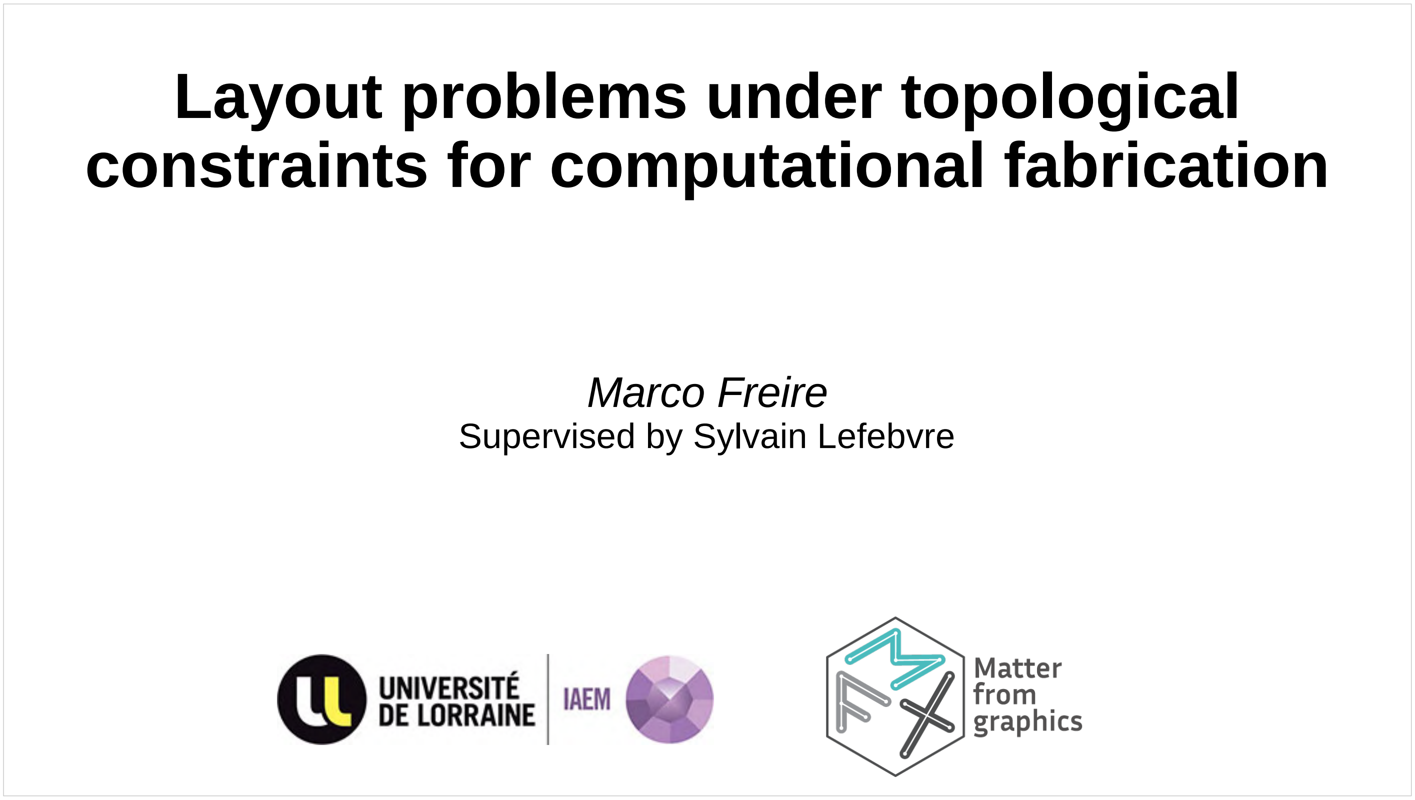


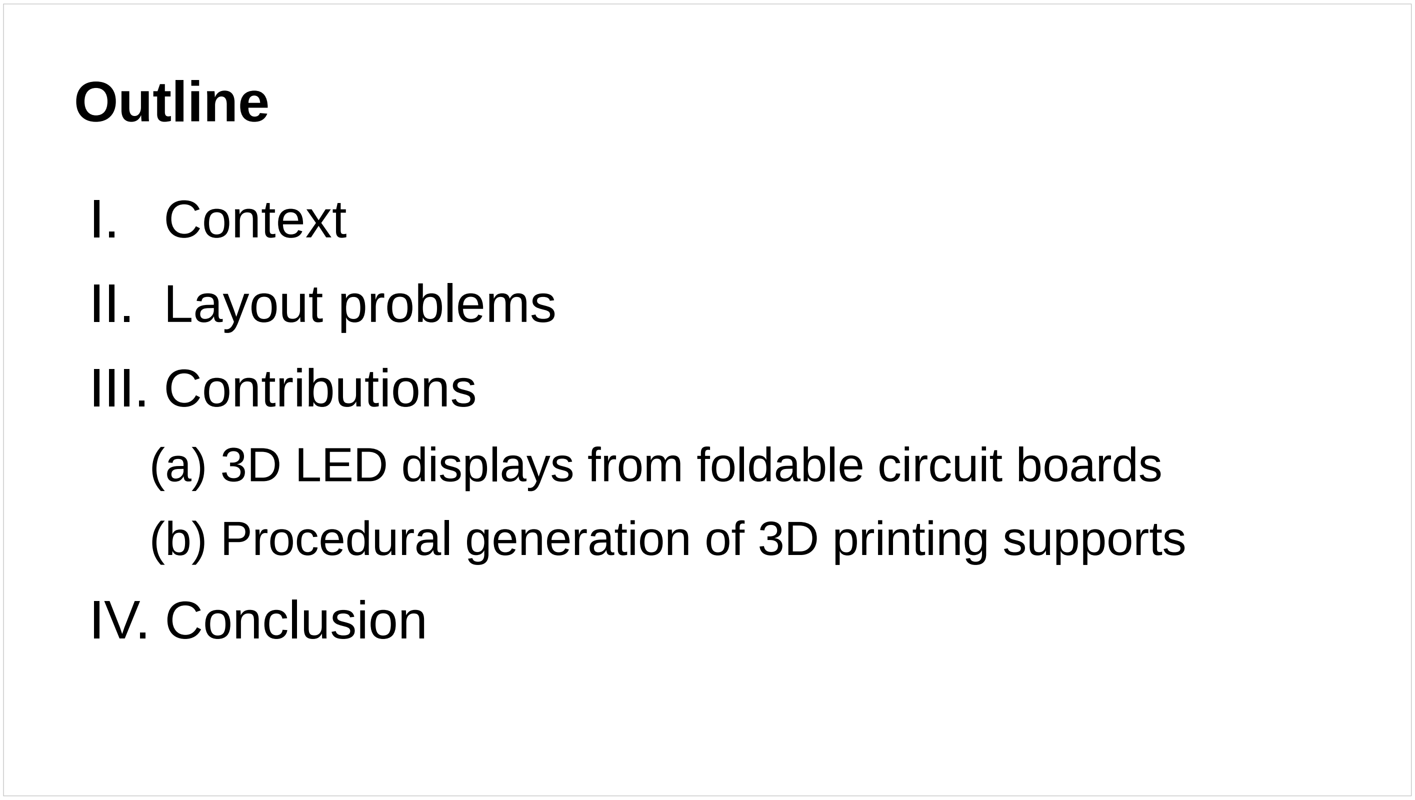
Layout problems under topological constraints for computational fabrication

Marco Freire
Supervised by Sylvain Lefebvre

UNIVERSITÉ DE LORRAINE | IAEM
Matter from graphics

Hi everyone, thank you all for assisting today to my PhD defense on the topic of *Layout problems under topological constraints for computational fabrication*.

This work started in October 2020, and was supervised all along by Sylvain Lefebvre, lead researcher of the Matter From Graphics team.



Outline

I. Context

II. Layout problems

III. Contributions

(a) 3D LED displays from foldable circuit boards

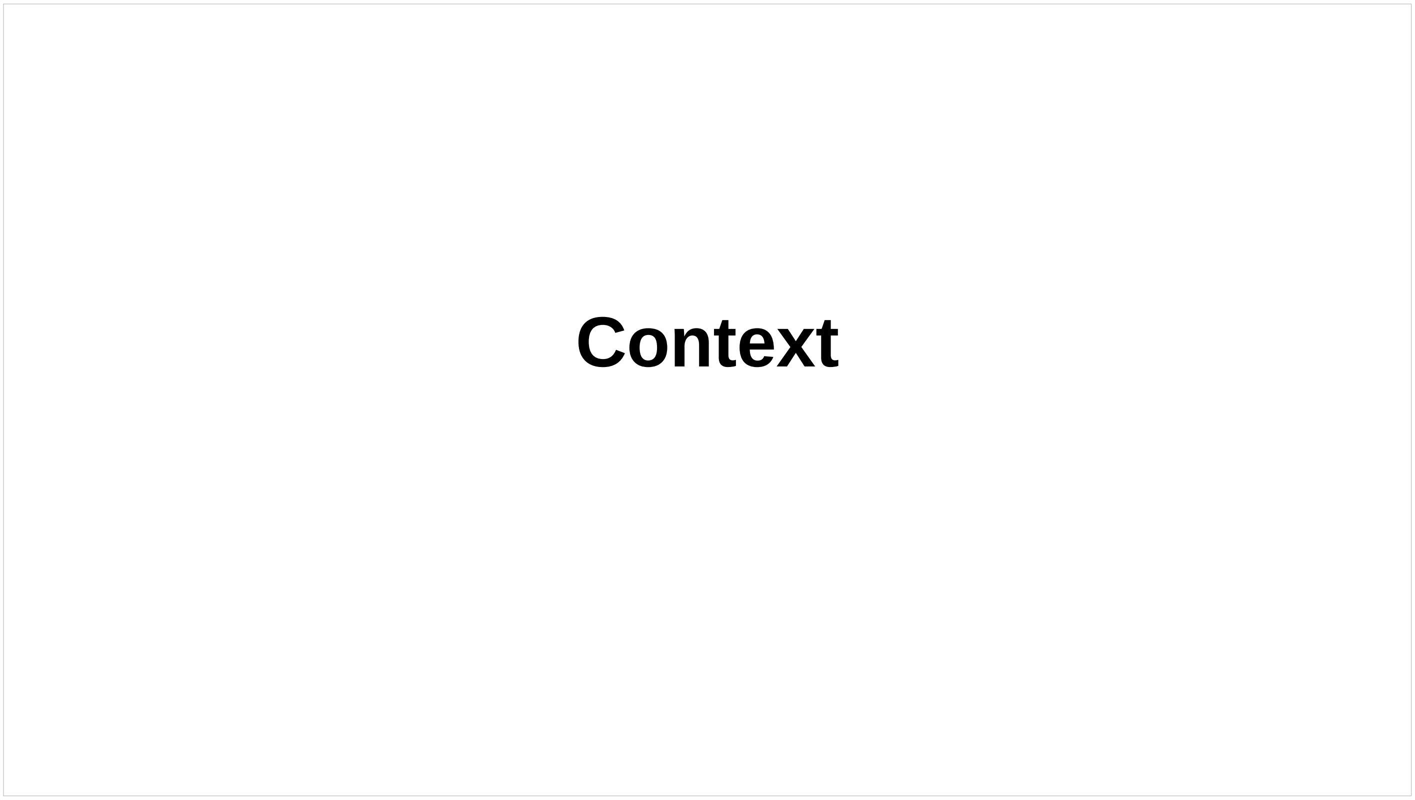
(b) Procedural generation of 3D printing supports

IV. Conclusion

This presentation will be structured as shown here.

I will first provide basic context on the main fields relevant to my thesis. Next, I will introduce layout problems, specifically under topological constraints, which are the core of my work.

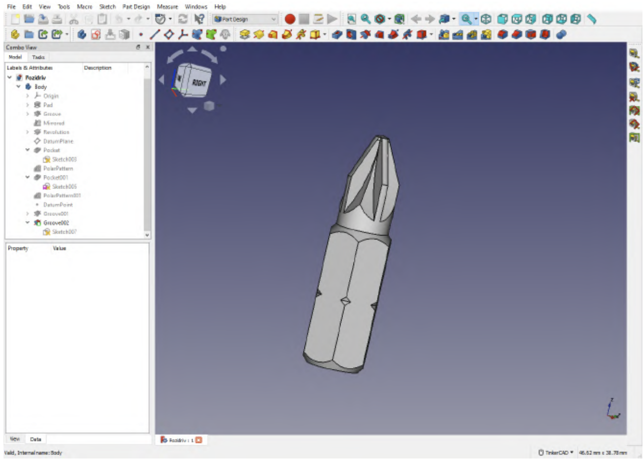
Then I will focus on the contributions accomplished during my PhD, and I will conclude with the perspectives gathered during this time.



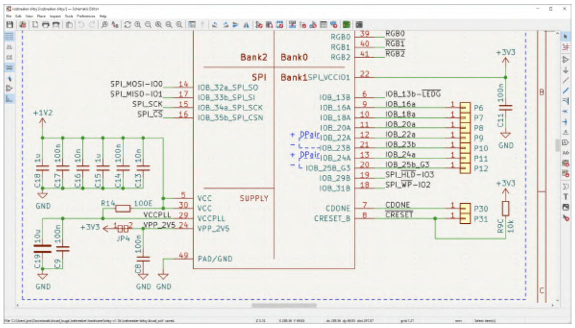
Context

Let’s start with a bit of context!

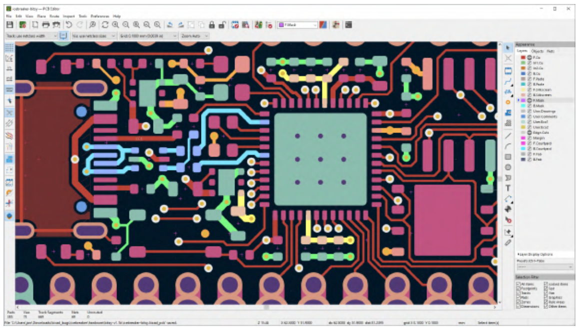
Computational design



FreeCAD



KiCad



During these years I have worked at the intersection of computational design, fabrication and computer graphics.

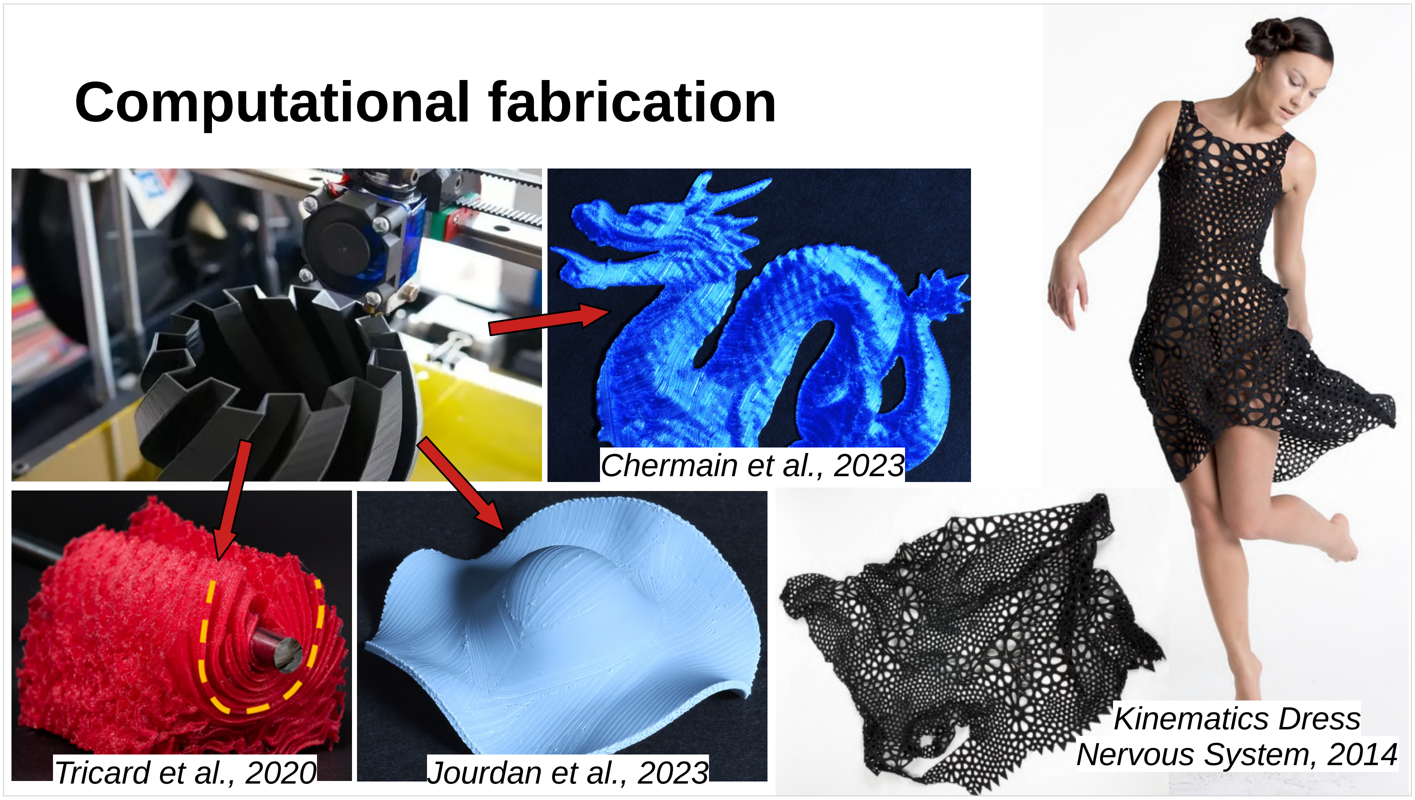
Computational design is the use of computer-based tools to help the design process. This ranges from assisting existing workflows, to completely changing them through novel approaches.

CAD software is now widespread among industrial users and amateurs alike, ranging from object modeling in engineering with tools such as FreeCAD (left), or electronics design with KiCAD (right).

Computational fabrication

The development of CAD software and the democratization of previously exclusively industrial tools, have enabled a wider public to be creative with them.

One example is 3D printing. Fused Filament Fabrication printers melt plastic that is deposited to create shapes layer by layer. This allows anyone to print any object represented by a 3D mesh.



Computational fabrication

Chermain et al., 2023

Tricard et al., 2020

Jourdan et al., 2023

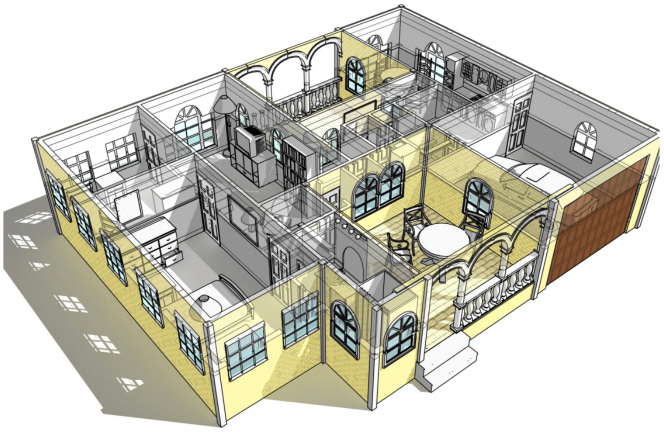
Kinematics Dress
Nervous System, 2014

Colleagues here at the MFX team find innovative ways to use them. Printers can be used to create objects with anisotropic appearances, foams that deform in a specified manner when force is applied, or even objects that deform when heated.

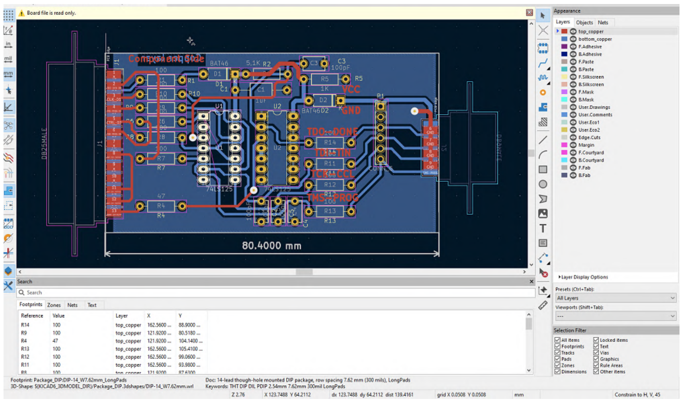
[CLICKx2] The *nervous system* studio specializes in leveraging 3D printing technologies to create procedurally generated art. Here on the right is the *Kinematics Dress*, a dress composed of unique interlocking components, 3D printed as a single piece. This piece is included in the permanent collections of museums such as the MoMA in NY, 3D printing has become a big deal.

Layout problems in design

Objects, space, constraints



Computer-generated residential building layouts,
Merrell et al., 2010



Reference	Value	Layer	X	Y
R14	100	top_copper	162.5600 ...	88.9000 ...
R9	100	top_copper	121.9200 ...	80.5180 ...
R4	47	top_copper	121.9200 ...	104.1400 ...
R13	100	top_copper	162.5600 ...	105.4100 ...
R12	100	top_copper	162.5600 ...	99.0600 ...
R11	100	top_copper	162.5600 ...	93.9800 ...
R8	100	top_copper	121.9200	87.6300

Footprint: Package_DIP:DIP-14_W7.62mm_LongPads
3D-Shape: ${KICAD6_3DMODEL_DIR}/Package_DIP.3dshapes/DIP-14_W7.62mm.wrl
Doc: 14-lead though-hole mounted DIP package, row spacing 7.62 mm (300 mils), LongPads
Keywords: THT DIP DIL PDIP 2.54mm 7.62mm 300mil LongPads

Z 2.76 X 123.7488 Y 64.2112 dx 123.7488 dy 64.2112 dist 139.4161 grid X 0.0508 Y 0.0508 mm Constrain to H, V, 45

My work specifically focuses on layout problems within computational design and fabrication. This is going to seem totally unrelated to what came before, but hopefully this all comes together by the end of my presentation.

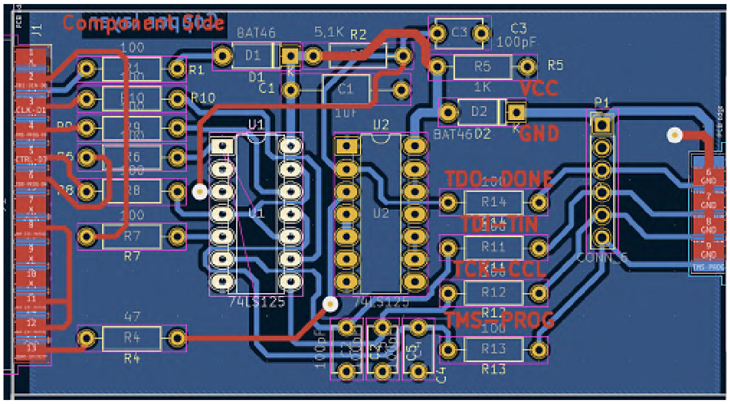
The concept of layout problem applies to many different situations. It consists of a set of **objects**, a **space**, and a set of **constraints**. Solving the layout problem means arranging the objects within the space while satisfying the constraints.

An example is [CLICK] building layouts, where a set of rooms, with a specific role have to be fit into a floorplan, while ensuring that e.g. all rooms are accessible, or optimizing window placement for daylight.

Another example is [CLICK] circuit layouts. Here a set of electronic components have to be fit into a circuit board and connected together as specified by the schematics.

Topological constraints

?



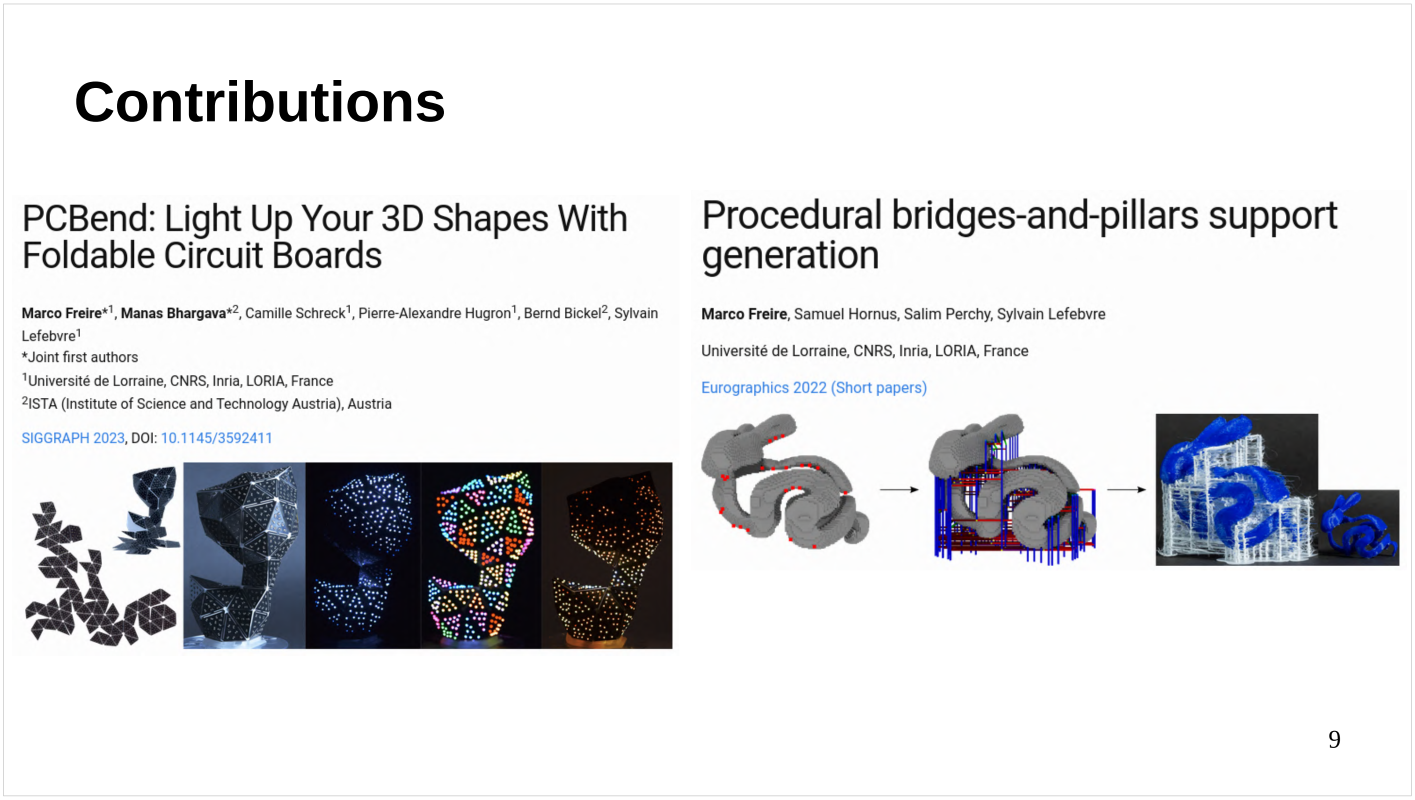
Objects are connected to each other!

I specifically deal with layout problems under topological constraints.

(As we all know, topologists tell us that a donut and a mug are the same, but react poorly when prompted to eat the mug.)

I use topological here in opposition to geometric. Geometry cares where objects are, whereas topology only cares about how they are connected.

What are topological constraints then? [CLICK] It means that the objects in the layout problem are connected to each other in some way or another. This is straightforward to see in the case of electronics circuits.



