jouent un rôle important dans l'agencement. La topologie considère des objets définis par les voisinages de leurs éléments, sans s'attarder sur leur géométrie spécifique. Comme raconte la blague : "un topologiste est une personne qui ne sait pas faire la différence entre une tasse et un donut". Par exemple, un graphe est une entité topologique, constituée uniquement des liens entre ses nœuds. Au contraire, dessiner un graphe est une opération géométrique, puisqu'elle demande de spécifier la position des nœuds. Dans le contexte des problèmes d'agencement, les contraintes topologiques portent sur les relations entre les objets. Par exemple, dans l'agencement de circuits électroniques, c'est-à-dire le placement et le routage (interconnexion) des composants électroniques dans le circuit imprimé ou intégré, l'information topologique est fournie par le schéma électrique. Dans l'agencement d'un appartement, une contrainte topologique pourrait porter sur l'adjacence de certaines pièces : la cuisine et la salle à manger, une salle de bains et une chambre. Souvent il est intéressant de considérer des contraintes topologiques locales, indiquant à quoi ressemble notre agencement 'de

près', et d'essayer de générer un agencement global les satisfaisant.

Cette thèse se focalise sur la résolution de deux problèmes d'agencement spécifiques liés à la fabrication et la conception computationnelles sujets à des contraintes topologiques. Plus particulièrement, il s'agit de la génération d'agencements de circuits électroniques et la génération de supports pour l'impression 3D.

La première contribution est un système pour la conception d'écrans surfaciques constitués de DEL RVB à travers l'utilisation de circuits imprimés pliables. Normalement les circuits électroniques sont intégrés sur des cartes de circuit imprimé, essentiellement constituant un sandwich de couches conductrices et isolantes. Des composants peuvent ensuite être soudés sur ces cartes, et les connexions entre ces composants sont gravées dans les couches conductrices. Cette approche permet de maximiser la densité des composants sur la carte. De plus, il existe des services de fabrication de circuits électroniques en ligne abordables, permettant d'envoyer des plans de fabrication et de recevoir les circuits fabriqués avec tous les composants soudés quelques semaines plus tard. Il n'a jamais été aussi facile de concevoir des circuits électroniques en partant de zéro, grâce à des outlis de *conception assistée par ordinateur pour l'électronique* tels que *KiCad* [CK92], et aux services de fabrication mentionnés ci-dessus. Il existe aussi des cartes de circuits imprimés flexibles, souvent utilisées dans des appareils restrictifs en termes d'espace, tels que des ordinateurs ou téléphones portables. Ce type de carte est significativement plus chère que les cartes rigides, et sont plus difficiles à concevoir et fabriquer. Afin d'obtenir des cartes imprimées flexibles, nous suivons une approche différente. Nous plions les circuits imprimés traditionnels en utilisant des motifs de découpe localisés, créant ainsi des 'charnières' dans la plaque. Cette technique est inspirée de la découpe du bois (*wood kerfing* en anglais), permettant de créer en retirant de la matière des surfaces et formes courbes autrement difficiles à réaliser. Le système prend en entrée un maillage basse-résolution et produit des plans pouvant être envoyés à des services en ligne de fabrication de circuits. Suite à la fabrication, l'écran est assemblé en pliant le circuit sur une impression 3D du maillage d'origine. Les écrans fabriqués peuvent être contrôlés à travers une interface programmatique pour créer des effets lumineux impressionnants en temps réel. Le problème global est découpé en sous-problèmes locaux grâce à la topologie chaînée du circuit, les plans finaux étant obtenus en 'recousant' les solutions aux sous-problèmes. Au lieu de suivre la méthode traditionelle d'agencement électronique (concevoir le schéma électrique, placer et connecter les composants) ; nous décidons du nombre de composants, leur placement et leur routage séparément pour chaque triangle au moment-même de la génération.