Contributions

PCBend: Light Up Your 3D Shapes With Foldable Circuit Boards

Marco Freire*[1], **Manas Bhargava***[2], Camille Schreck[1], Pierre-Alexandre Hugron[1], Bernd Bickel[2], Sylvain Lefebvre[1]
*Joint first authors
[1]Université de Lorraine, CNRS, Inria, LORIA, France
[2]ISTA (Institute of Science and Technology Austria), Austria

SIGGRAPH 2023, DOI: 10.1145/3592411

Procedural bridges-and-pillars support generation

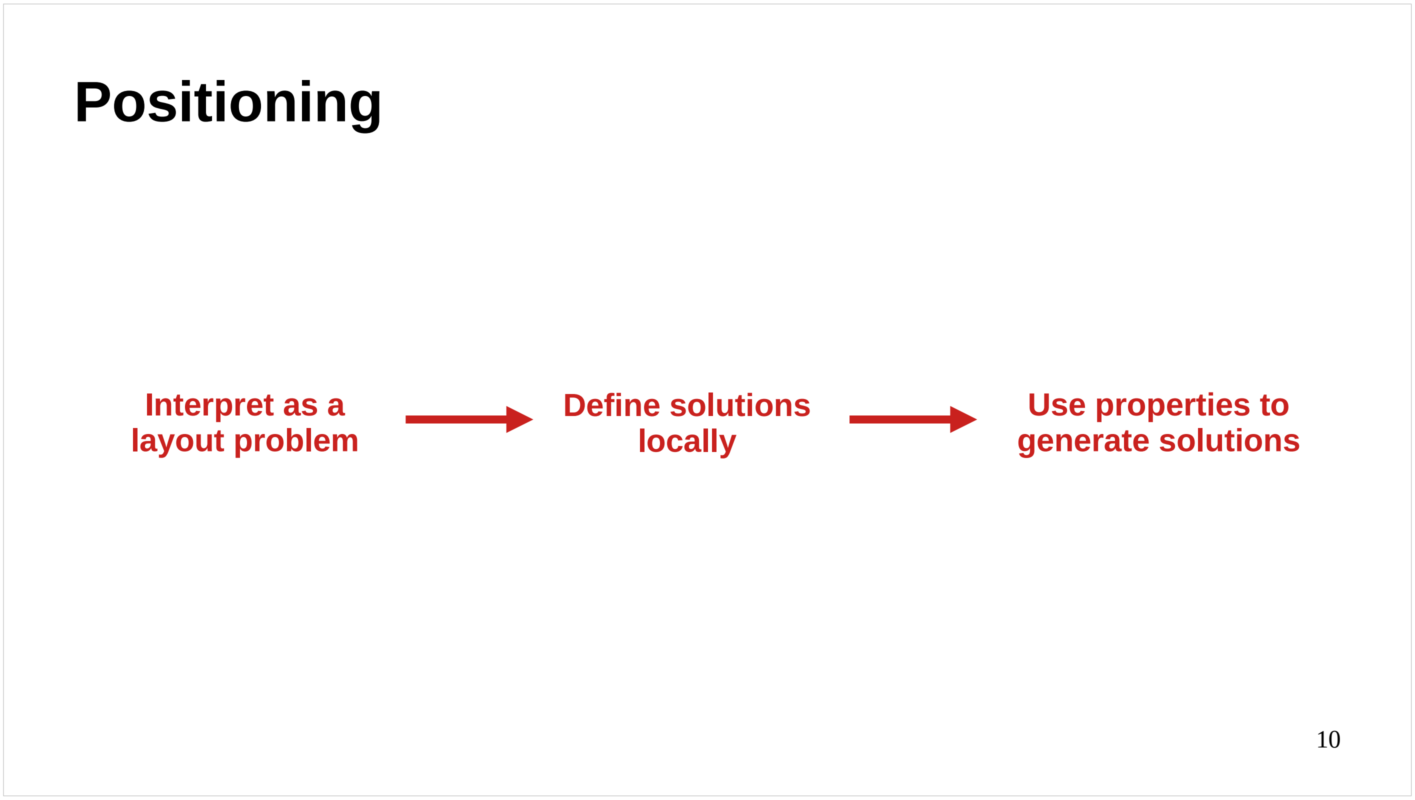
Marco Freire, Samuel Hornus, Salim Perchy, Sylvain Lefebvre

Université de Lorraine, CNRS, Inria, LORIA, France

Eurographics 2022 (Short papers)

During my thesis, I have tackled two specific problems, each having led to a publication.

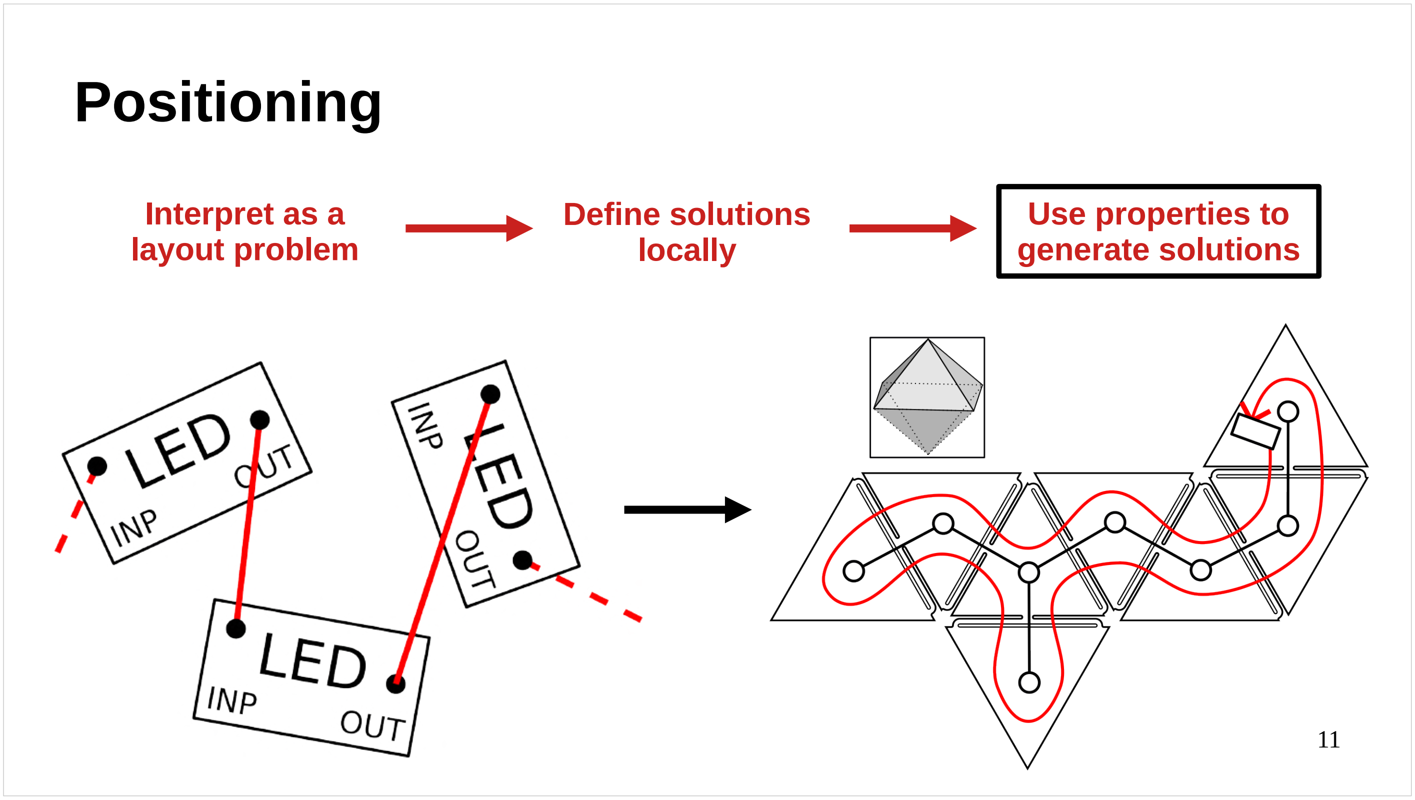
The first is circuit layout generation in the context of creating foldable circuit boards for 3D LED surface displays. The second is the procedural generation of support structures for 3D printing based on local descriptions of the supports.



Positioning

Interpret as a layout problem → Define solutions locally → Use properties to generate solutions

In both cases, we frame the challenge as a layout problem, which we solve [CLICK] by constructing solutions defined locally, and we exploit properties [CLICK] this definition provides us to design synthesis algorithms.

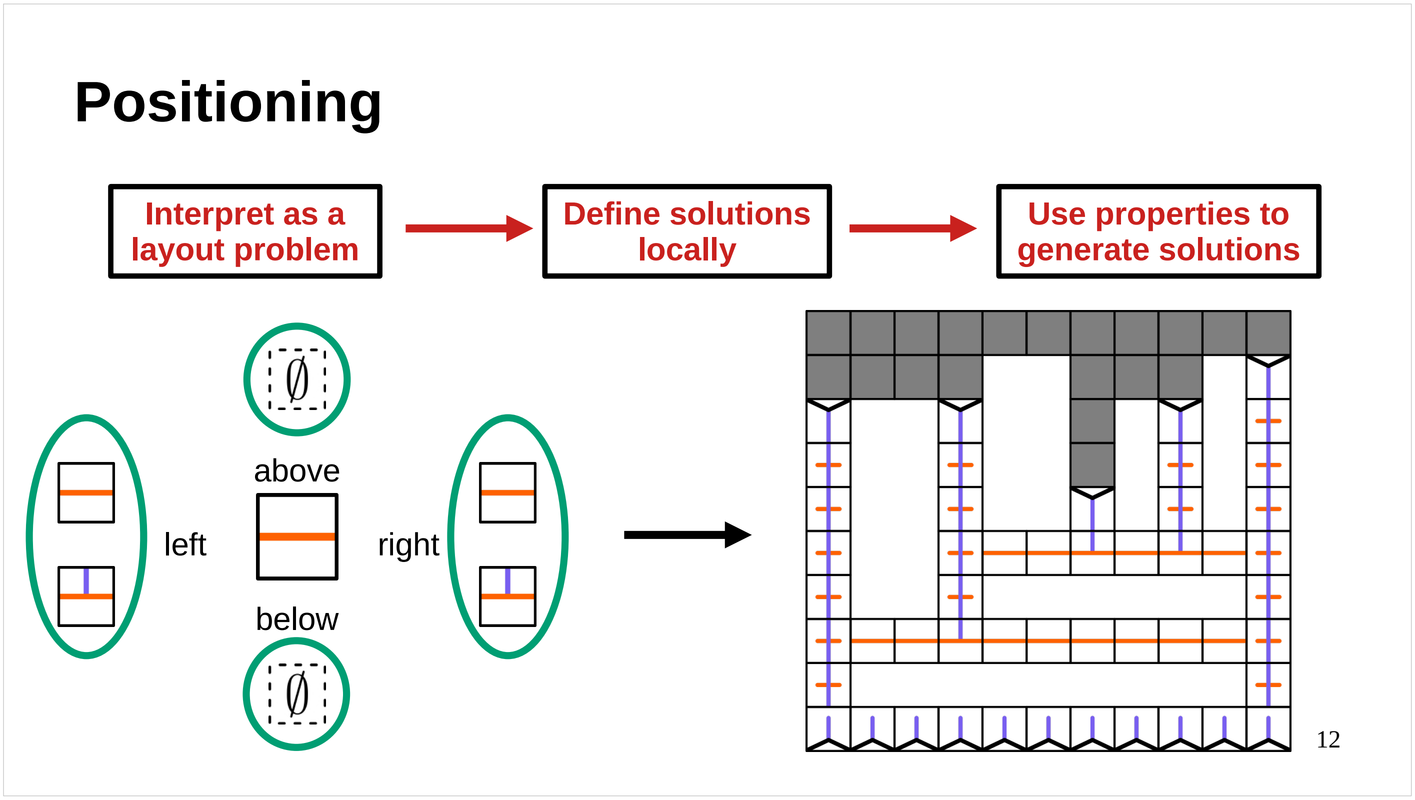


Positioning

Interpret as a layout problem → Define solutions locally → Use properties to generate solutions

It is straightforward to see that circuit layout generation is, a layout problem. The specific topology of our circuit (an LED chain), allows us to embed it [CLICK] in a challenging space (the PCB), resulting from the fabrication constraints.

The main contribution as a part of this thesis is the design of objects with fabrication concerns in mind, and the use of their emergent properties to efficiently generate circuit layouts.



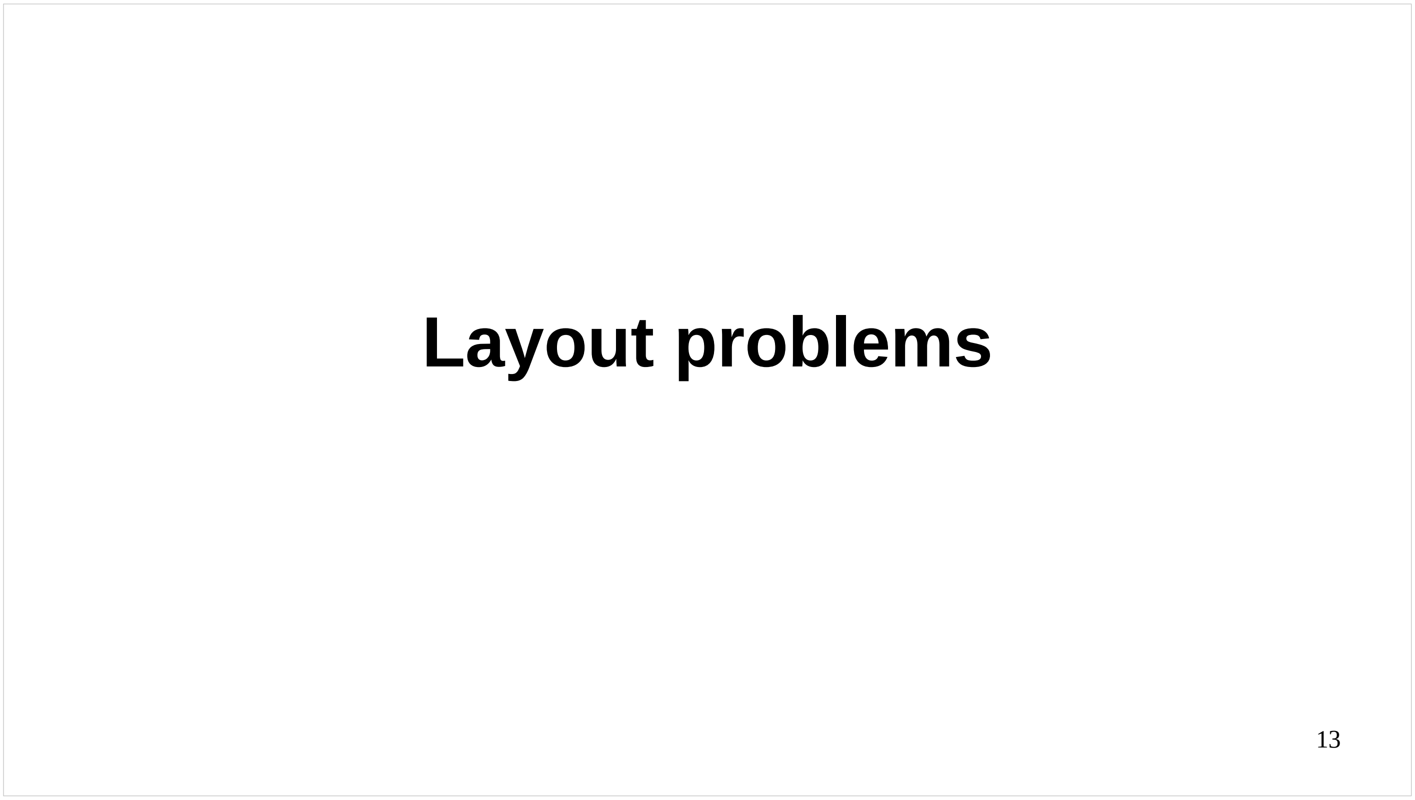
Positioning

Interpret as a layout problem → Define solutions locally → Use properties to generate solutions

It is less obvious to see how generating support structures for 3D printing is a layout problem.

We work by tiling 3D space with a set of building blocks that define the support structure. These blocks come with extra information specifying how they can be assembled with other blocks. This defines a space of all possible support structures, [CLICK] among which we synthesize a solution.

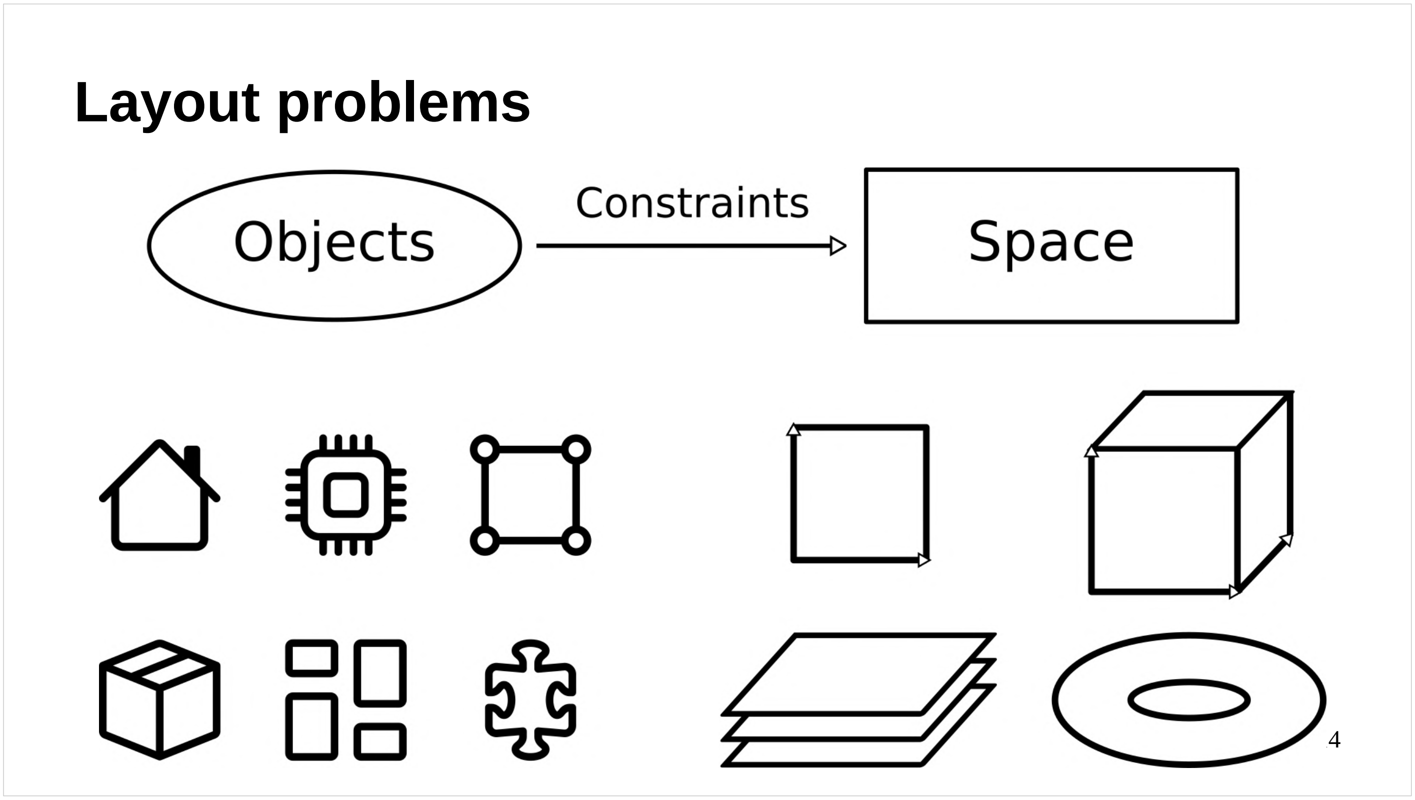
Here the contribution is, in my opinion, particularly original. Everything from seeing this as a layout problem, to properly defining supports locally, to generating a coherent support structure was a challenge.



Layout problems

[~6min]

Now let me provide a short introduction into layout problems.



Layout problems

Objects

Constraints

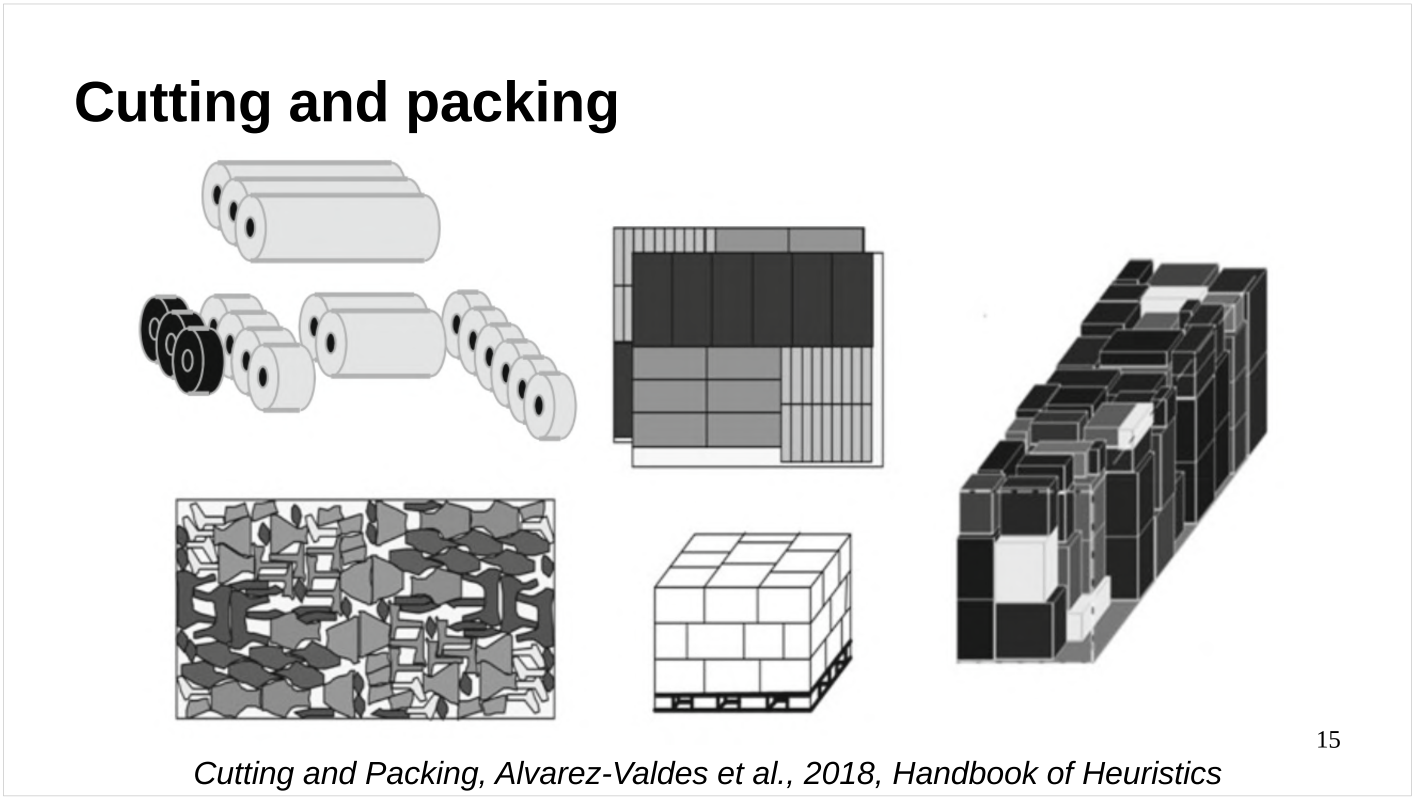
Space

4

Layout problems are a very general class of problems. They consist of three things: a set of objects, [CLICK] a space, [CLICK] and a set of constraints and objectives.

Solving a layout problem simply means finding an arrangement of the objects within the space that satisfy the constraints and optimize the objectives.

Unfortunately, these are often NP-hard combinatorial problems. This makes it often necessary to find clever heuristics to find approximate solutions.



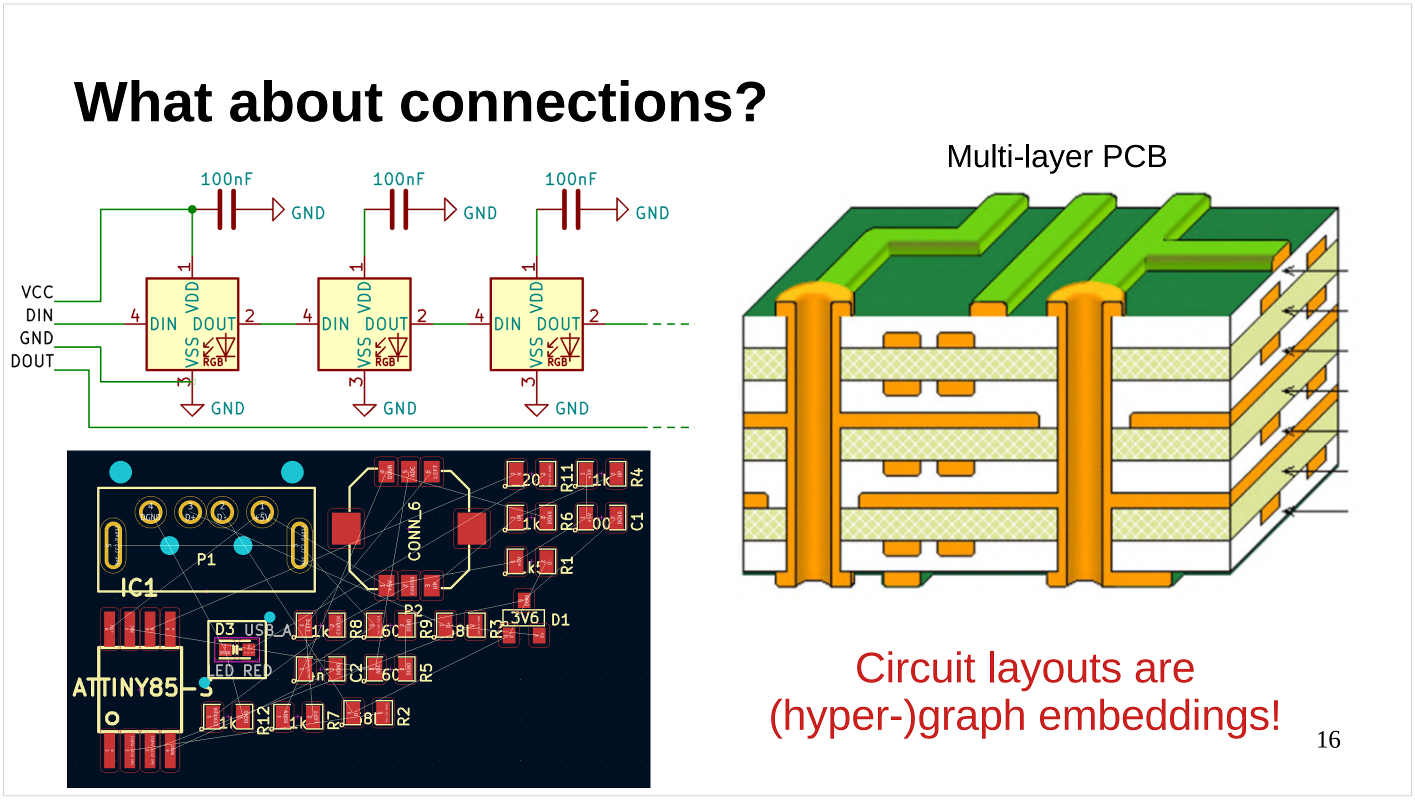
Cutting and packing

Cutting and Packing, Alvarez-Valdes et al., 2018, Handbook of Heuristics

Cutting and packing problems are simple to describe and are relevant to all sorts of fabrication and logistics.

They arise when an object’s or container’s space has to be divided into smaller items while minimizing waste. The defining constraint here is **geometric**: items must not overlap.

Due to the importance of this type of problems, there exists a large body of literature on them.



What about connections?

Multi-layer PCB

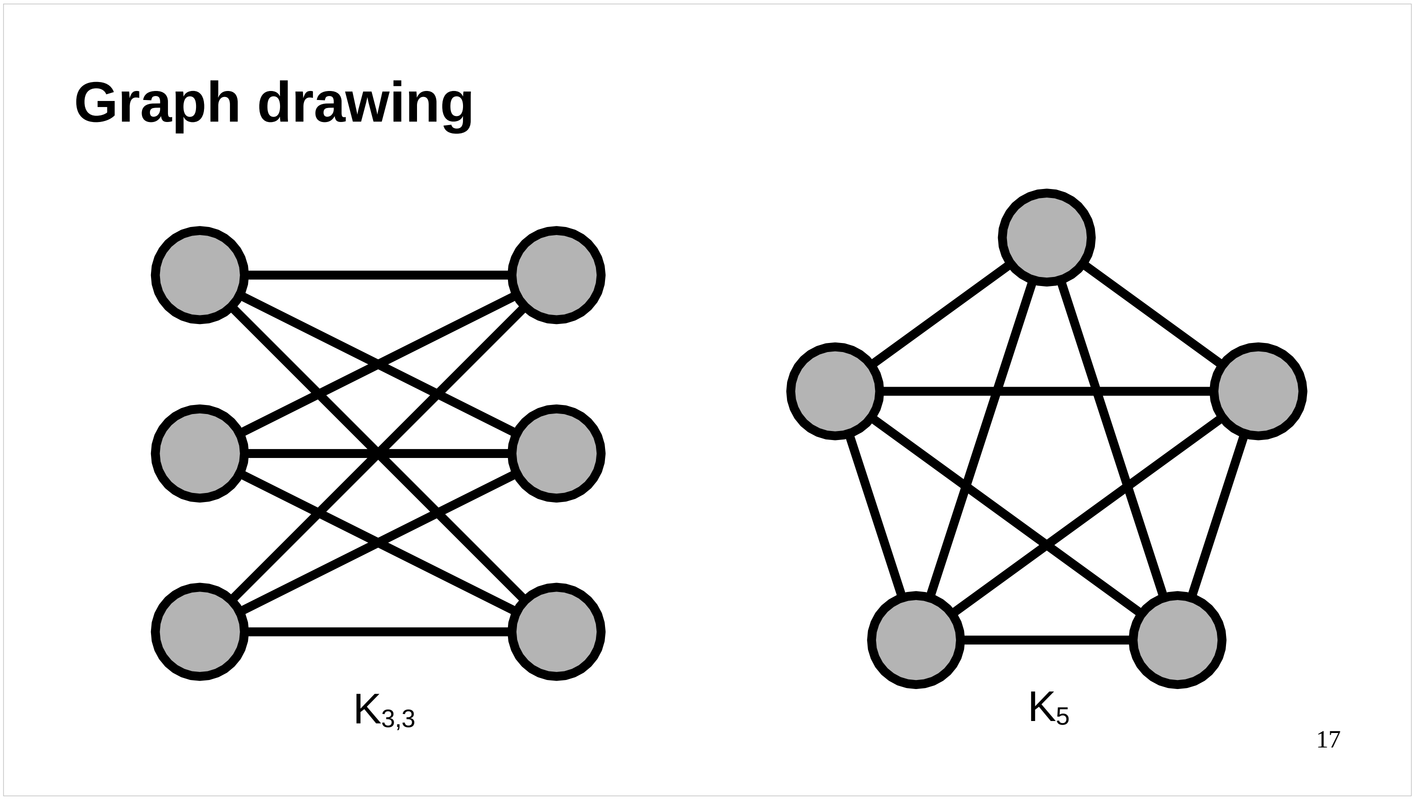
Circuit layouts are (hyper-)graph embeddings!

In cutting and packing, there are no connections between elements, which are key in the formulation of the layout problems we target.

Indeed, circuits are graphs, they consist of components connected together by conductive material. This connectedness is easily seen in the schematics and rat’s nest on the left.

[CLICK] Also, PCBs are complicated spaces, with multiple conductive layers connected through vias. If I want to be fancy, circuit layouts [CLICK] are then (hyper-)graph embeddings with electrical constraints.

We cannot ignore these connections: they are integral to the circuit function, and poor layouts may even make performance worse or increase the cost of the circuit.



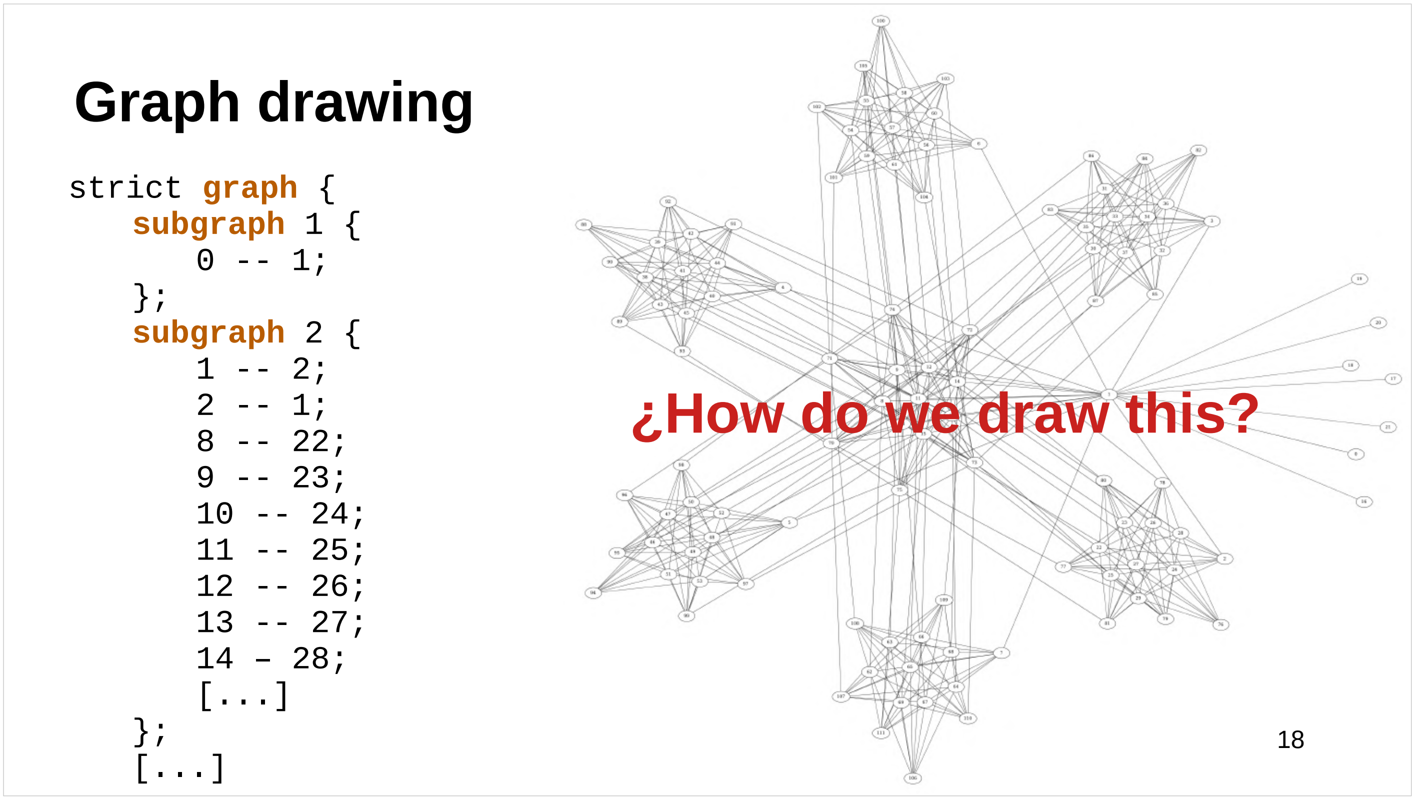
Graph drawing

To better understand the complexity of these layout problems, we’ll look at **graph drawing**. Graph drawing is the quintessential layout problem, hiding under a deceptively simple appearance. It appears within more complex layout problems such as circuit design.

As a quick reminder, a graph consists of nodes or vertices and edges connecting them. Graph drawing is the process of drawing all nodes and edges usually on a plane.

Nodes are connected to each other, endowing the graph with a **topological** structure. This structure can be represented in drawings, and usually provides information on the real situation modeled by the graph.

For example, on the left we have a bipartite graph, K33, where every edge connects a node on the left with a node on the right. On the right we have a complete graph, K5, where all pairs of nodes are connected with an edge. These properties are clearly visible in these drawings.



Graph drawing

```
strict graph {
   subgraph 1 {
      0 -- 1;
   };
   subgraph 2 {
      1 -- 2;
      2 -- 1;
      8 -- 22;
      9 -- 23;
      10 -- 24;
      11 -- 25;
      12 -- 26;
      13 -- 27;
      14 – 28;
      [...]
   };
   [...]
```

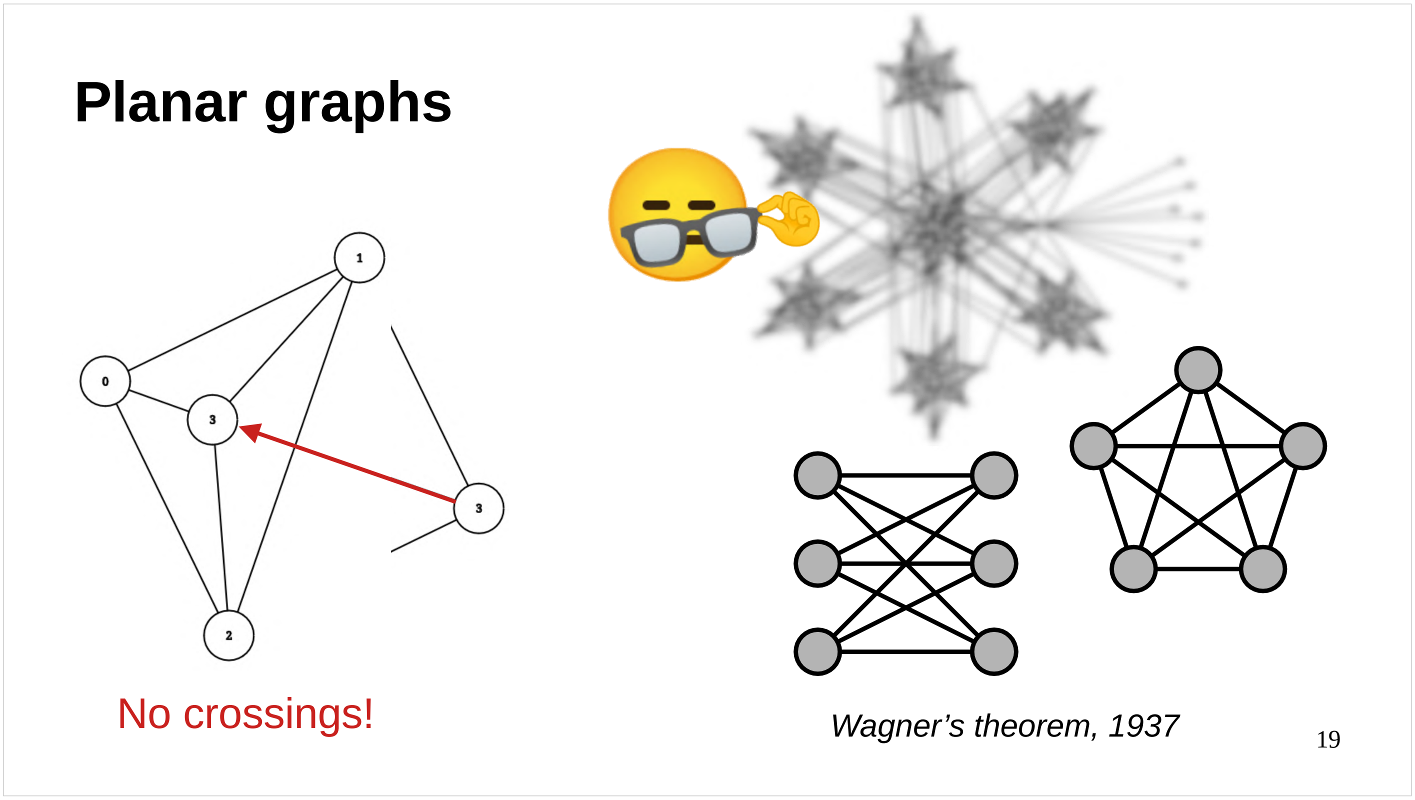
¿How do we draw this?

But these are very simple graphs, what about larger ones?

Here on the left is a written representation of a graph, specifying what pairs of nodes are connected. We don’t know what the graph looks like from this description!

The question is then, [CLICK] how do we draw this? Here is a possible solution [CLICK] generated with graphviz, a graph drawing software.

As you can see, there are similar clusters of tightly connected nodes drawn closer together. This representation definitely helps building an intuition about the structure of the graph.



Planar graphs

No crossings!

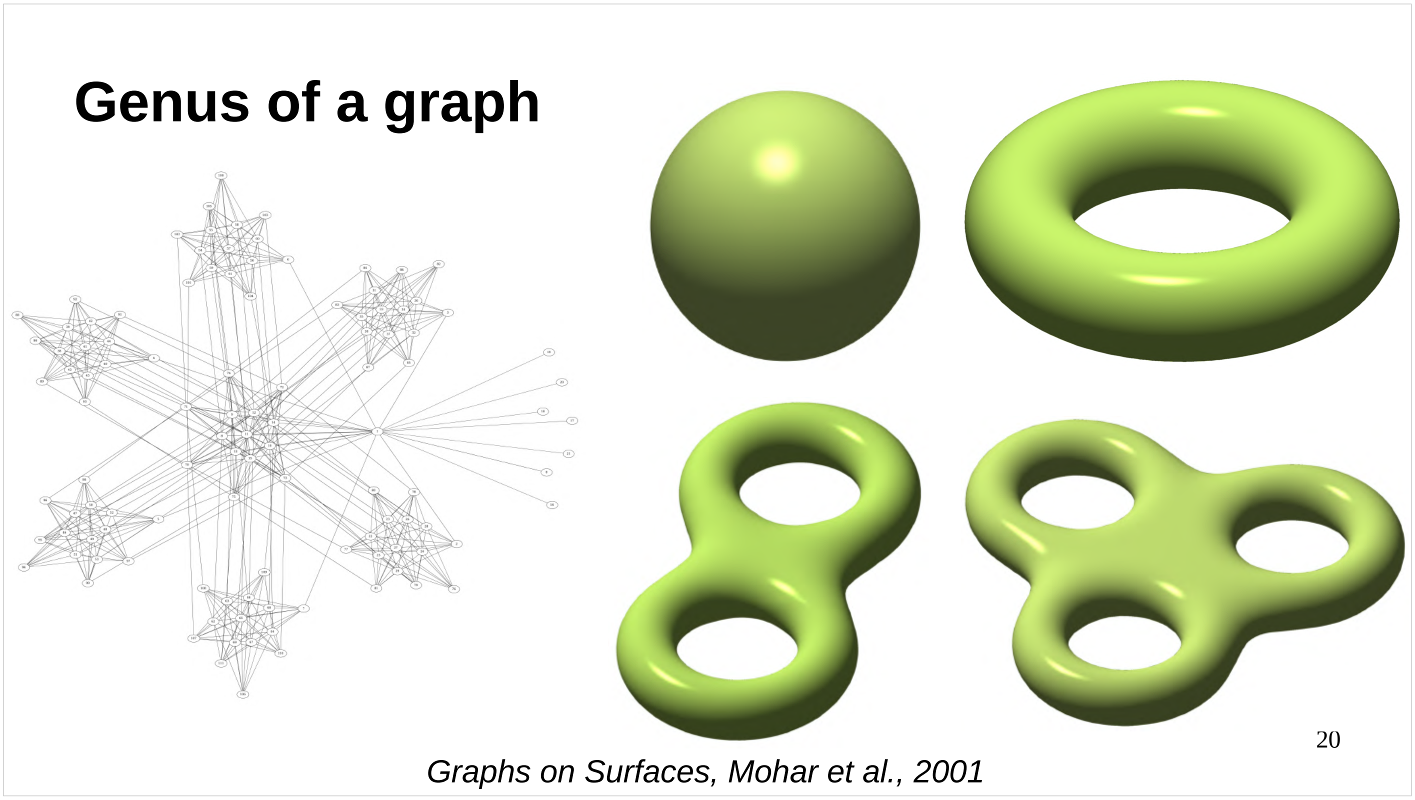
Wagner's theorem, 1937

Connectedness introduces a host of new problems and properties we can study, e.g. planarity.

A graph is planar if it can be drawn without any of its edges crossing. This drawing is not planar, but by [CLICK] moving node 3 inside of this triangle, we obtain [CLICK] a planar drawing of the graph. This shows that the graph itself is planar. Planarity is very relevant to circuit design, since copper traces must not intersect.

How do we do this for more complicated [CLICK] graphs? Well, if we squint a bit [CLICK] and see that the complicated graph contains [CLICK] K33 or K5, we know that the graph is not planar! (with a graph containing another having a precise mathematical definition).

Wagner's theorem states that K33 and K5 are the "fundamentally non-planar graphs", providing a surprisingly simple characterization for a complex notion.



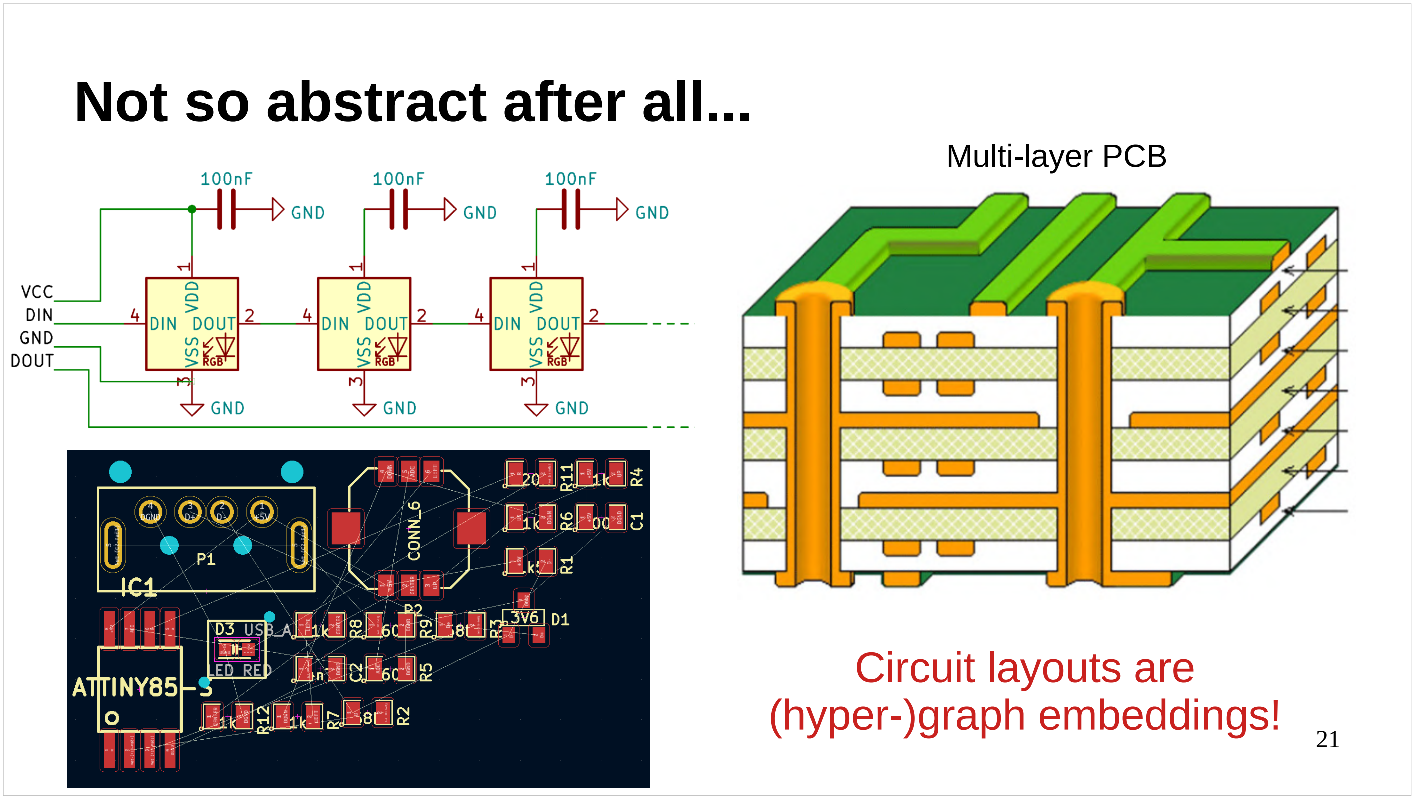
Genus of a graph

Graphs on Surfaces, Mohar et al., 2001

We can actually complicate things even further. Trust me when I say that drawing a graph on a sphere or a plane is the same. But of course, a sphere is simply a donut without a hole.

It can be proven that [CLICK] every finite graph can be drawn without crossings onto a surface with enough holes. The number of holes is known as the genus of the graph.

For us, non-graph theorists, this might seem like a pretty far-fetched concept.



Not so abstract after all...

Multi-layer PCB

Circuit layouts are (hyper-)graph embeddings!

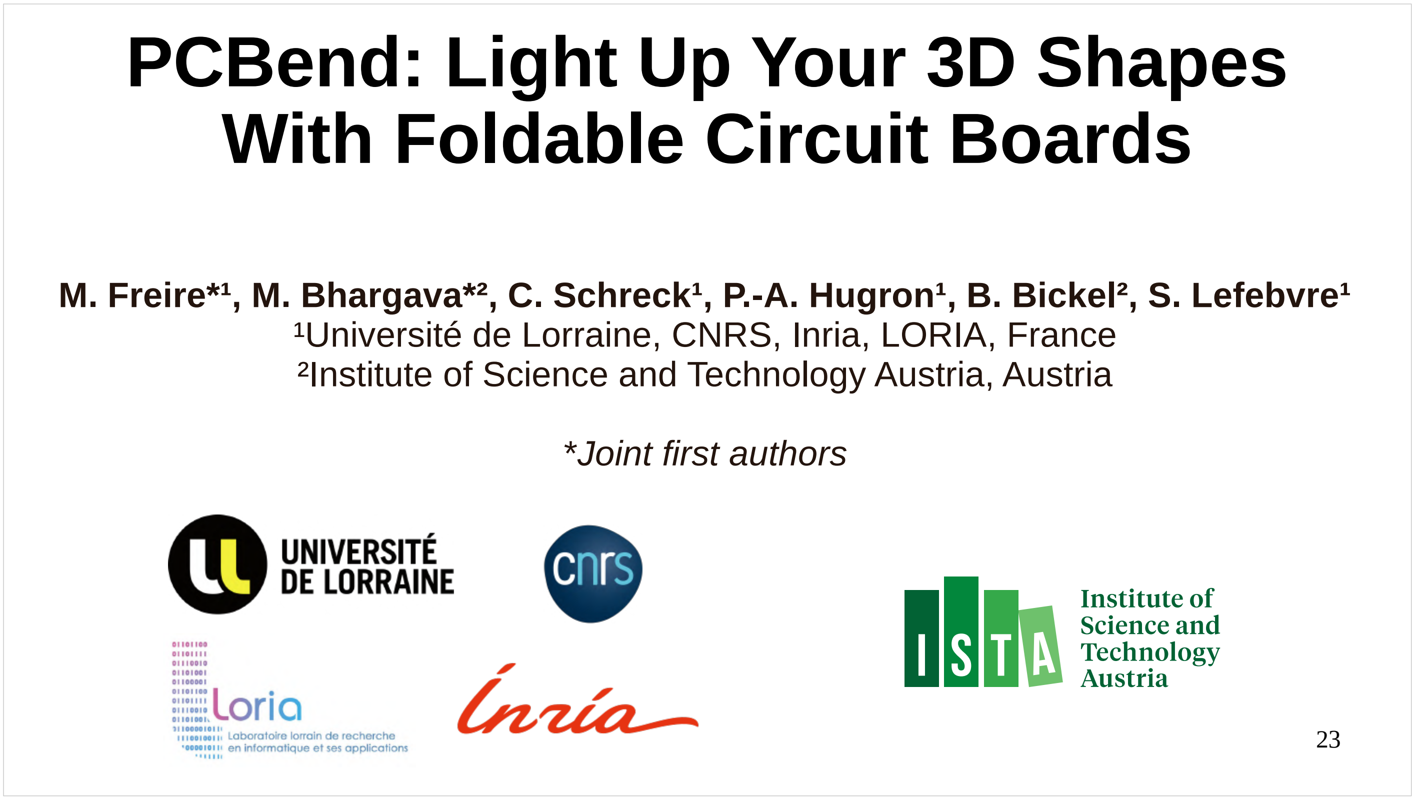
But remember! Circuit layouts are hypergraph embeddings into complex spaces consisting of multiple sheets connected by holes.

The seemingly unrelated complexity of graph drawing and topological graph theory, arising only from edges between nodes, comes up in many other layout problems, making them fundamentally hard.

Contributions

[~12min]

With a now clearer view on layout problems, and their surprising complexity, let us focus on the contributions of my thesis.



PCBend: Light Up Your 3D Shapes With Foldable Circuit Boards

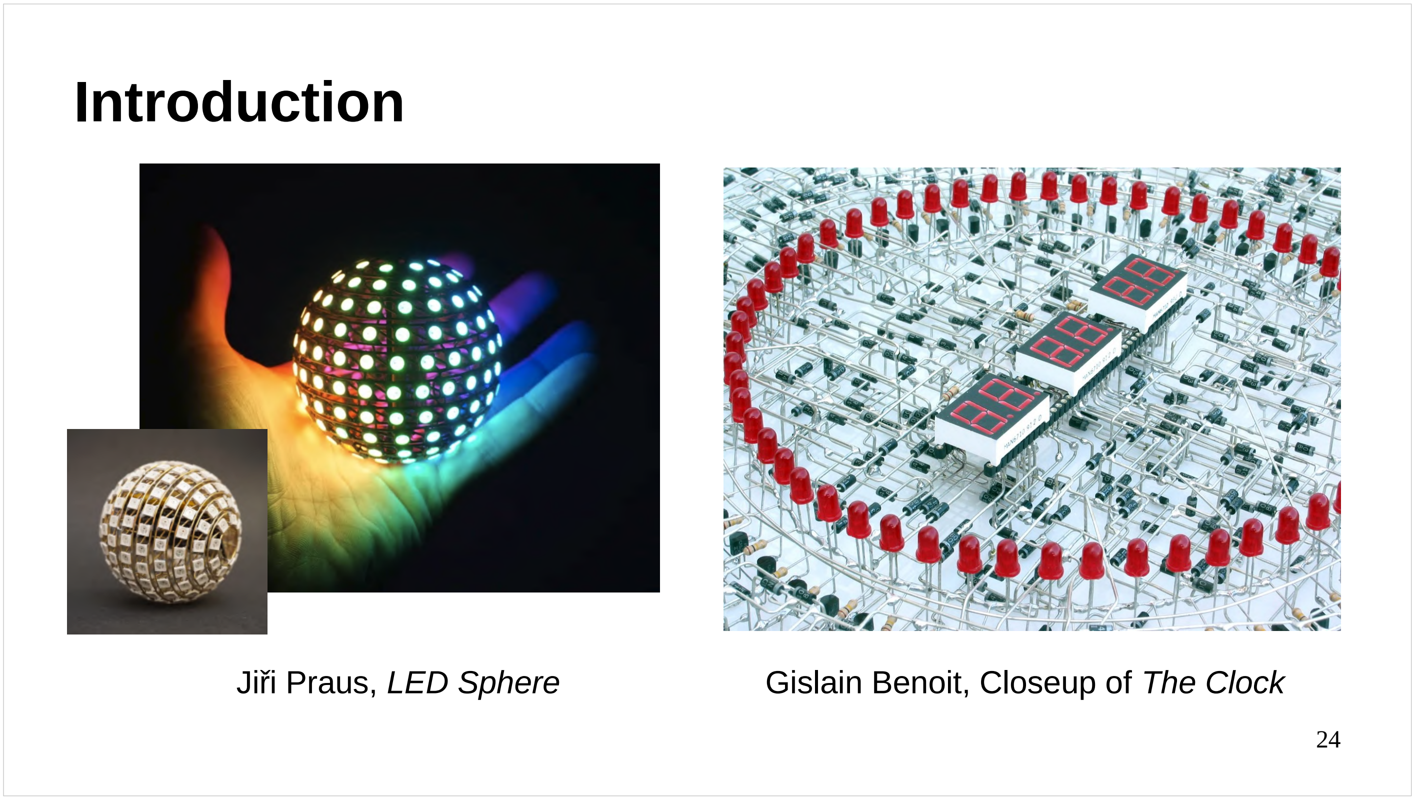
M. Freire*[1], M. Bhargava*[2], C. Schreck[1], P.-A. Hugron[1], B. Bickel[2], S. Lefebvre[1]
[1]Université de Lorraine, CNRS, Inria, LORIA, France
[2]Institute of Science and Technology Austria, Austria

**Joint first authors*

First I will talk about the PCBend project that resulted in a SIGGRAPH 2023 publication.