La deuxième contribution est un algorithme procédural pour la génération de supports pour l'impression 3D sous forme d'échafaudage. Les technologies de fabrication additive et impression 3D sont devenues très accessibles au grand public dans les dernières décennies. La facilité d'utilisation d'imprimantes grand public, l'ap-

parition de services en ligne d'impression 3D, et la grande diversité de modèles 3D disponibles sur des plateformes de partage rendent plus facile que jamais de rentrer dans cet espace. Une majorité de méthodes de fabrication additive produisent les objets couche par couche, de bas en haut. Ceci rend souvent nécessaire l'utilisation de structures de support pour l'impression de zones surplombantes. L'importance des supports pour l'impression 3D ne doit pas être sous-estimée. Ils permettent la fabrication d'objets ayant des géométries particulièrement complexes, mais doivent ensuite être enlevés à la main. De plus, ils augmentent la quantité de matière utilisée et donc le temps d'impression. Ils laissent aussi des traces visibles sur la surface des objets fabriqués une fois enlevés. Concevoir des structures de support adaptées et les optimiser est donc un élément clé de la recherche en fabrication additive. Les supports générés par notre méthode s'impriment de manière fiable et sont stables [DHL14]. L'algorithme précédent ne considère pas les intersections entre les supports et l'objet imprimé, laissant des marques indésirables sur la surface de l'objet. De plus, la complexité de l'algorithme dépend du nombre de points à porter. Nous proposons un nouvel algorithme inspiré du *Model Synthesis* (MS) [Mer09]. Il évite implicitement les intersections et sa complexité est indépendante du nombre de points à porter. Les supports sont représentés indirectement à travers un ensemble d'étiquettes, chacune représentant une partie de la structure (par exemple une partie de pilier, de pont, ou une jonction) ; et un ensemble de contraintes d'adjacence déterminant quelles combinaisons d'étiquettes sont possibles dans toutes les directions. Les supports sont générés de haut en bas en attribuant de façon répétée une étiquette à un voxel, puis en propageant les contraintes afin d'éliminer les étiquettes rendues impossibles. Cet algorithme, les contraintes d'adjacences et les heuristiques utilisées sont conçues ensemble pour générer des supports sans essai-erreur ou retours arrière, typiques du MS et autres méthodes similaires.

Ces problèmes semblent au premier abord très différents, mais peuvent en fait être vus à travers le même prisme. Dans les deux cas nous traitons des problèmes d'agencement soumis à des contraintes topologiques définissant les types de solution admissible. Dans le premier cas il s'agit de circuits constitués de boucles de DEL, et dans le deuxième cas il s'agit de structures en échafaudage, constituées de barres horizontales et verticales interconnectées. Ces contraintes peuvent être exploitées d'un point de vue algorithmique et computationnel : pour limiter l'espace de recherche de solutions, ou bien pour extraire des propriétés des agencements valables, entre autres. Nos méthodes peuvent ensuite générer des plans de fabrication, c'est-à-dire des réalisations géométriques prenant en compte les technologies de fabrication visées, à partir des solutions obtenues.

Le but de cette thèse est de fournir une ressource utile pour la résolution de problèmes d'agencement pour la fabrication computationnelle, en mettant l'accent sur les contraintes topologiques et comment elles peuvent être exploitées pour simplifier ces problèmes.

Lay summary

Layout problems arise in many engineering and computing domains. They consist of a set of objects, a space and a set of constraints and objectives. Finding a valid layout means placing the objects within the space while satisfying the constraints and optimizing the objectives. Electronics physical layout design and floorplanning are examples of layout problems. Often these include topological constraints, i.e. constraints relating to relationships between the objects, such as components being connected in electronic circuits or two rooms being non-adjacent in a floorplan. This thesis considers layout problems with topological constraints in computational fabrication, more specifically in the context of electronic circuit layout and support structures for 3D printing. These technologies being more accessible than ever thanks to affordable 3D printers and online electronic prototyping services means that developing techniques for them is more important than ever.

Résumé vulgarisé

Les problèmes d'agencement surviennent dans de nombreux domaines en ingénieurie et en informatique. Ils sont constitués d'objets, d'un espace, de contraintes et d'objectifs. Un agencement valide est un placement des objets dans l'espace respectant les contraintes et optimisant les objectifs. La conception de circuits électroniques et la planification architecturale sont des exemples de problèmes d'agencement. Ceux-ci impliquent souvent des contraintes topologiques portant sur les relations entre les objets, comme des composants électroniques étant connectés ou des salles n'étant pas contigües sur un plan. Cette thèse s'intéresse aux problèmes d'agencement avec des contraintes topologiques dans la fabrication computationnelle, plus précisément pour l'agencement de circuits électroniques et la génération de supports pour l'impression 3D. Ces technologies sont plus accessibles que jamais grâce à l'existence d'imprimantes 3D abordables et de services en ligne de fabrication de circuits.