This project was done in collaboration w/ Manas Bhargava and Bernd Bickel from ISTA.

In here we tackle a set of layout problems related to electronics design.



Introduction

Jiři Praus, *LED Sphere*

Gislain Benoit, Closeup of *The Clock*

Do-it-yourself LED-based projects are hugely popular on the internet, among electronic hobbyists and makers. They often showcase impressive creations and lighting effects using LEDs and freeform circuitry.

Here on the left is a freeform LED sphere, where all LEDs are soldered onto copper wire shaped as a ball. On the right is a closeup of a much larger piece, *The Clock* which is a fully functional clock created only with components and conductive wire.

Introduction

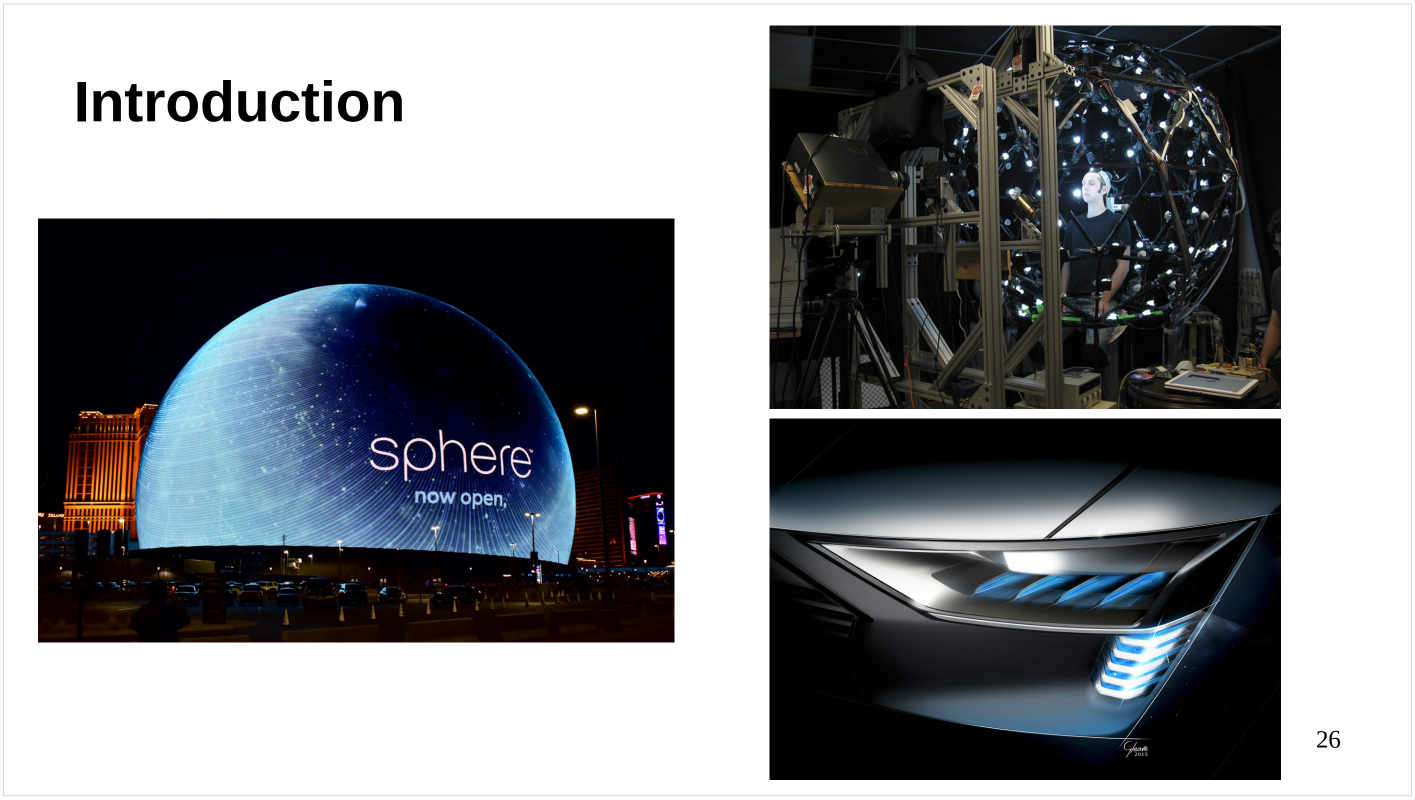
“I’m not sure how long it took me to complete all the rework, but I did make it through a 60h audio book that week.”

Greg Davill

These DIY projects take enormous amounts of skill and time to execute. Just a simple rework due to a design mistake on the icosahedral display shown here required many tens of hours to fix.

All of these LED creations require *ad hoc* solutions needing to be designed from the ground up, making this approach difficult to scale and generalize.

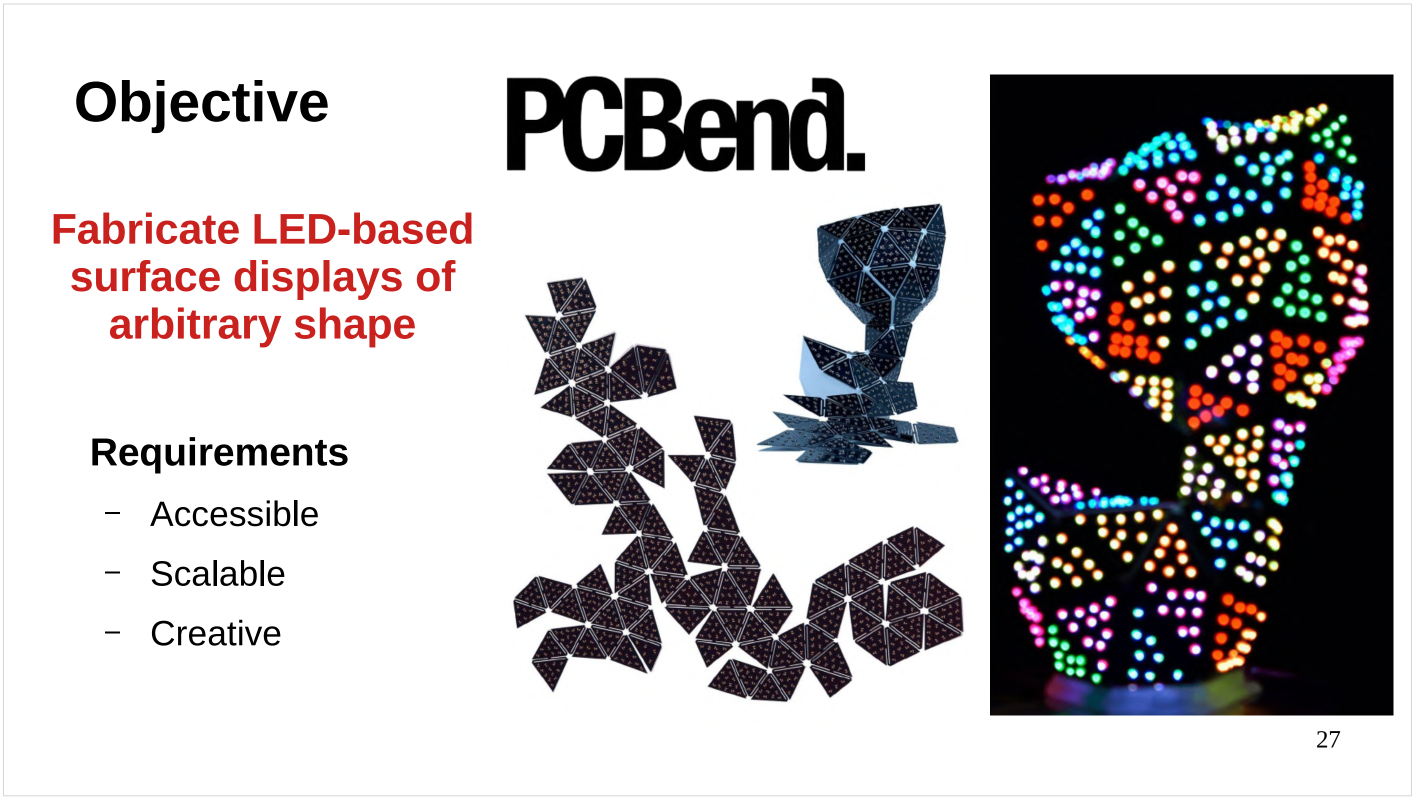
Seeing these examples, it’s no surprise that these creations are so popular, given the diversity of visual effects that can be created with them.



Introduction

More generally, LED-based lighting installations have become highly popular in the last decades, finding a trove of different uses for commercial and industrial applications.

This ranges from the Las Vegas LED sphere, able to display animations in every direction, to LED-based light stages used for facial relighting in the movie industry, passing by the development of light signatures in the automotive industry, where light patterns now serve to distinguish between car brands and models.



Objective

PCBend.

Fabricate LED-based surface displays of arbitrary shape

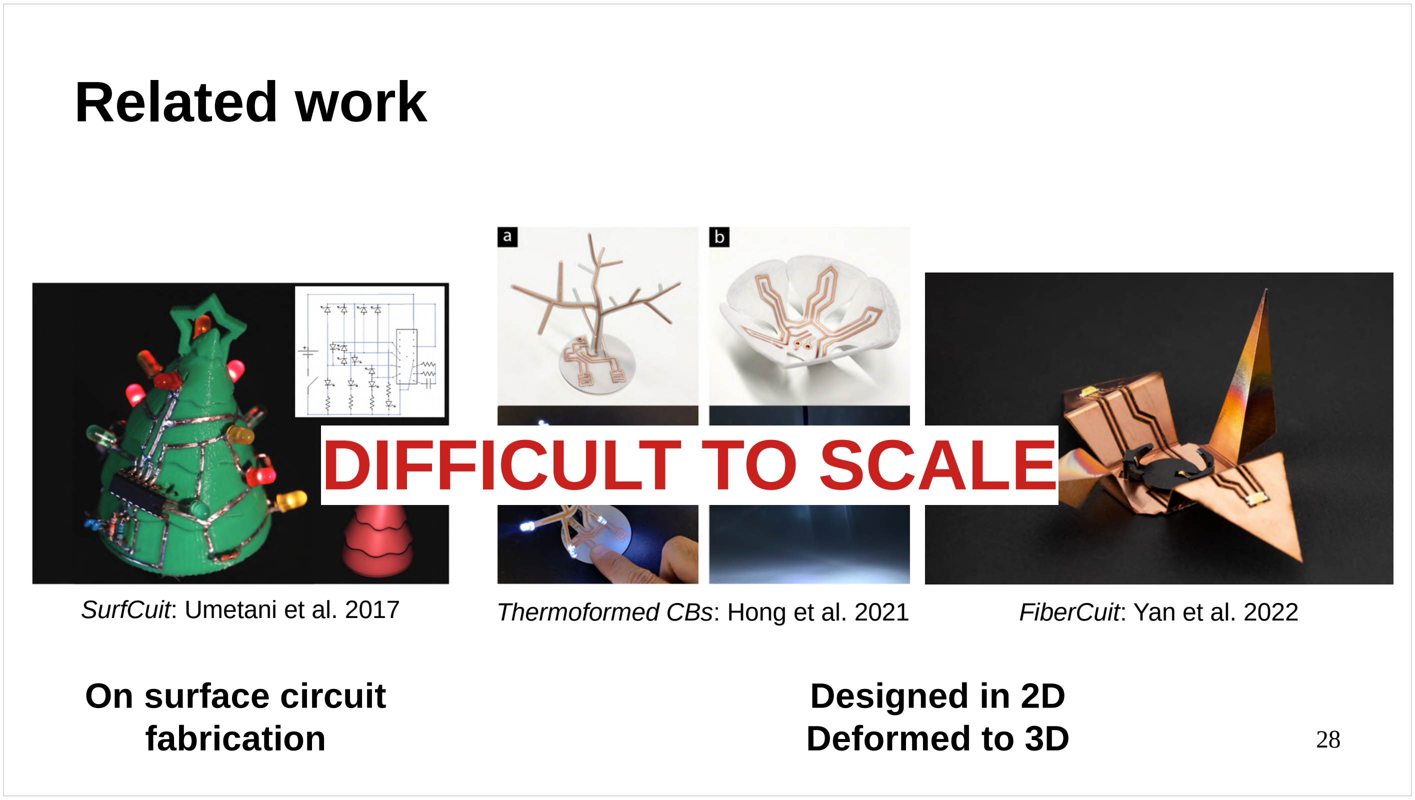
Requirements

- Accessible
- Scalable
- Creative

Our project tackles the challenge of creating custom surface LED displays from a user-provided input mesh.

Our goal with this system was to make it simultaneously: accessible, scalable, easy to use and still enable the users’ creativity.

We wanted to keep the spirit of the DIY LED projects, while making it accessible to a broader public with less technical knowledge. For this reason, we made the whole system open-source, accessible on github with instructions.



Related work

DIFFICULT TO SCALE

SurfCuit: Umetani et al. 2017

Thermoformed CBs: Hong et al. 2021

FiberCuit: Yan et al. 2022

On surface circuit fabrication

Designed in 2D Deformed to 3D

The development of 3D circuits and novel methods for circuit fabrication is not new. Previous work on these areas have followed two main approaches.

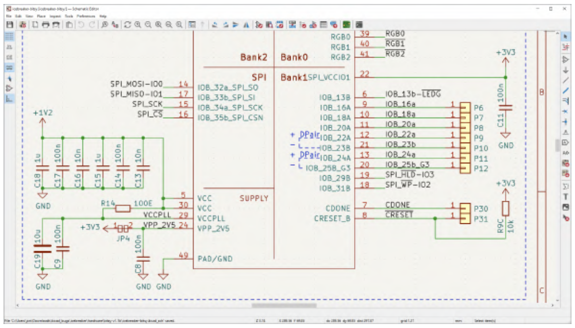
The first approach [CLICK] consisted in directly designing and fabricating the circuit on a 3D surface.

The second approach [CLICK] consists in designing the circuit in 2D and later deforming it to 3D by various methods such as heating or folding.

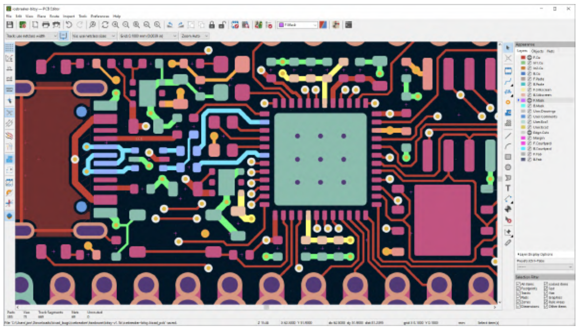
These methods use non-traditional conductors (such as inks or plastic filament), or atypical substrates (such as paper or 3D printed objects) to design non-planar circuits. While these technologies enable new types of circuitry and electronics, [CLICK] it is difficult to scale them to more complex circuitry with 100s or 1000s of components.

Before getting into the weeds, let me say a few words on traditional circuit design.

Traditional circuit design



Schematics design

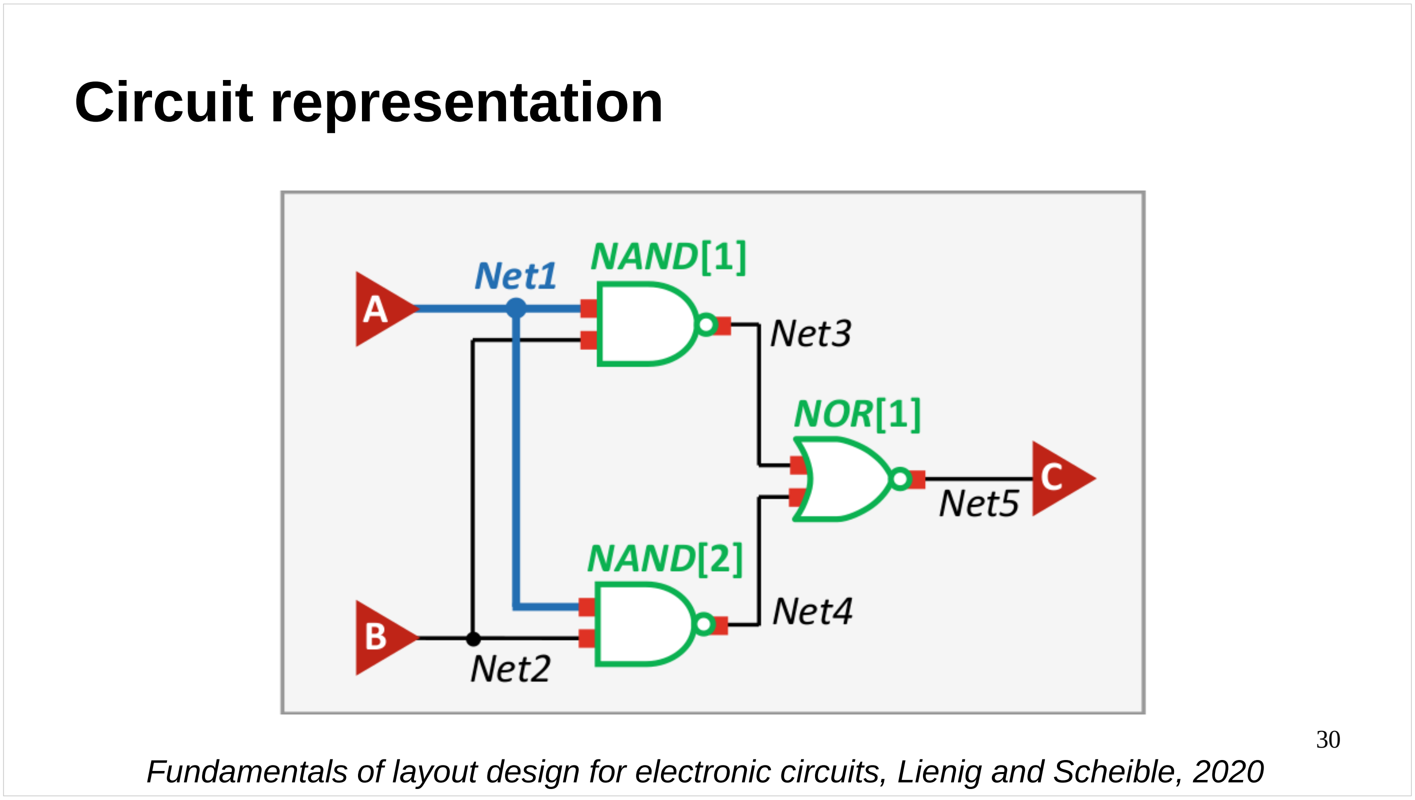


Physical layout design

Fundamentals of layout design for electronic circuits, Lienig and Scheible, 2020

Traditional circuit design starts with a schematics based on a functional specification. The schematics specify the components of the circuit and how they are connected together.

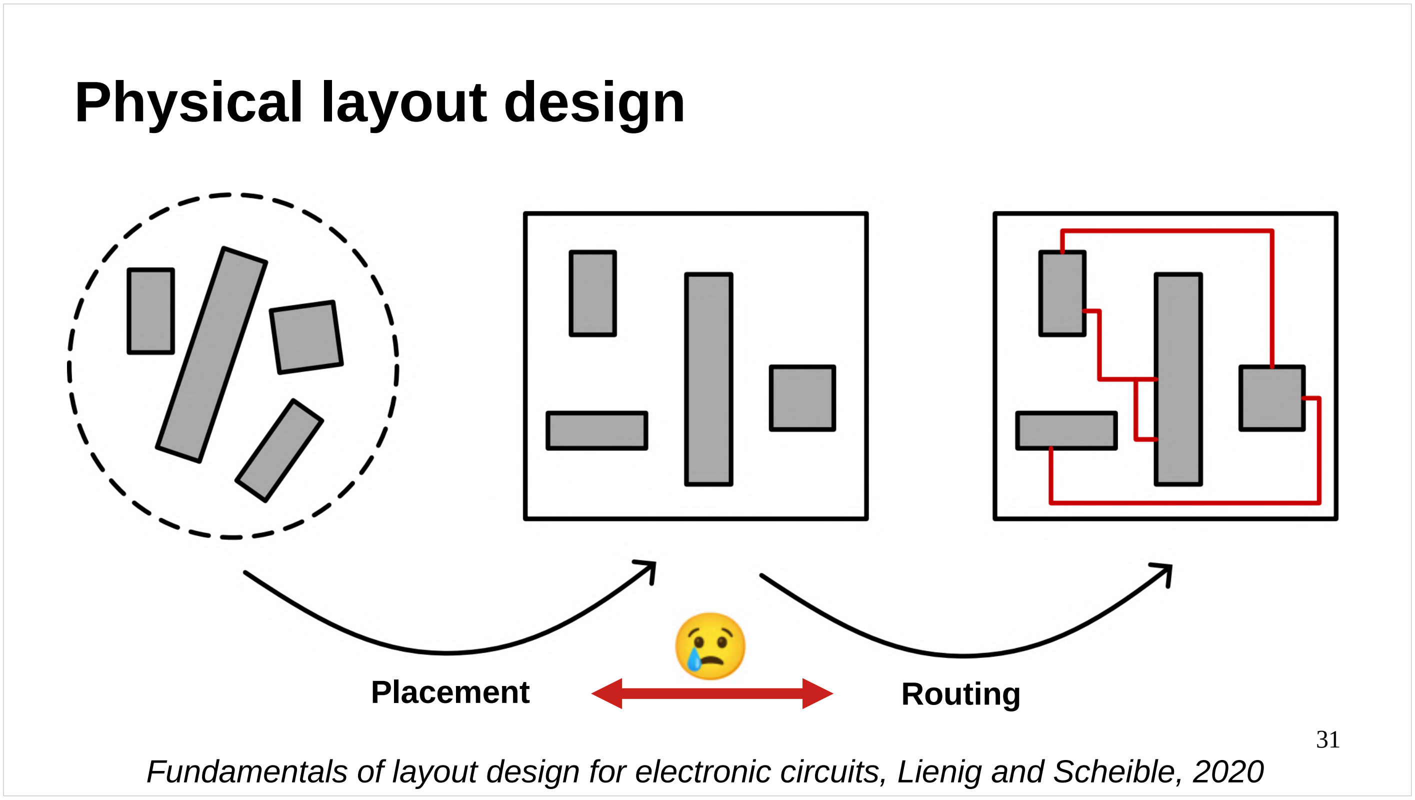
Then, physical layout design takes this information and physically embeds it in the circuit board, defining the actual geometry of the circuit, while avoiding electrical and thermal problems among many others.



Circuit representation

Fundamentals of layout design for electronic circuits, Lienig and Scheible, 2020

Computationally, this circuit structure is represented as a netlist, where a net is a wire connecting two or more pins. This representation contains all of the information necessary for the physical layout phase.



Physical layout design

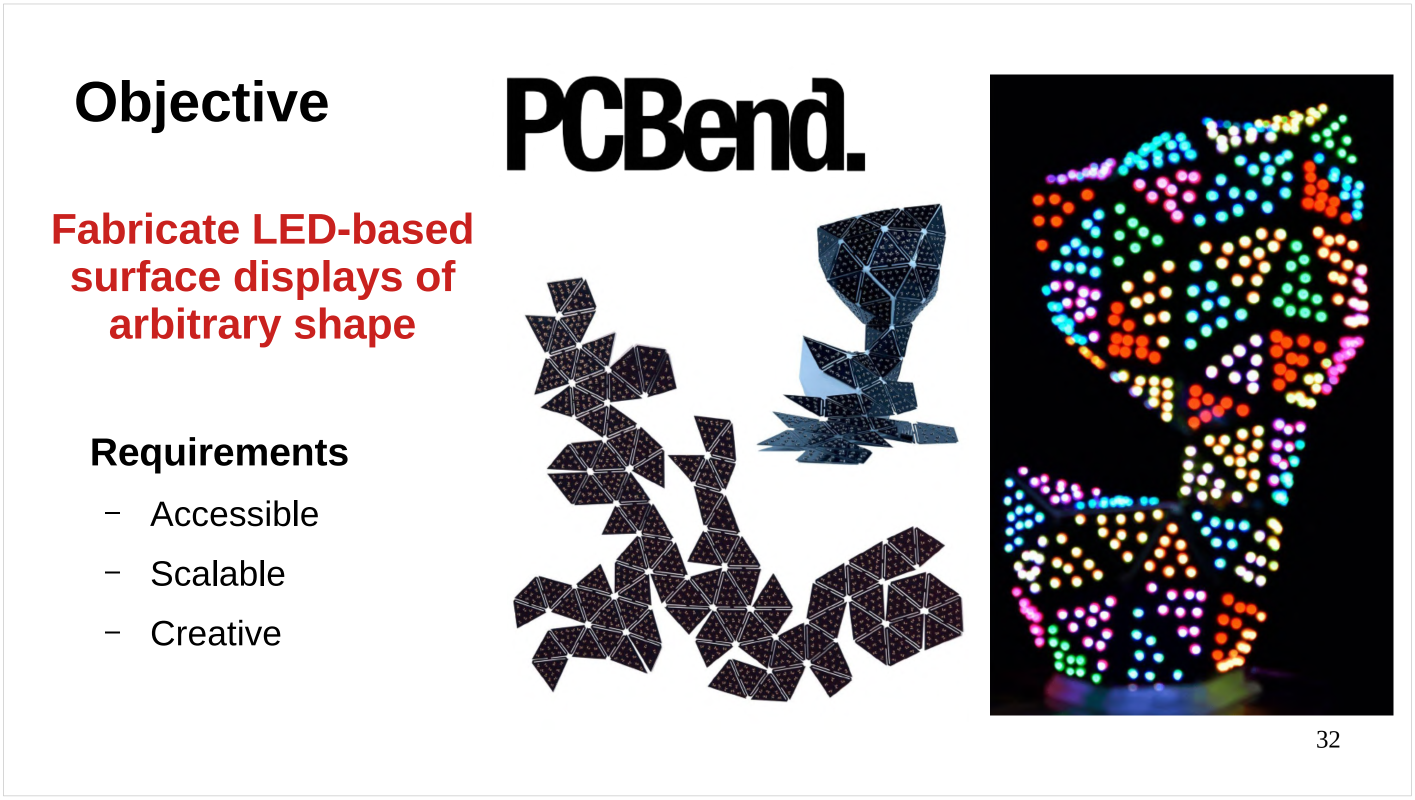
Placement 😢 Routing

Fundamentals of layout design for electronic circuits, Lienig and Scheible, 2020

Physical layout consists of two important steps, **placement** and **routing.** Placement takes the components specified in the netlist and places them within the board. Routing uses the connectivity information and embeds the interconnects in the space.

Modern designs can be incredibly large, so additional steps are used to break down the problem into smaller tractable instances.

A good routing usually minimizes total wirelength, which is relatively easy to estimate. However, a good placement is one that can be routed [CLICK], which as you can expect, is pretty hard to check without having to route the whole design. These constraints make these problems NP-hard in general.



Objective

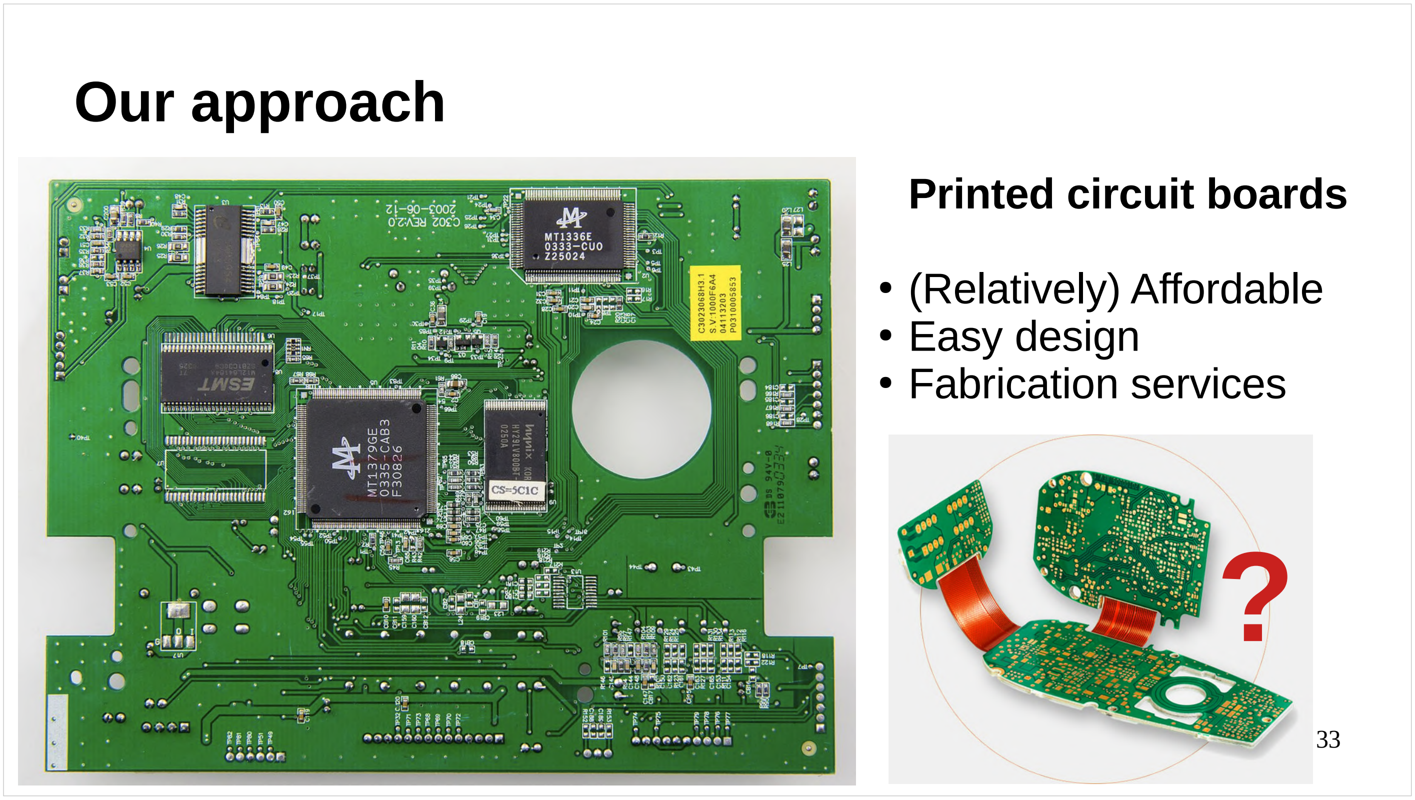
PCBend.

Fabricate LED-based surface displays of arbitrary shape

Requirements

- Accessible
- Scalable
- Creative

With this out of the way, we are back to our main topic! Remember that our goal is to create surface displays of arbitrary shape.



Our approach

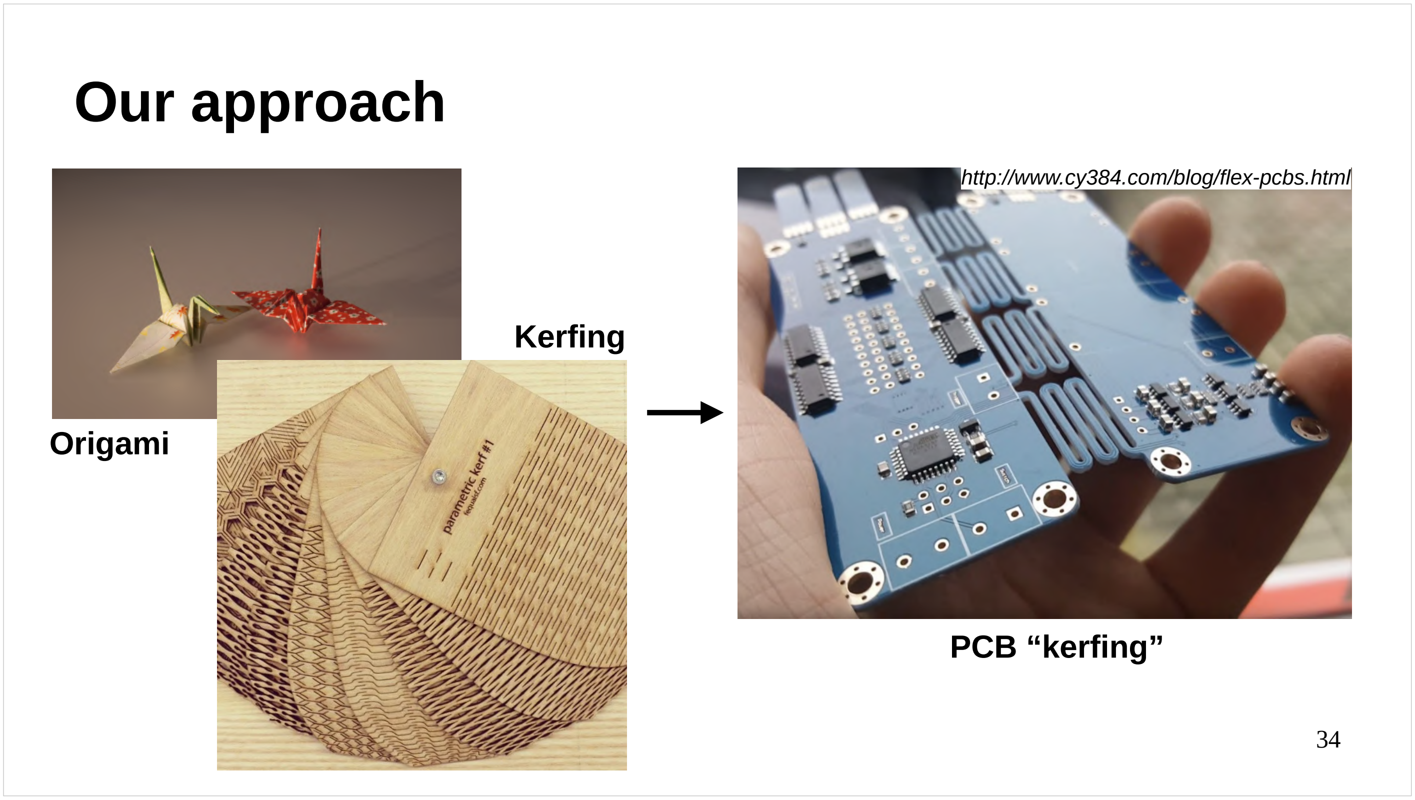
Printed circuit boards

- (Relatively) Affordable
- Easy design
- Fabrication services

We approach the fabrication of 3D electronics from a different point of view.

We decided early to target traditional PCBs, which are a very mature technology, that can be produced cheaply through online services, by inexperienced designers.

Rigid-flex boards also exist. [CLICK] These consist of multiple rigid PCBs connected by flexible ribbons. These are 5x to 10x more expensive, harder to design for, and take longer to manufacture, so we will limit ourselves to regular PCBs.



Our approach

Origami

Kerfing

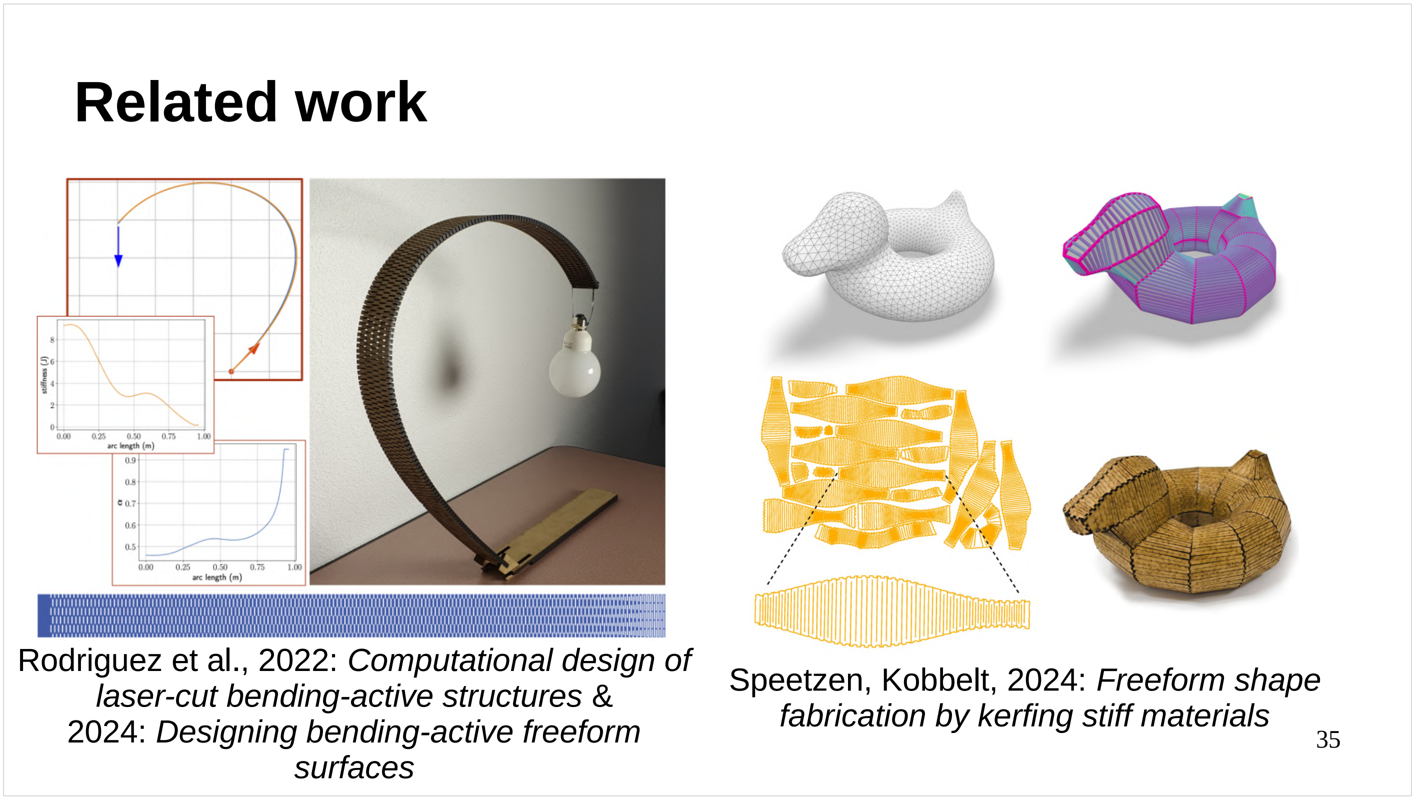
http://www.cy384.com/blog/flex-pcbs.html

PCB “kerfing”

To turn a PCB into a 3D object, we take inspiration from two techniques used on different materials. First, origami [CLICK], which is the art of bending paper to create complex shapes. Second, [CLICK], wood kerfing, where a saw or a laser cutter is used to engrave of cut patterns the wood.

Kerfing allows the wood to take on new mechanical properties, being able to bend where it previously couldn’t.

We combine these two approaches in PCB “kerfing”, [CLICK] where we use patterns cut into the PCB to allow it to bend where we wish.

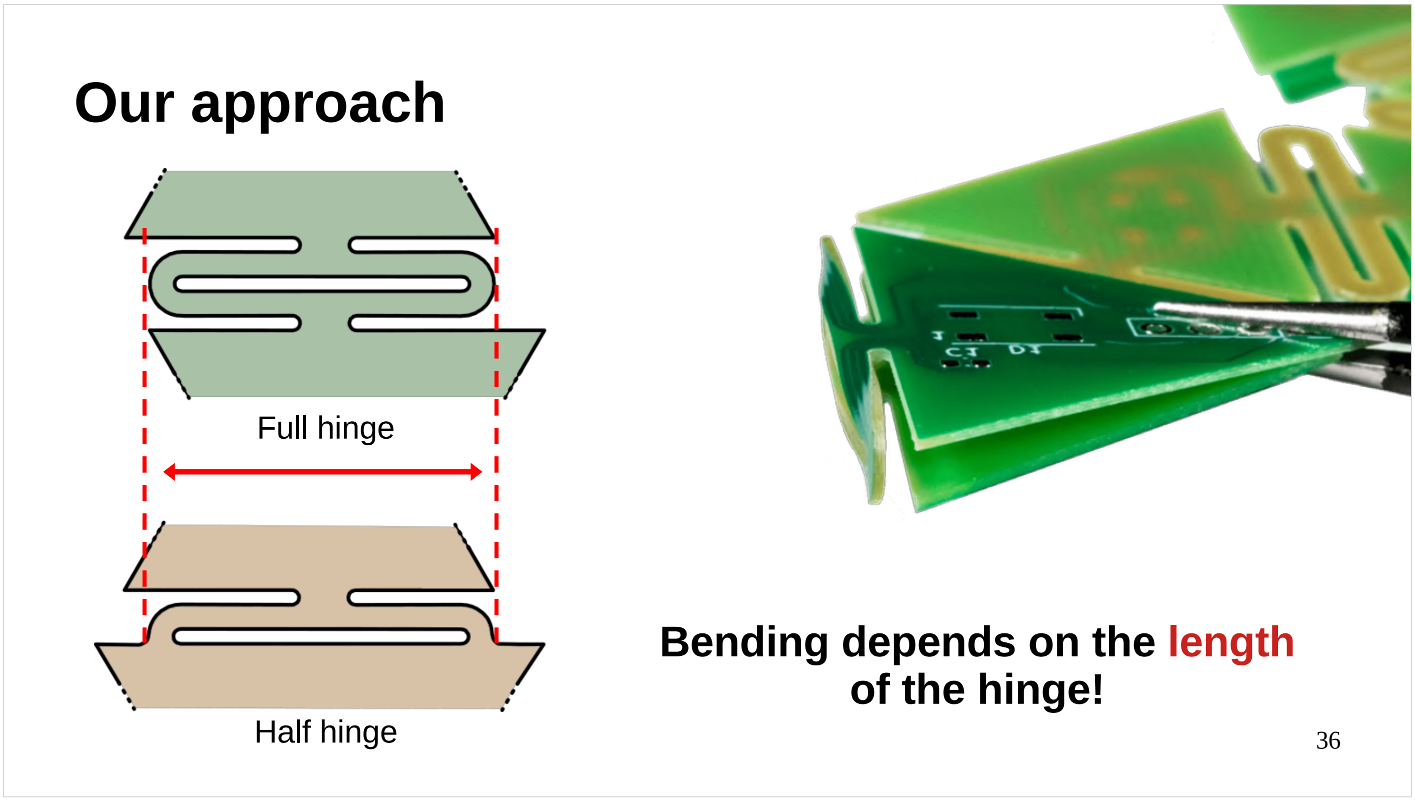


Related work

Rodriguez et al., 2022: *Computational design of laser-cut bending-active structures* & 2024: *Designing bending-active freeform surfaces*

Speetzen, Kobbelt, 2024: *Freeform shape fabrication by kerfing stiff materials*

Unlike origami and kirigami, kerfing as a technique is just starting to attract interest in computational fabrication, mostly still with wood or cardboard. Examples of this type of research include the computational design of bending-active structures from 2022 and the following 2024 paper just presented at SCF 2024; and the fabrication of freeform shapes from 2024.



Our approach

Full hinge

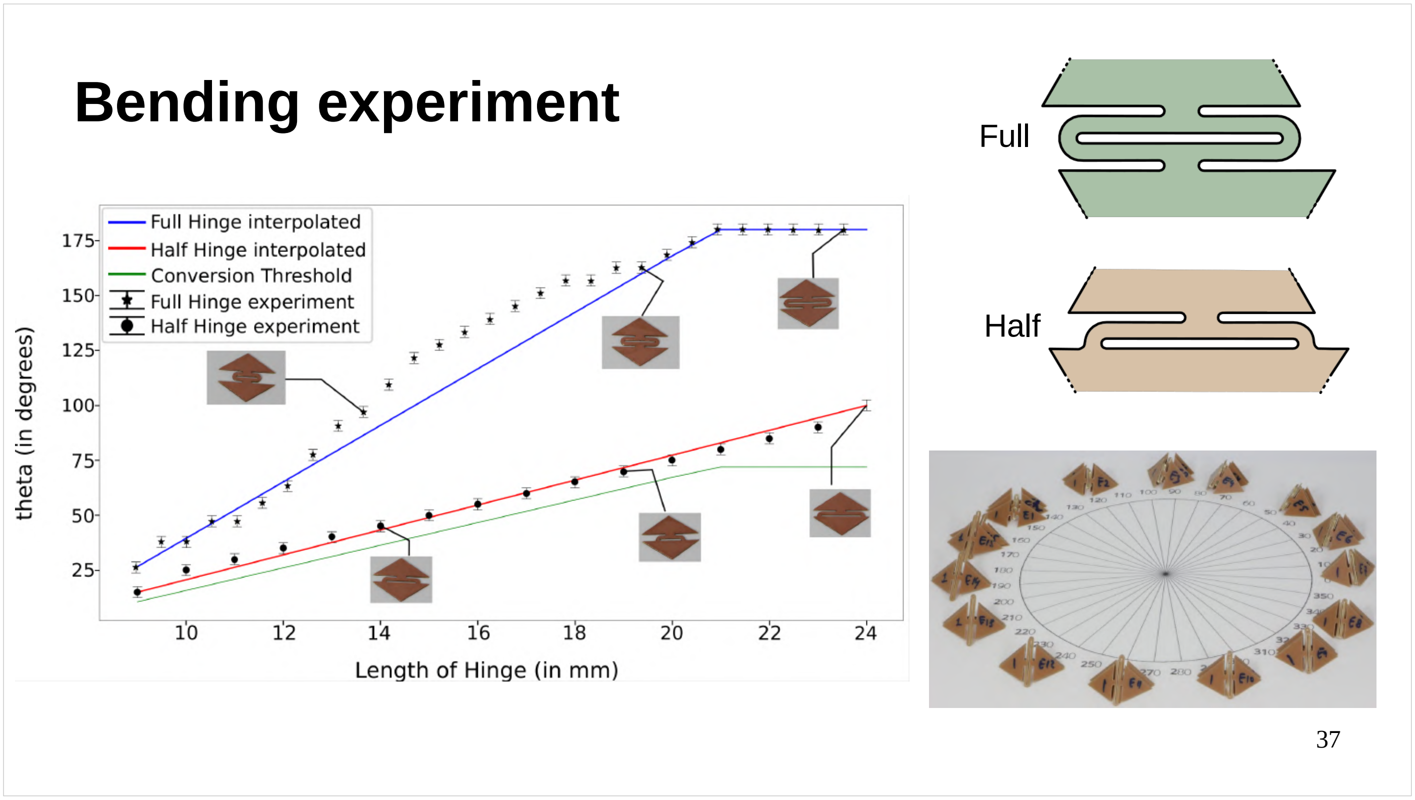
Half hinge

Bending depends on the length of the hinge!

We call our kerfing patterns **hinges**. After many iterations, we converged toward these two specific patterns, that we call **full** and **half** hinges.

These are pretty similar and consist of two separate paths connecting two solid PCB areas. [CLICK] They bend through the torsion of the middle element.

We carried out bending experiments to determine how much these hinges can be bent [CLICK] before breaking as a function of their length, represented on the left.

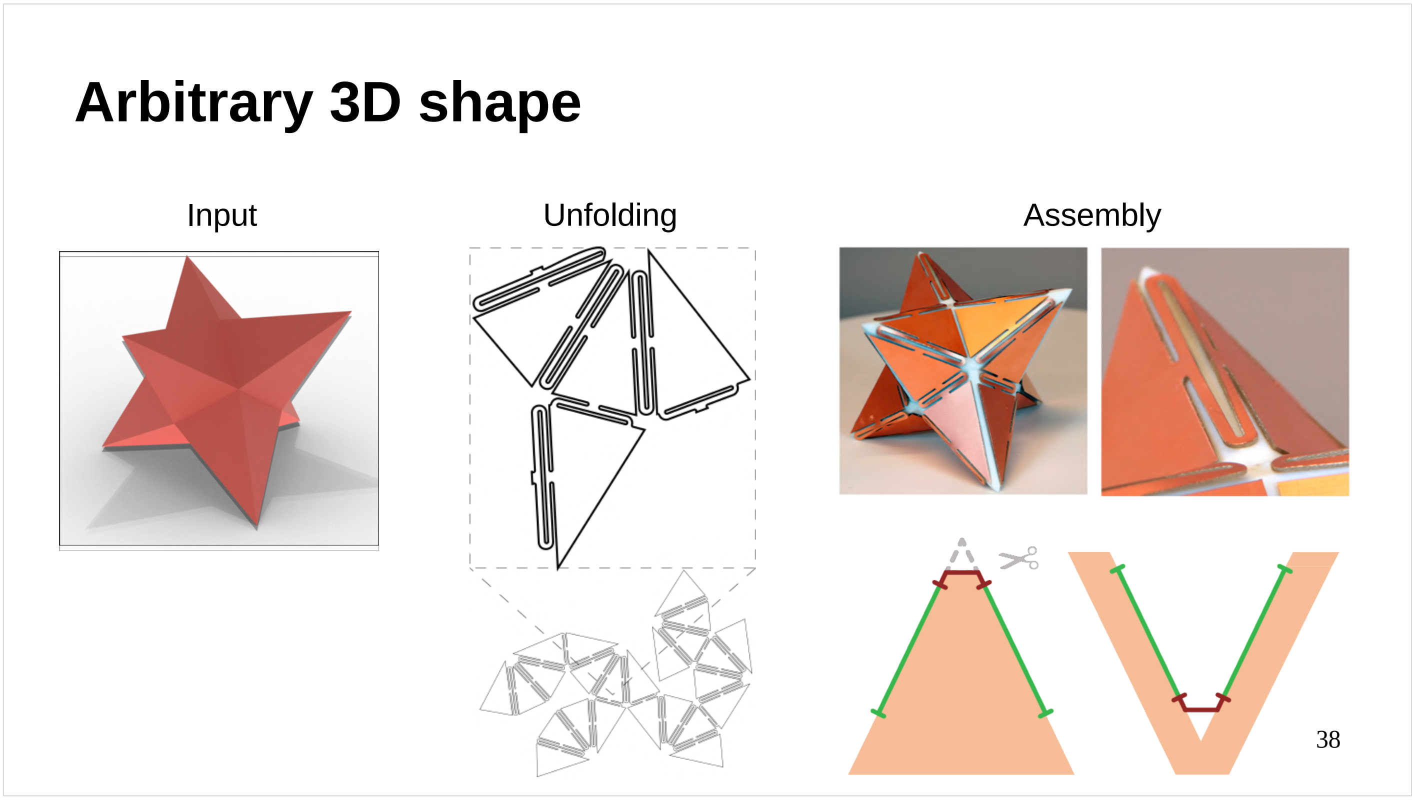


Bending experiment

These experiments showed, as we expected, that the longer the hinge is, the more it is able to bend.

We plotted here the bending angle as a function of hinge length. The top and bottom plot correspond respectively to the full and half hinge.

Based on this data, we defined a linear function relating hinge length to a safe bending angle where the hinge does not suffer any damage.



Arbitrary 3D shape

Input

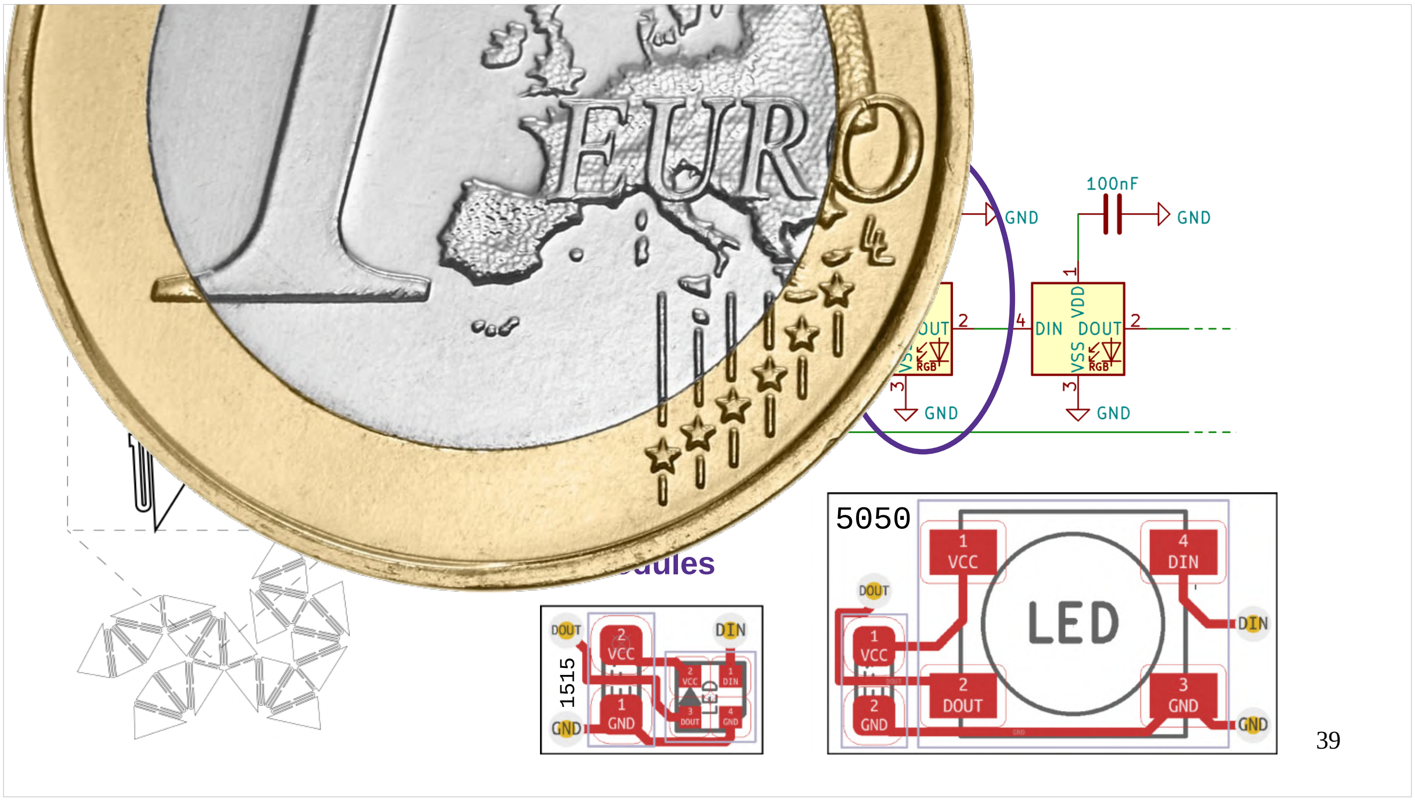
Unfolding

Assembly

These hinges allow us to create 3D objects from 2D PCBs fabricated as usual.

By taking a 3D mesh as an input, we can [CLICK] unfold it flat and insert hinges in between the triangles, and then [CLICK] fabricate it and assemble it on top of a 3D printed frame.

Note that since the hinges don’t crease like paper, but [CLICK] bend instead, the frame’s edges have to be chamfered to ensure proper assembly.

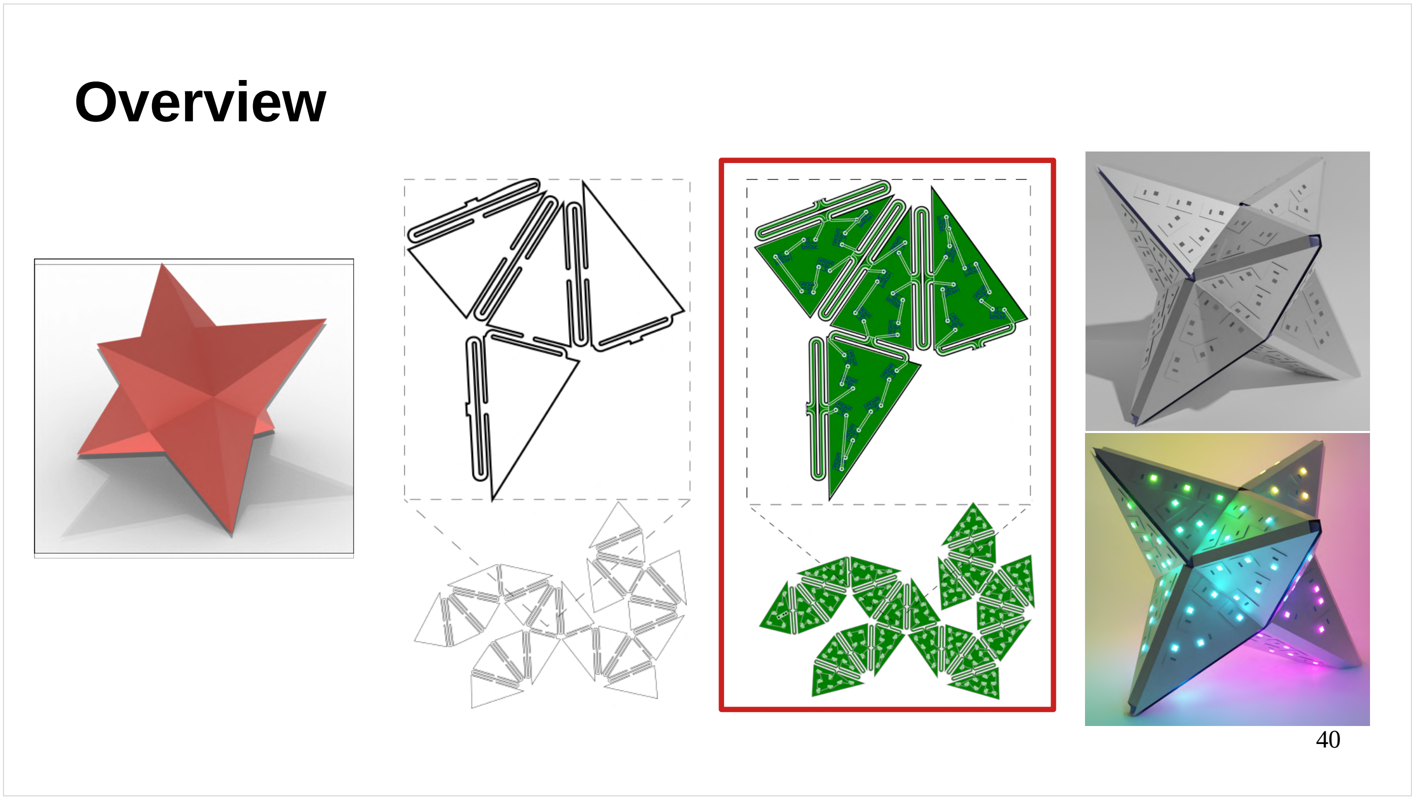


We now have an unfolded mesh, and a circuit [CLICK] to embed within it [CLICK].

Our circuit consists of simple LEDs connected in a chain. These come in two sizes [CLICK], 1.5x1.5mm2 and 5x5mm2. We use Neopixel LEDs, they are RGB, easy to control, and very common for home lighting applications such as LED strips.

We pack the LEDs into rectangular modules containing a small decoupling capacitor. As you can see [CLICK] in the schematics, each module connects to PWR and GND, and also to the previous and next module in the chain.

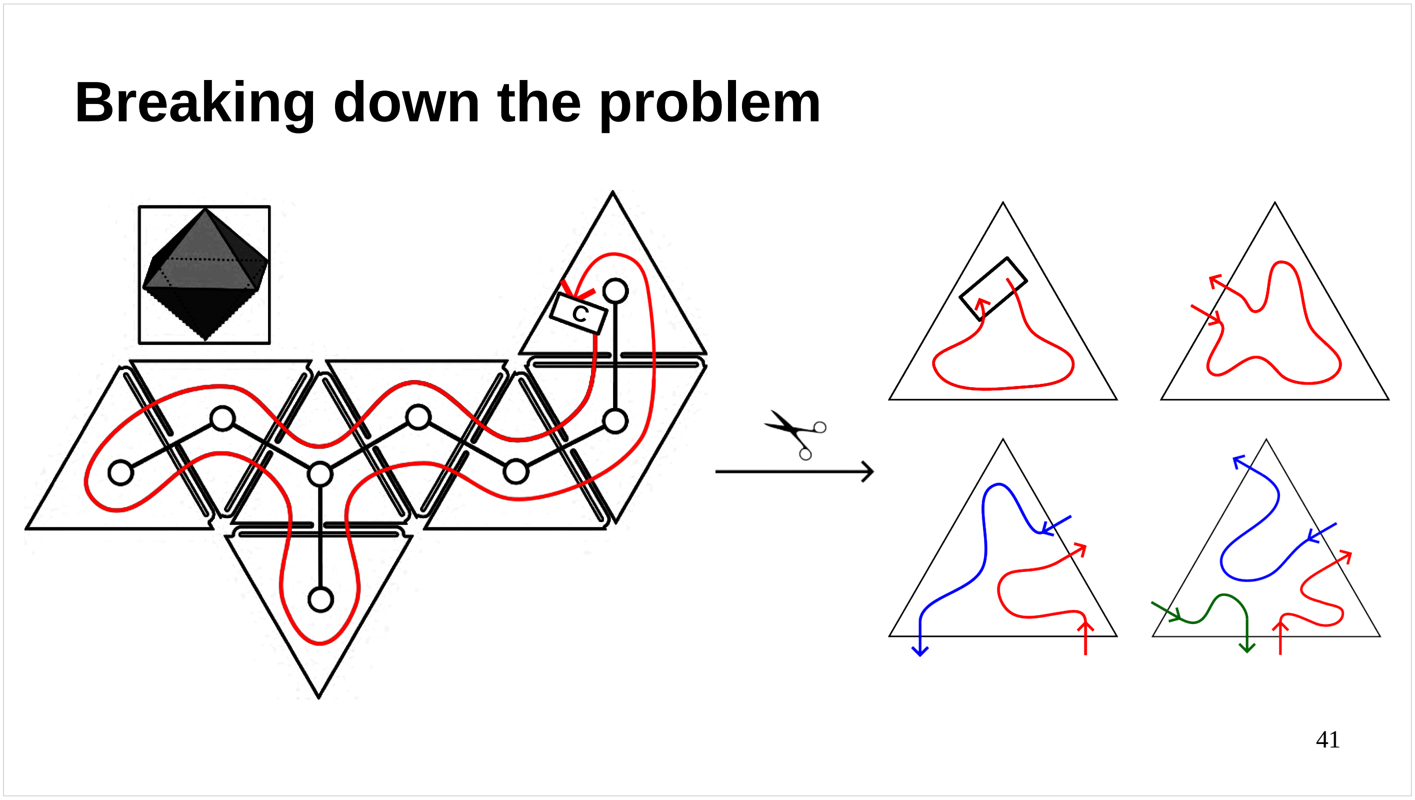
To give you a sense of their size [CLICK], this is what a 1 Euro coin looks like at the same scale as the LED modules.



Overview

With this, we have a full pipeline from an input mesh to a fabricated object. First, [CLICK] the input mesh is unfolded and hinges are inserted to enable folding. Next, [CLICK] a functional electronic circuit is generated in the unfolded shape, resulting in fabrication blueprints. [CLICK] The circuit can be sent for fabrication, and after folding onto a 3D printing scaffolding, it can be used to design lighting effects. I will focus now on our circuit layout generation method.

Generating a circuit layout in the unfolding is a complex task. Unfoldings can be pretty large, having up to hundreds of triangles, with a complex shape due to the hinges. This makes placing the components in the whole circuit at once too expensive and makes automatic routing non-applicable. We break down the problem into smaller tractable instances.

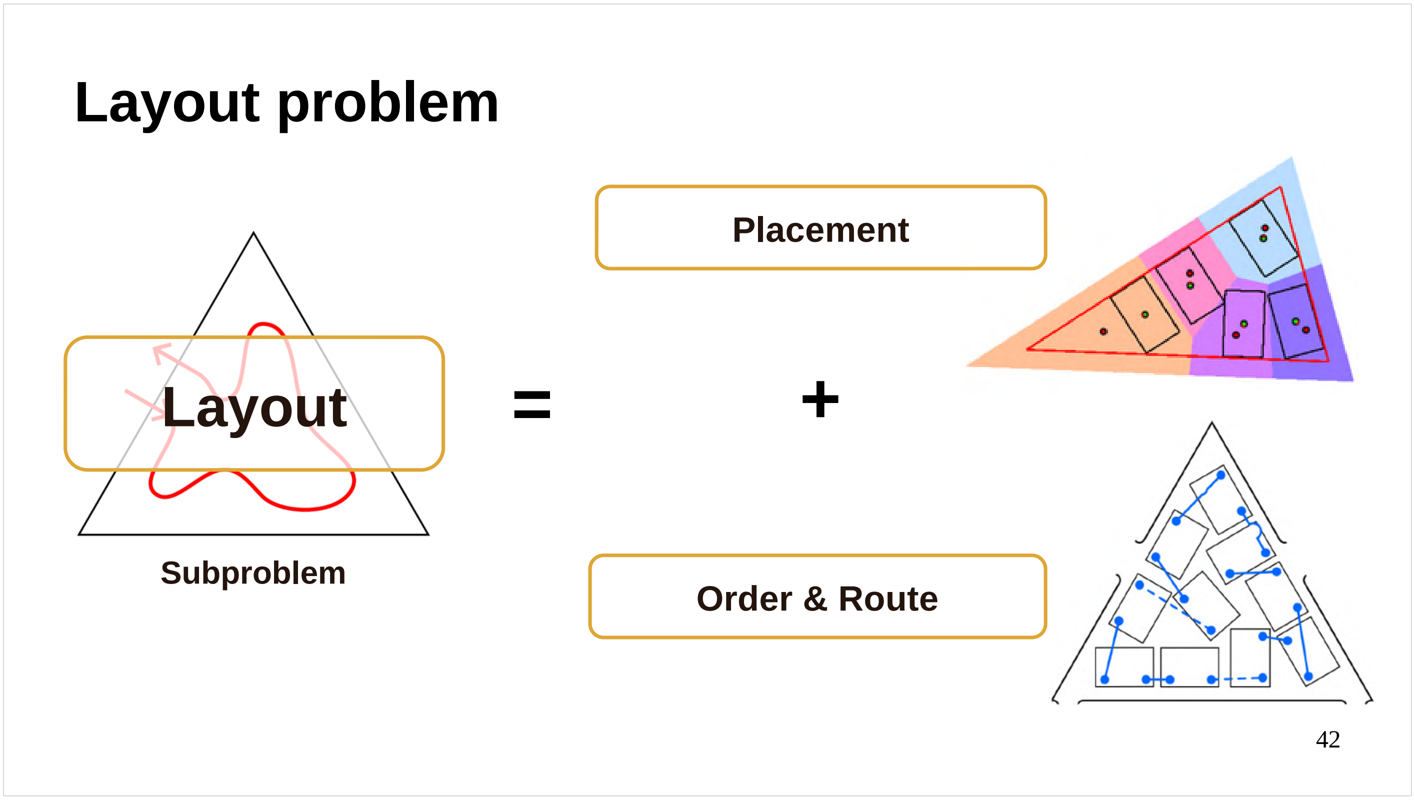


Breaking down the problem

41

As you can see [CLICK], the unfolding is a tree. Also, the LEDs are connected sequentially, starting and ending at the connector [CLICK] represented by a rectangle. Using these properties, by making the chain always “go on the right side of the hinge” [CLICK], we ensure that it goes into every triangle and passes twice through every hinge, once in each direction. [CLICK] This allows us to split [CLICK] the global layout problem into separate subproblems for every triangle.

The subproblem [CLICK] is then uniquely determined by the shape of the triangle and how and where the chain enters and exits it. The colored arrows represent separate LEDs subchains.



Layout problem

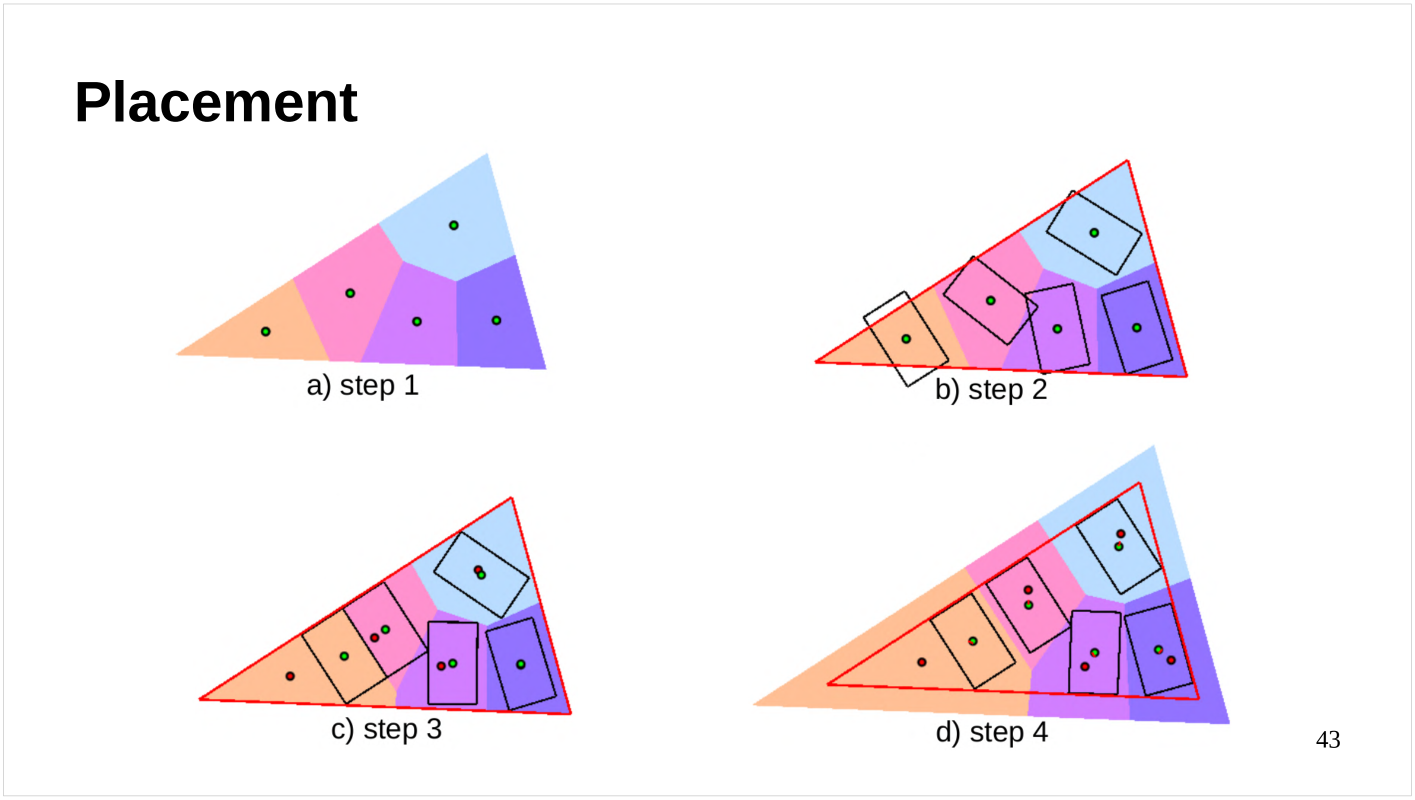
Layout = Placement + Order & Route

Subproblem

From this subproblem information, our goal is to generate a local layout. For this we first [CLICK] need to place the LEDs within the triangle, and then [CLICK] order and route them jointly to form LED subchains connected to the inputs and outputs.

This does not correspond to the typical workflow of electronics design, namely: schematics design, placement and routing. Our global circuit is not fully defined until the end of the layout phase. Before placement, we don’t know the exact number of components of the circuit. And before ordering & routing, we do not know the order of the specific LEDs in the chain.

Let’s quickly go over our method for these two phases:



Placement

a) step 1

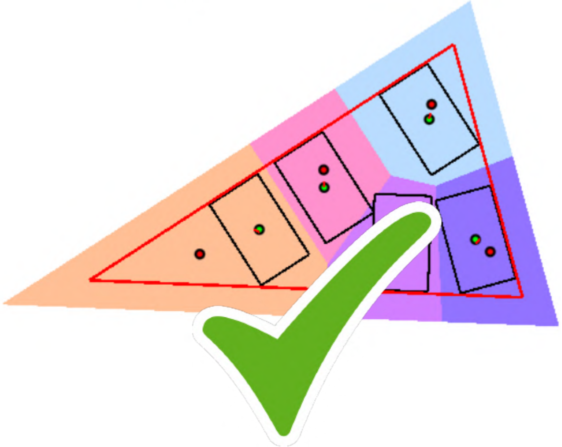
b) step 2

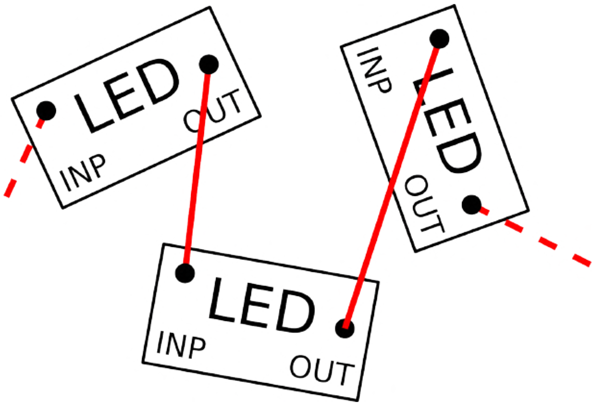
c) step 3

d) step 4

LED placement starts with a [CLICK] Centroidal Voronoi Tessellation (or CVT) in the triangle that distributes a set of points around. Then, the LED modules [CLICK] are spawned at the point locations, and later [CLICK] collisions are resolved. A final CVT [CLICK] step is used to redistribute the modules, fully exploiting the available space.

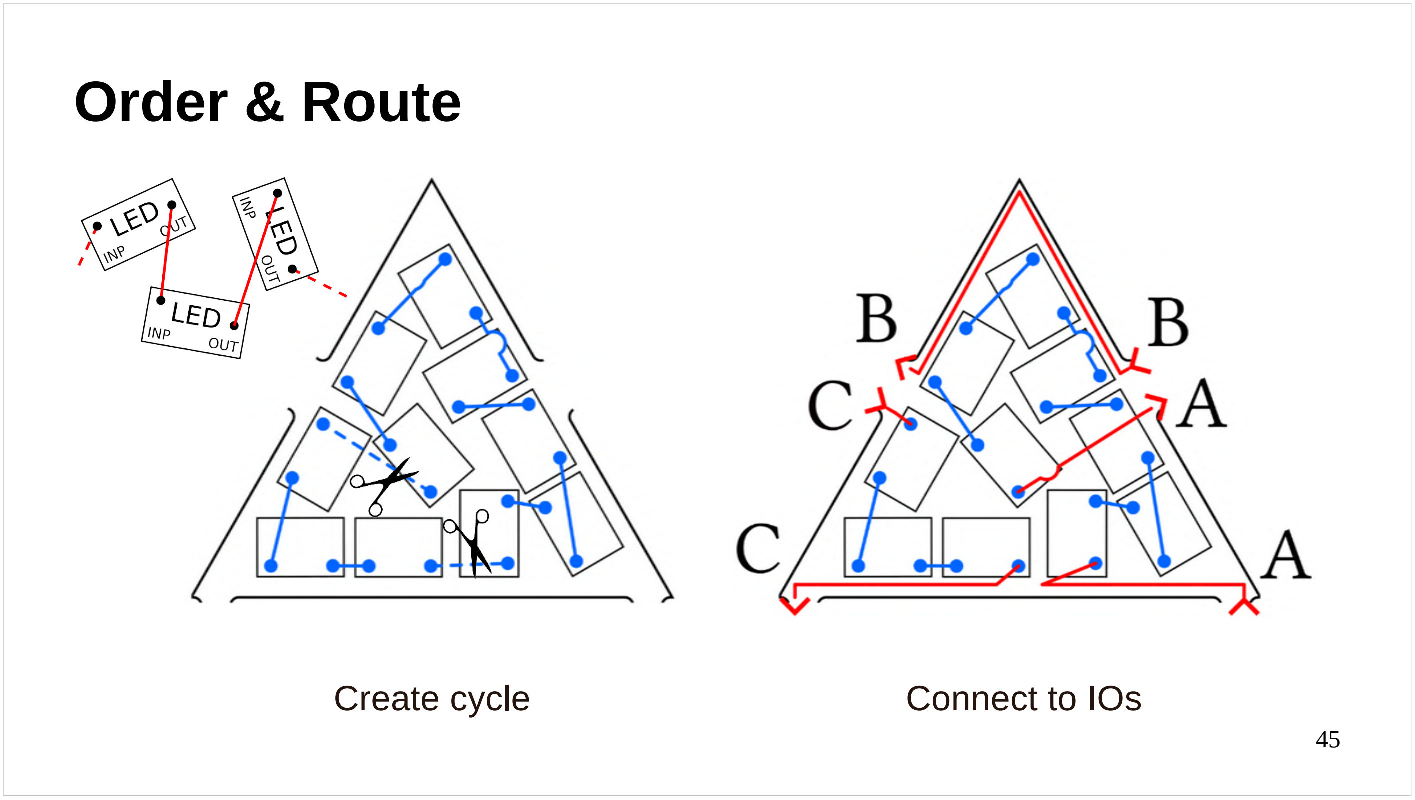
Order & Route





Orientation? Order?

Placing rectangles [CLICK] is only half of the job though, we need to connect them now. LEDs have an input and an output [CLICK], and thus two possible orientations. They also have an order: the output of one has to be connected to the input of the next.



Order & Route

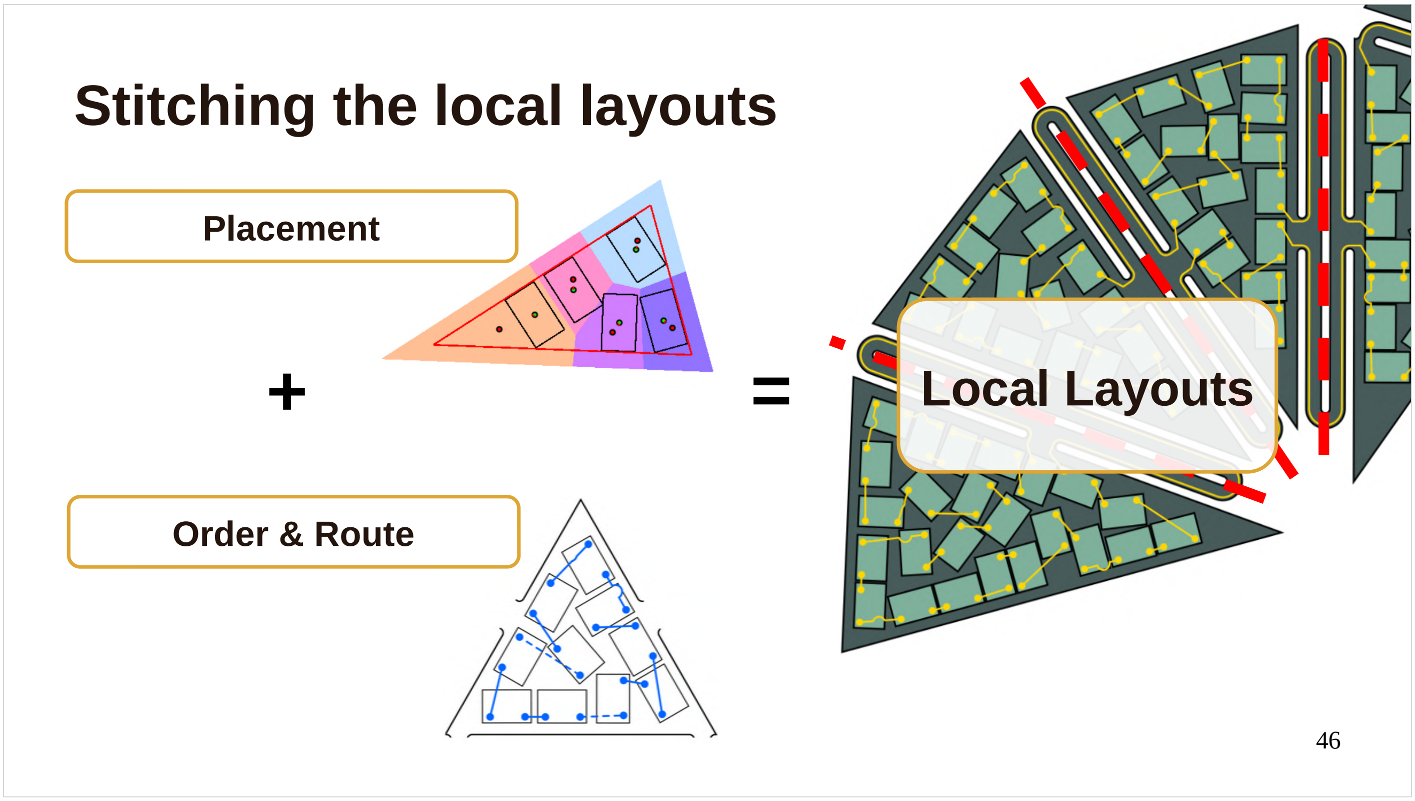
Create cycle

Connect to IOs

Choosing the order and the orientations [CLICK] gives us a cycle. The order is obtained with a TSP solver, using the distances between the centers of the modules as the metric. The orientations are then optimized so that there are no intersections between the segments in blue connecting the LEDs.

The cycle then needs to be connected to the inputs and outputs of the triangle. We split it [CLICK] to create separate subchains, each connected to an input and output [CLICK].

Here we cut the dashed lines on the left and connect them to the inputs and outputs with the red arrows on the right. Each subchain is represented by a different letter.



Stitching the local layouts

Placement

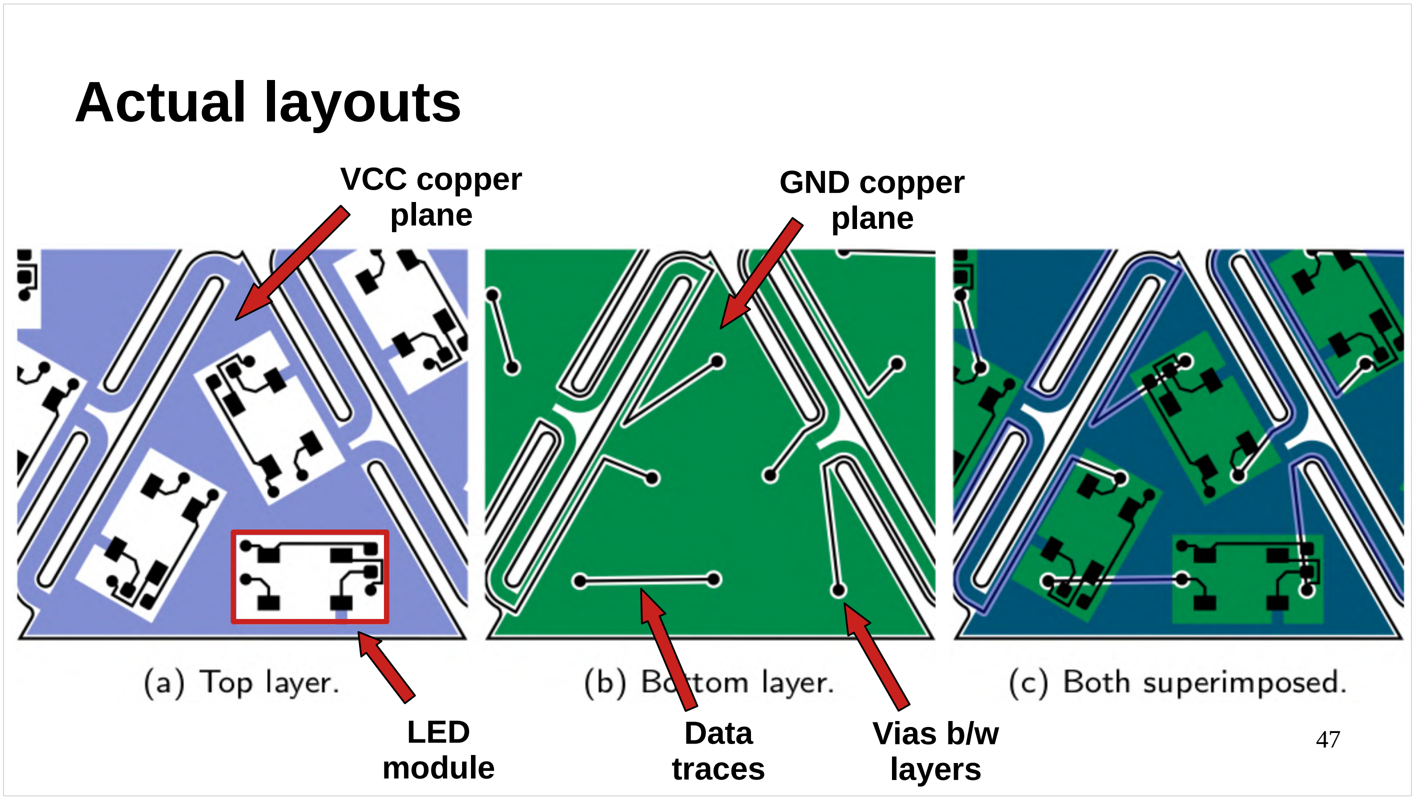
+

Order & Route

=

Local Layouts

Finally, after placement, ordering and routing, we have all local layouts [CLICK]. The last step stitches them [CLICK] into a single global layout that can be sent for fabrication.



Actual layouts

VCC copper plane

GND copper plane

(a) Top layer.

(b) Bottom layer.

(c) Both superimposed.

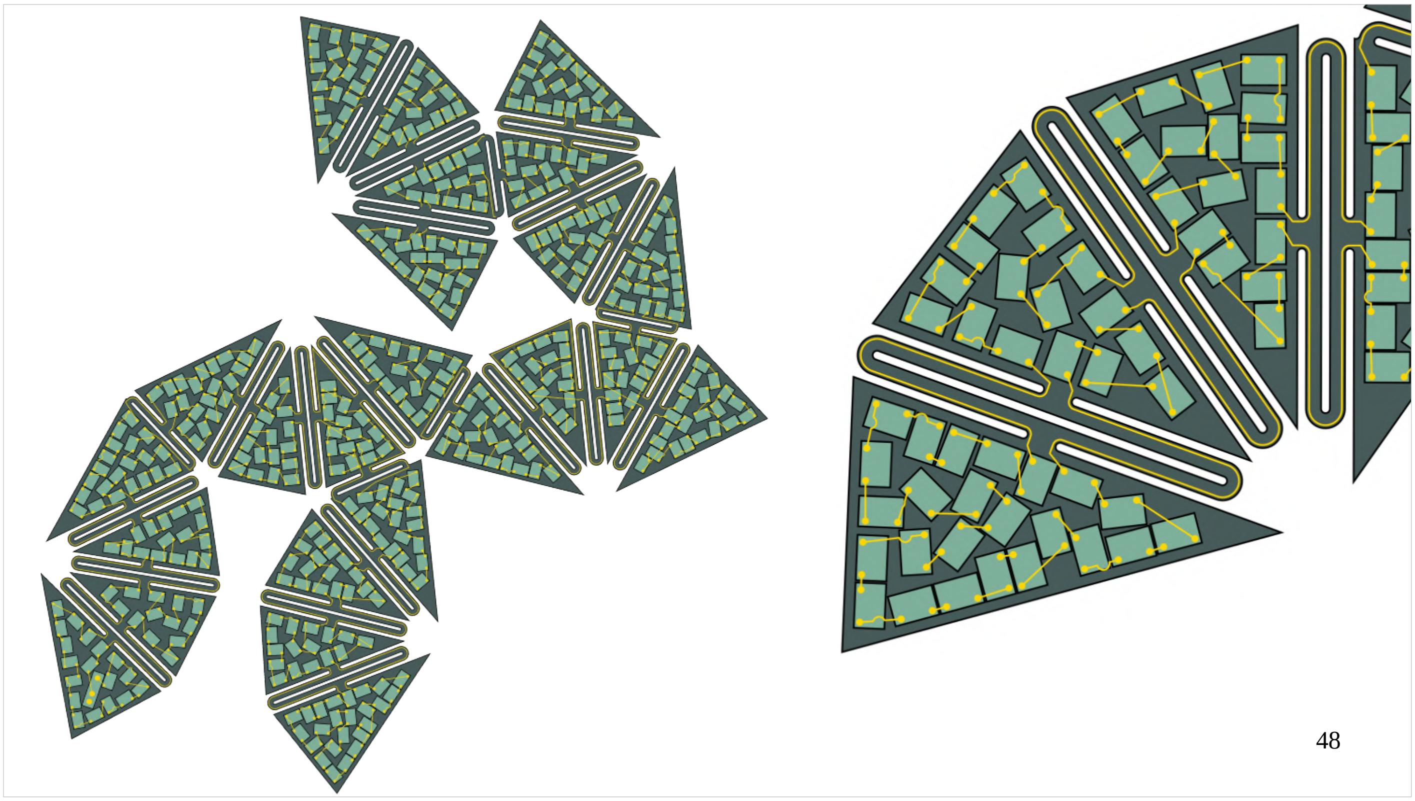
LED module

Data traces

Vias b/w layers

If you remember the circuit schematics, you will have realized that I have conveniently ignored PWR and GND.

Here you can see an actual triangle layout that we generated. On the top layer, [CLICK] we have the LED modules surrounded by the power copper plane. [CLICK] On the bottom layer, we have [CLICK] the data traces connecting to the LEDs through vias between layers, all surrounded by the ground copper plane (green). This [CLICK] is what the two layers look when superimposed.



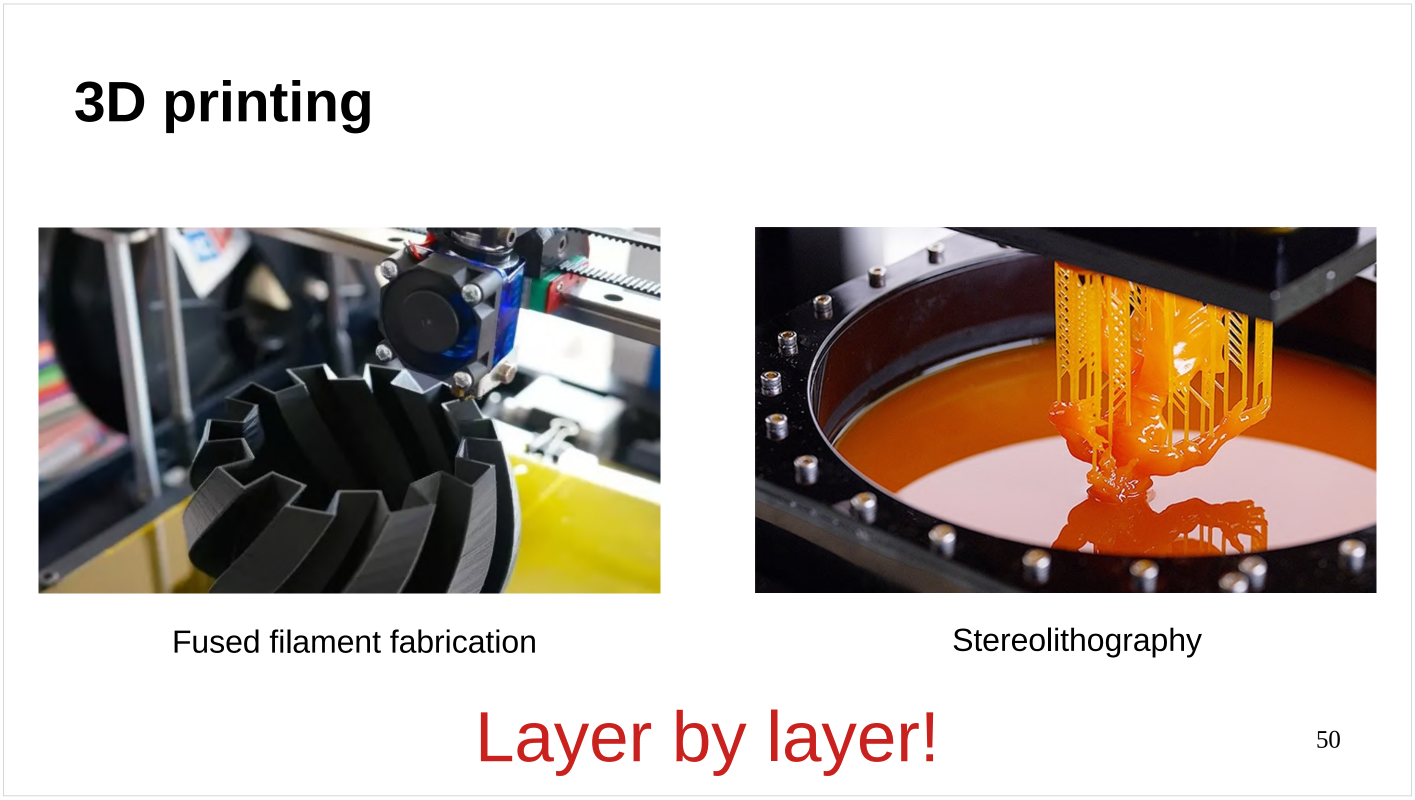
Here is a simplified example of the complete layout for the star mesh you have seen in previous slides (hiding the ugly truth). After the physical PCB is received, the last step is assembling the object.

Procedural bridges-and-pillars support generation

M. Freire, S. Hornus, S. Perchy, S. Lefebvre
Université de Lorraine, CNRS, Inria, LORIA, France

[~28min]

This contribution deals with the procedural generation of supports for 3D printing, that resulted in a 2022 Eurographics short paper.



3D printing

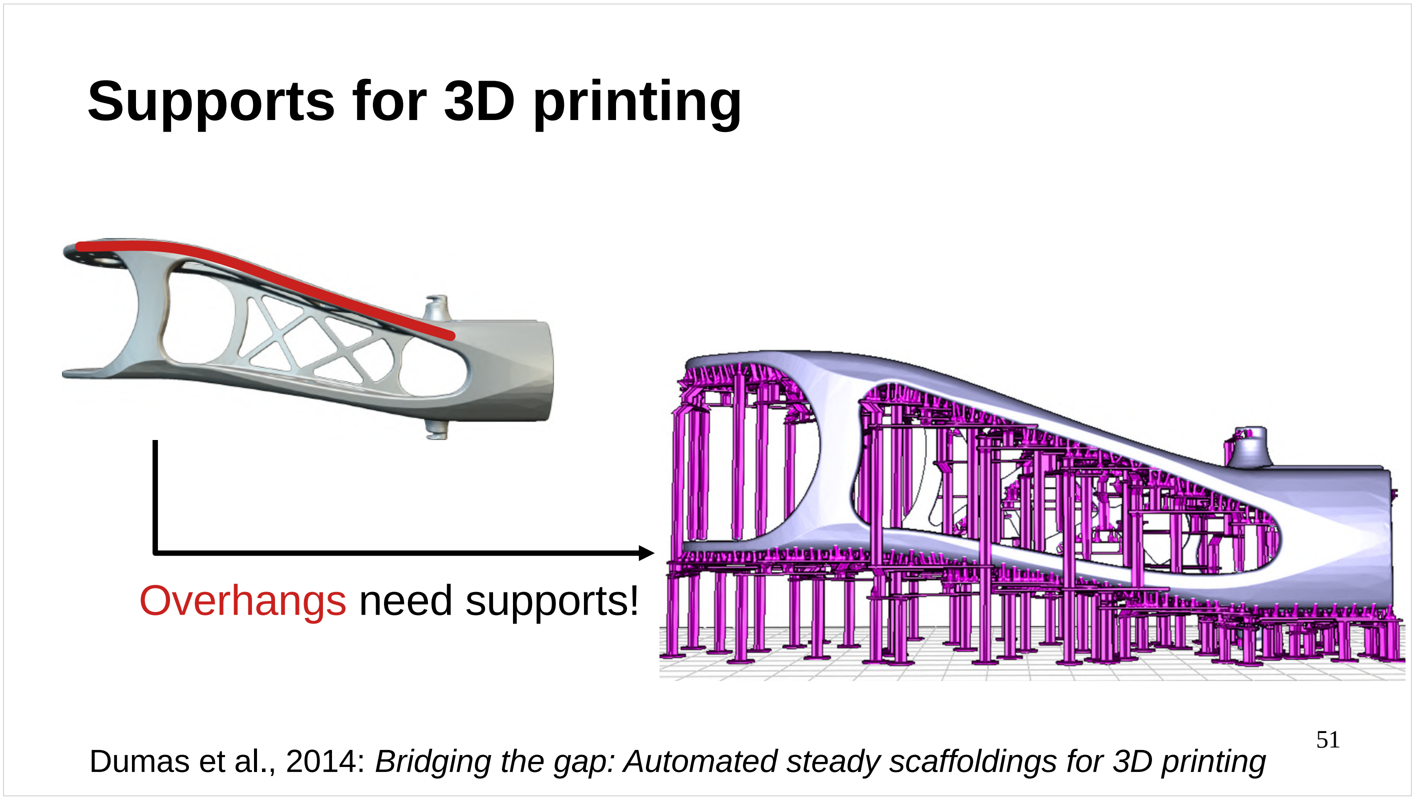
Fused filament fabrication

Stereolithography

Layer by layer!

As I said in the introduction, 3D printing allows us to create physical shapes from a digital model through various techniques such as Fused Filament Fabrication or Stereolithography.

[CLICK] The fact that these technologies print objects layer by layer impact what type of shape can be printed.



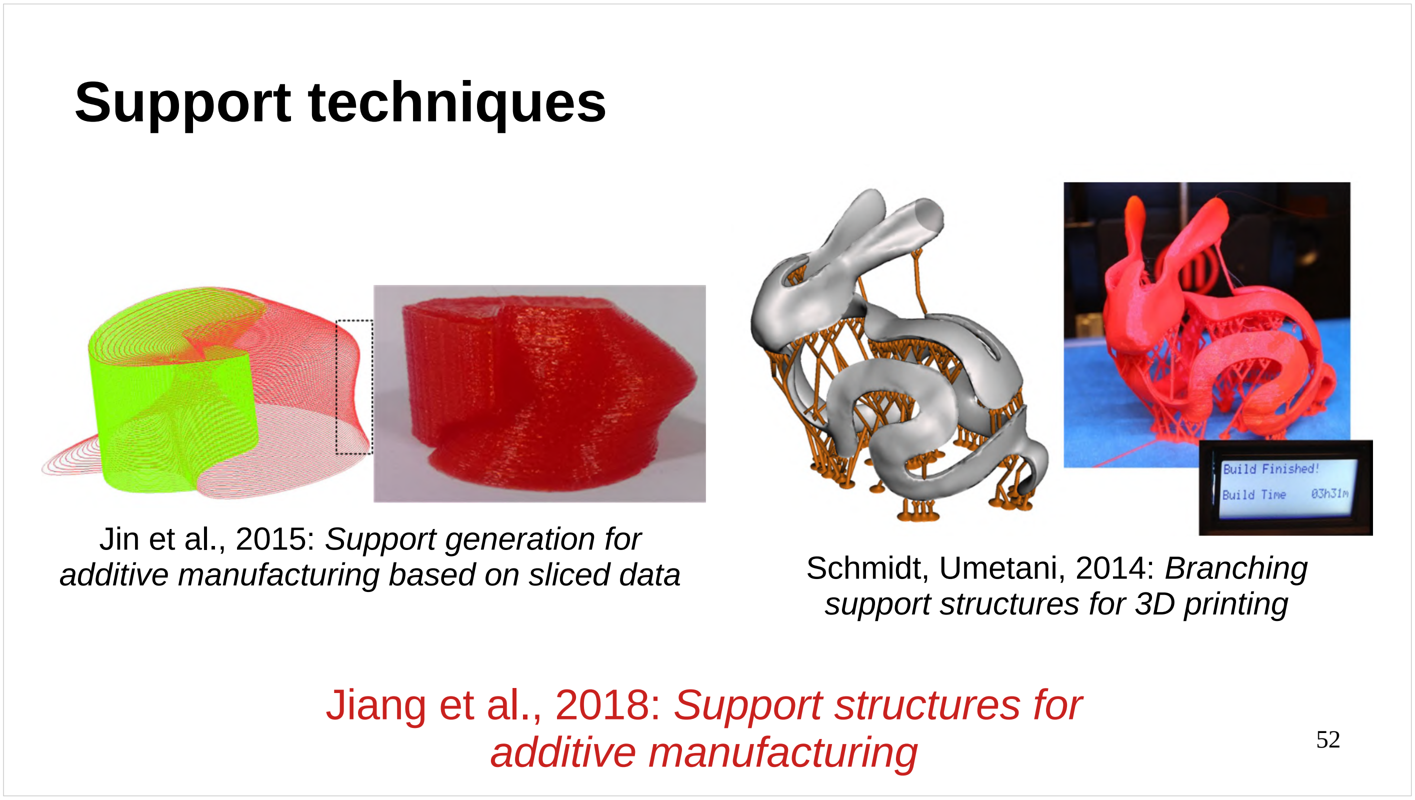
Supports for 3D printing

Overhangs need supports!

Dumas et al., 2014: *Bridging the gap: Automated steady scaffoldings for 3D printing*

This is a problem for objects containing steep overhangs and local minima. These [CLICK] need support structures to be printed, or else we get this [CLICK] type of result.

Support structures [CLICK] help support overhangs in the object during printing.



Support techniques

Jin et al., 2015: *Support generation for additive manufacturing based on sliced data*

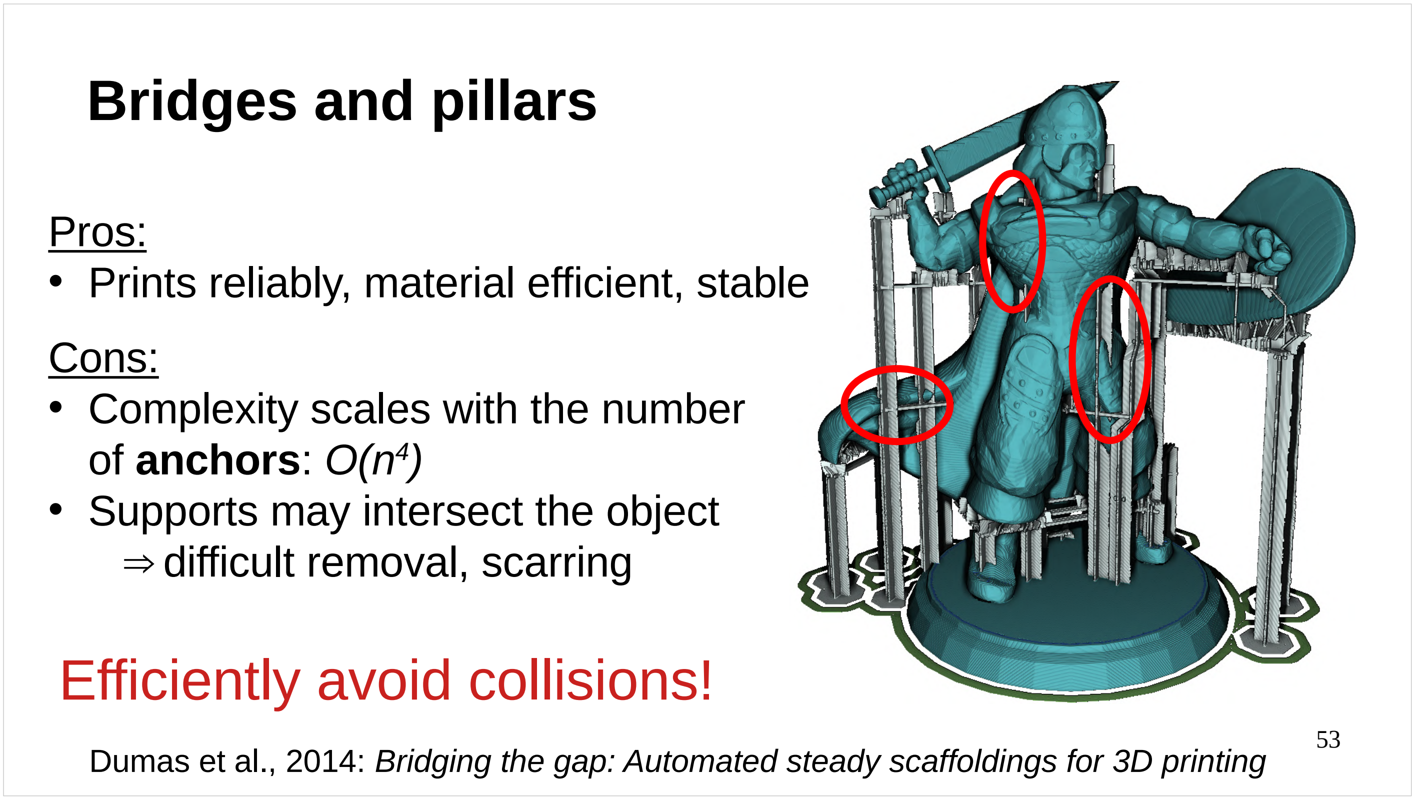
Schmidt, Umetani, 2014: *Branching support structures for 3D printing*

Jiang et al., 2018: *Support structures for additive manufacturing*

There are many support generation techniques, balancing reliability, sturdiness, printing time, material cost and plenty of other factors.

A core challenge is minimizing support size while maintaining support reliability. Here are two opposite approaches. On the left you have a dense support structure, while on the right you have a sparse, tree-like support structure.

Support generation and optimization is a complex topic that I will not explore further today. You can refer to the 2018 review for more details.



Bridges and pillars

Pros:

- Prints reliably, material efficient, stable

Cons:

- Complexity scales with the number of **anchors**: $O(n^4)$
- Supports may intersect the object
  ⇒ difficult removal, scarring

Efficiently avoid collisions!

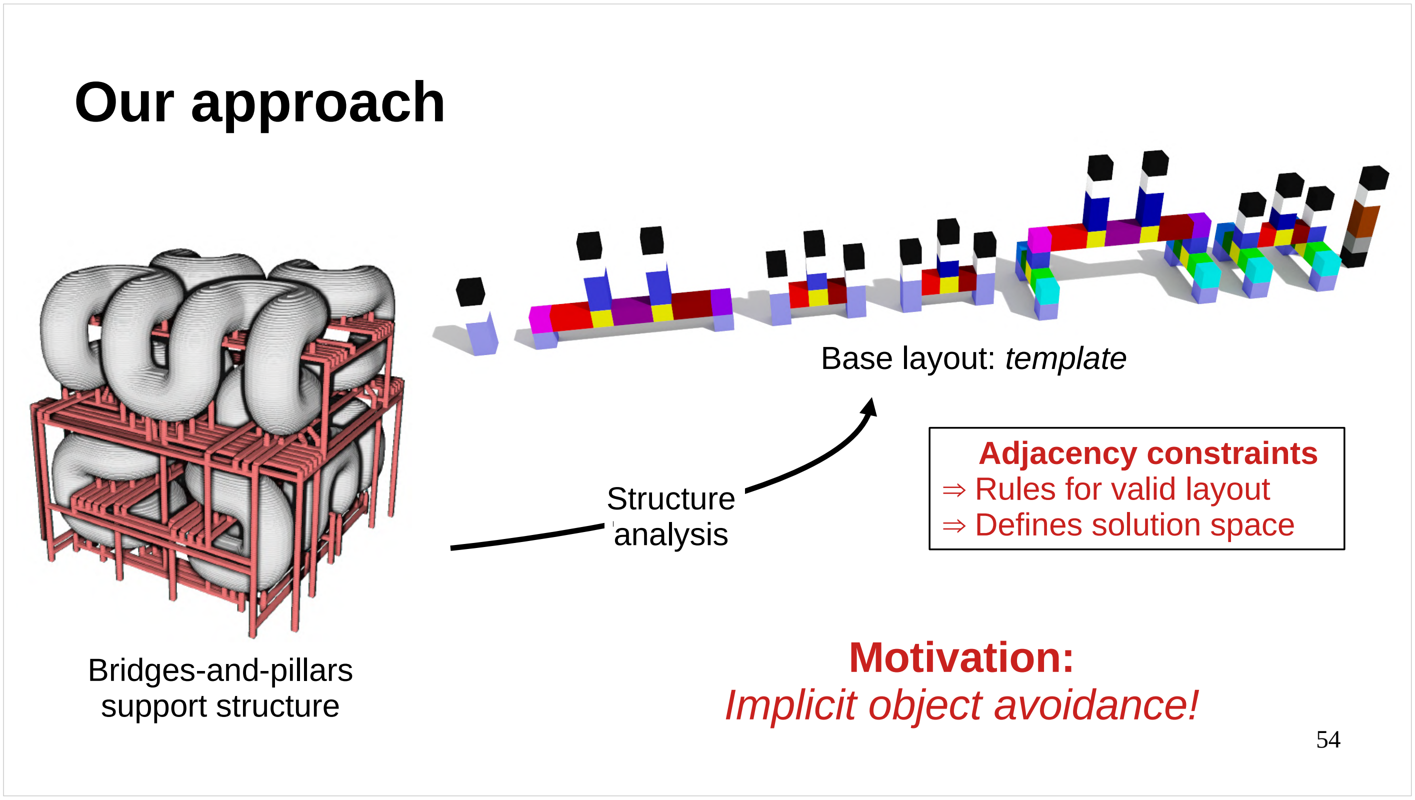
Dumas et al., 2014: *Bridging the gap: Automated steady scaffoldings for 3D printing*

We specifically improve the generation of bridges-and-pillars supports introduced by Dumas et al in 2014. These are reliable, material efficient and stable during printing.

[CLICK] Their algorithm scales steeply with the number of points to support, or anchors, as I will call them. This is a problem for objects with many overhanging features.

[CLICK] Also, the public implementation of the method does not avoid intersections between the object and the support. This makes support removal harder and leaves scars on the printed object. You can see on the right circled in red, zones where the supports intersect the figurine.

[CLICK] We aim to generate the same kind of structures while efficiently avoiding collisions!



Our approach

Base layout: *template*

Structure analysis

Adjacency constraints
⇒ Rules for valid layout
⇒ Defines solution space

Bridges-and-pillars support structure

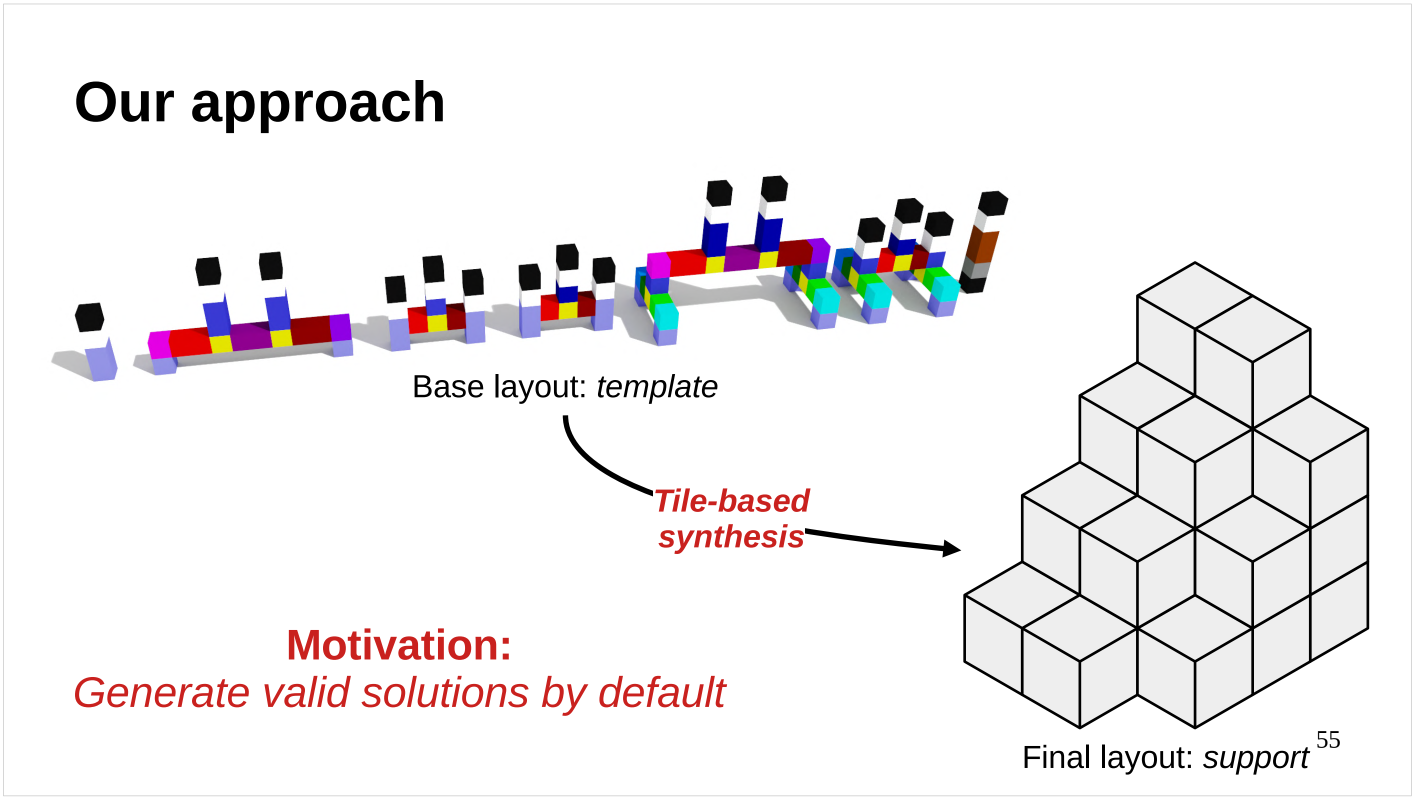
Motivation:
Implicit object avoidance!

We accomplish this by casting it as a layout problem.

We analyzed the bridges-and-pillars support structure to extract a base layout consisting of building blocks or labels, represented in different colors. This acts as a template, specifying how these labels can be assembled through adjacency constraints.

The template [CLICK] defines the rules for a valid layout, which in turns defines the solution space inhabited by our support structures.

This was motivated by [CLICK] wanting the support structures to avoid the object. With this approach, we can define the object and the supports as different labels that cannot be adjacent except at anchors needing support.



Our approach

Base layout: *template*

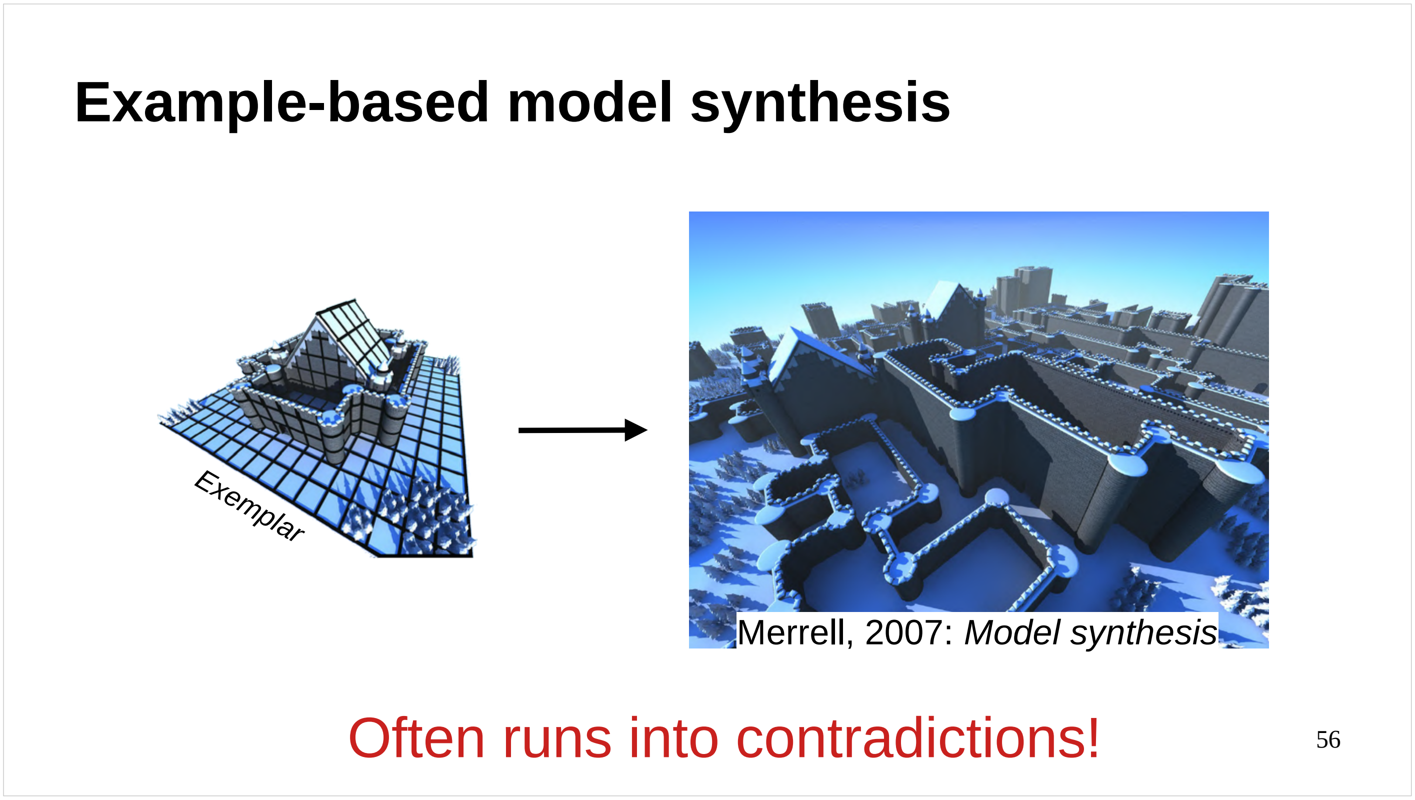
Tile-based synthesis

Motivation:

Generate valid solutions by default

Final layout: *support*

From this template defining the solution space, we use a tile-based synthesis algorithm [CLICK] called Model Synthesis to generate support instances that satisfy our design constraints by default.



Example-based model synthesis

Merrell, 2007: *Model synthesis*

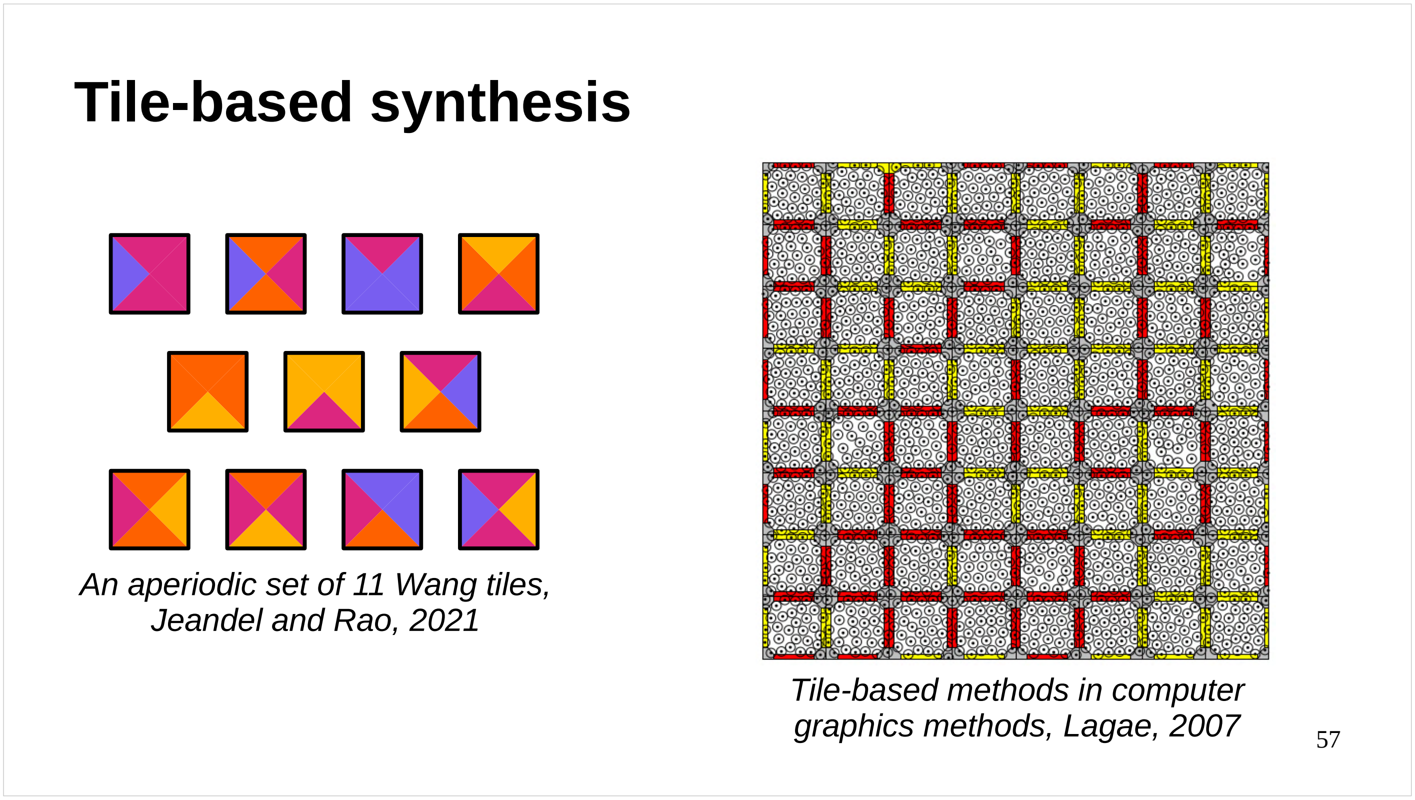
Often runs into contradictions!

56

Model Synthesis generates content similar to an provided exemplar by extracting adjacency information from it. Here you can see the result of Model Synthesis applied to a small castle tileset, generating a large 3D castle-like structure.

The main issue with these algorithms for our application is that [CLICK] they often encounter contradictions at runtime. These happen when in a given cell, all tiles break the constraints.

This is normally solved by backtracking or by restarting from scratch. Merrell realized that the larger the domain is, the more likely it is that the algorithm will run into a contradiction, making them very likely for our application.



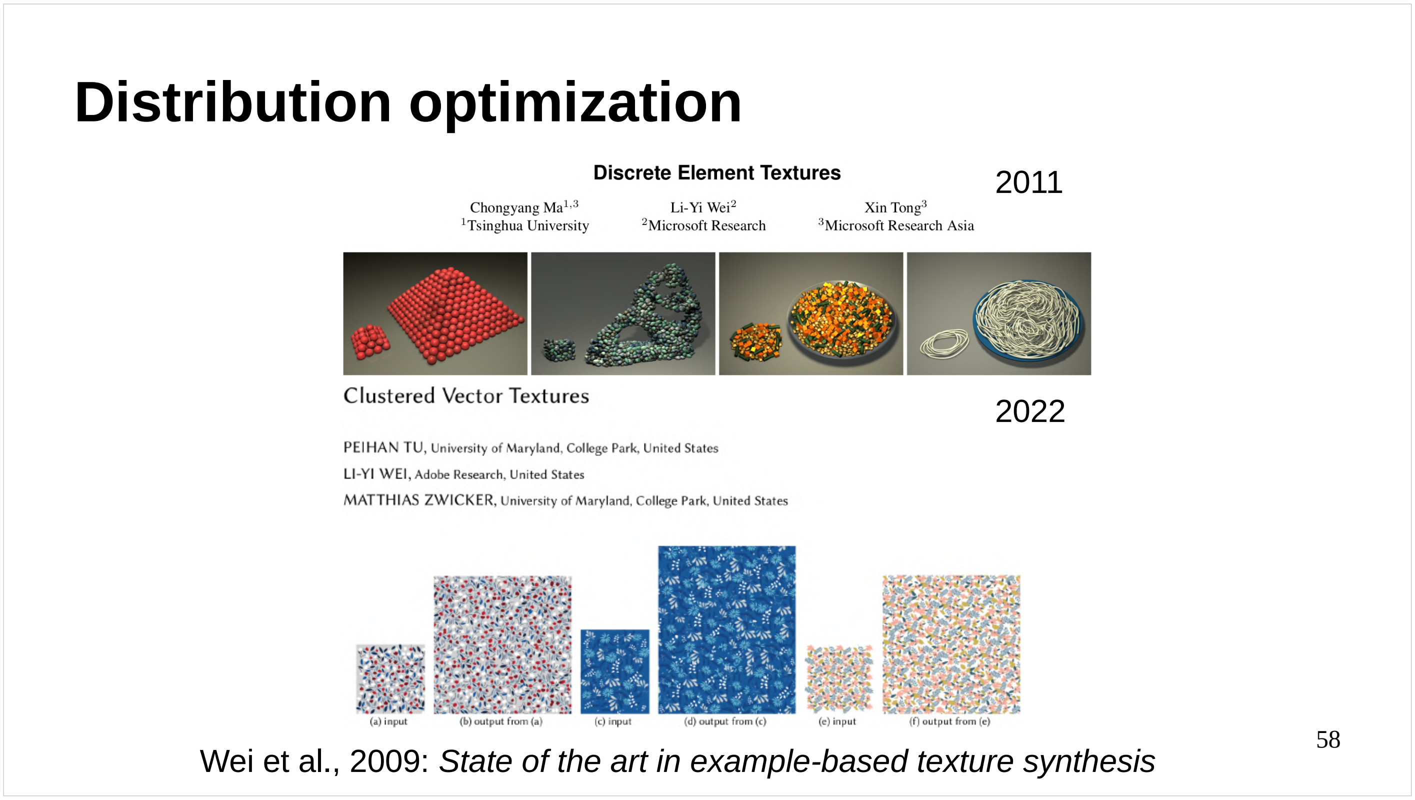
Tile-based synthesis

An aperiodic set of 11 Wang tiles, Jeandel and Rao, 2021

Tile-based methods in computer graphics methods, Lagae, 2007

Tile-based techniques are common in computer graphics. Here on the left is a set of Wang tiles that can be assembled if they share the same color along their edges. This set of tiles is aperiodic, meaning that the resulting tiling of the plane does not contain repeating patterns.

This allows generating [CLICK] diverse content at an arbitrary size by generating the content over the small set of tiles, and then tiling the space with these tiles.



Distribution optimization

Discrete Element Textures

2011

Chongyang Ma[1,3] [1]Tsinghua University
Li-Yi Wei[2] [2]Microsoft Research
Xin Tong[3] [3]Microsoft Research Asia

Clustered Vector Textures

2022

PEIHAN TU, University of Maryland, College Park, United States
LI-YI WEI, Adobe Research, United States
MATTHIAS ZWICKER, University of Maryland, College Park, United States

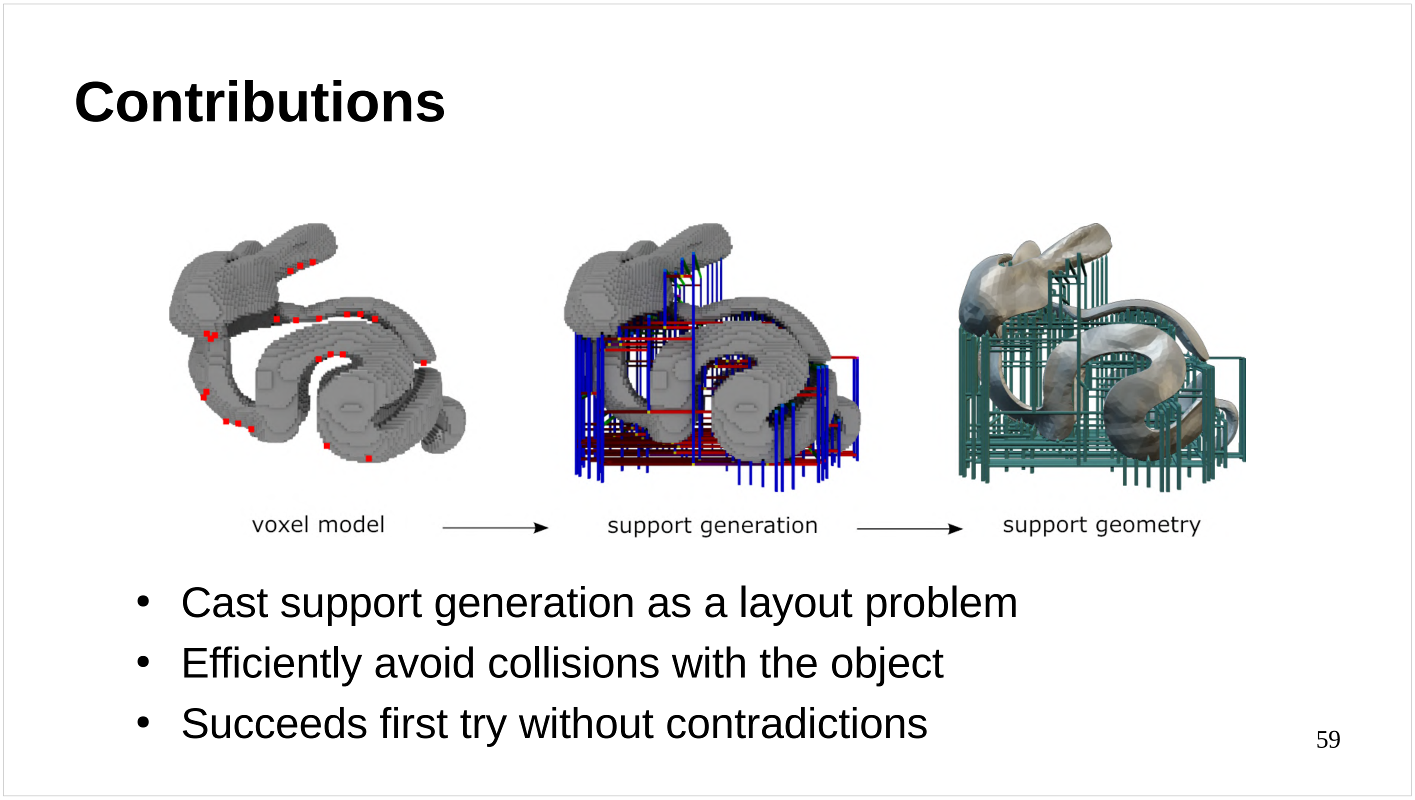
(a) input (b) output from (a) (c) input (d) output from (c) (e) input (f) output from (e)

Wei et al., 2009: *State of the art in example-based texture synthesis*

Other texture synthesis techniques can be seen as layout solvers.

These two generate structured textures resembling a provided input. This could be interpreted as an object packing problem, which is fundamentally a layout problem.

Despite the similarities, these are not suited for our purpose due to our strict discrete local constraints defining the support structures. There is no concept of connections between elements, only spatial distributions. For more details on this field, please refer to this 2009 state of the art.



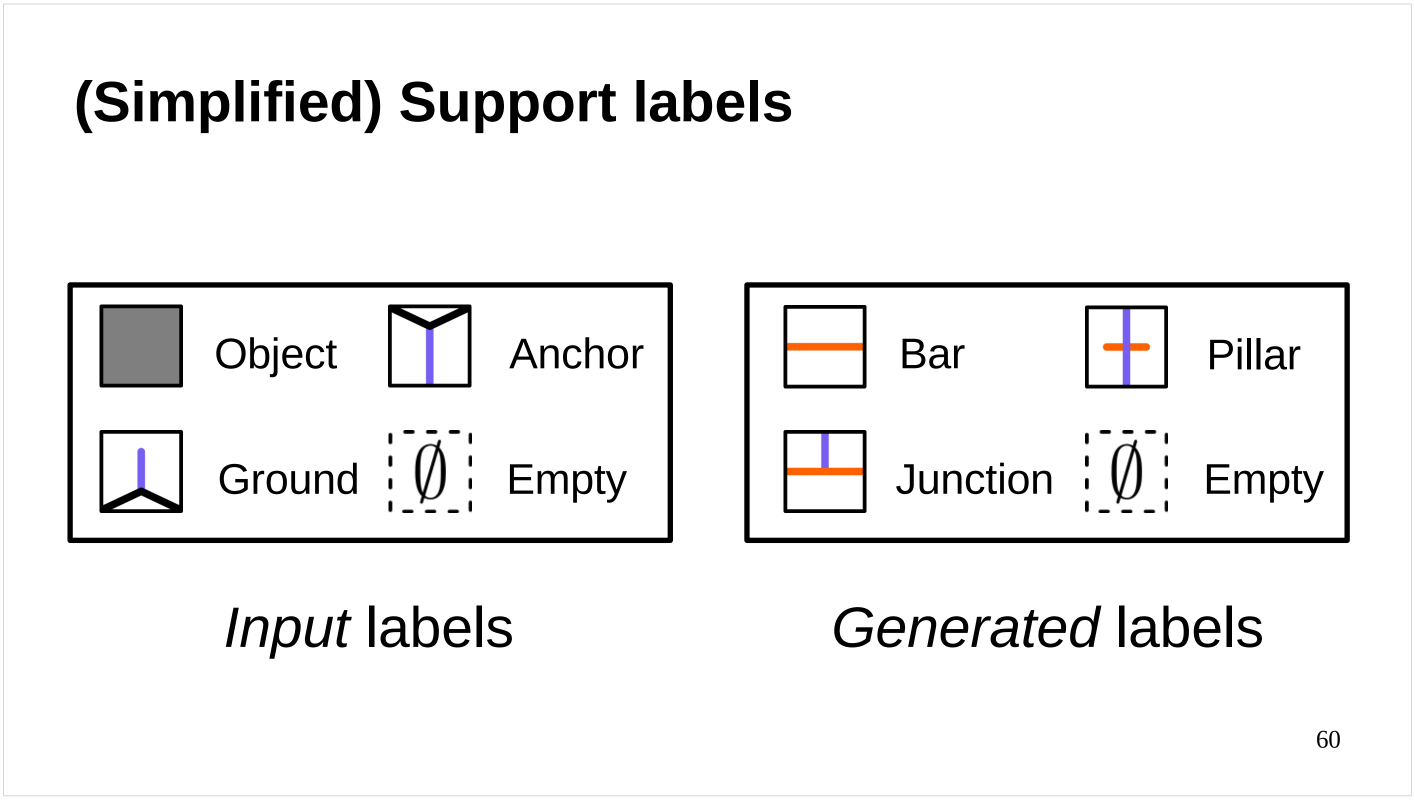
Contributions

- Cast support generation as a layout problem
- Efficiently avoid collisions with the object
- Succeeds first try without contradictions

Our contribution is a support generation algorithm inspired by model synthesis that creates a bridges-and-pillars support structure for any given voxel object.

Due to the formulation as a layout problem with adjacency information, collision avoidance is implicit and incurs no extra cost. Finally, we manage to avoid contradictions during synthesis, always succeeding first try without trial-and-error.

This method was implemented within the IceSL slicer developed by the MFX team.



(Simplified) Support labels

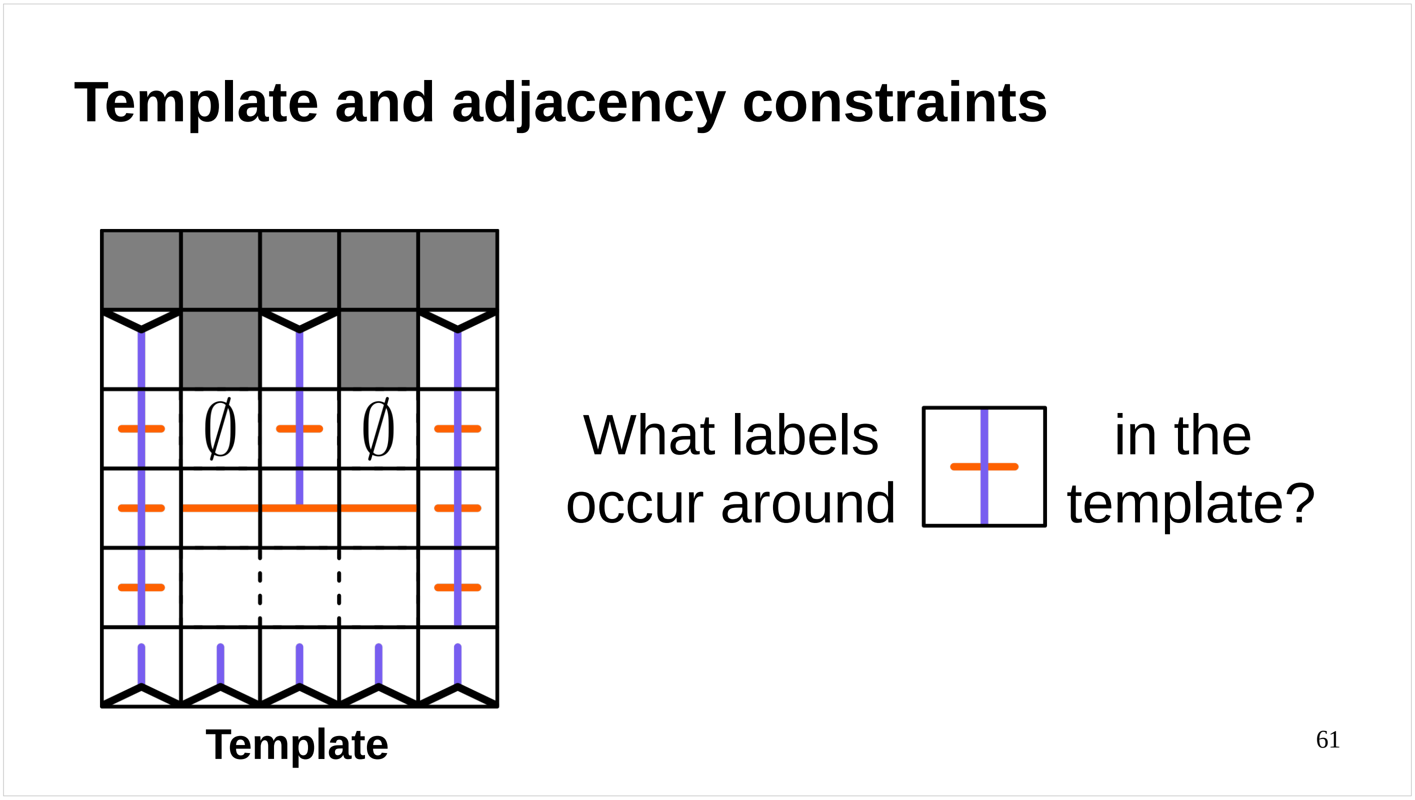
Input labels

Generated labels

I will explain the process with a simplified 2D version of the supports.

We start with a voxelized input of the object we want to print. Input voxels can be part of the object, an anchor (i.e. a point needing support), part of the ground or just empty. [CLICK] A voxel in the generated support structure can be: a part of a horizontal bar, a part of a vertical pillar, a junction between a bar and a pillar, or just empty. The empty label appears on both sides since it can be either an input, or generated by the algorithm.

The set of labels only tells us what our building blocks are, we still need to know how they can be assembled.



Template and adjacency constraints

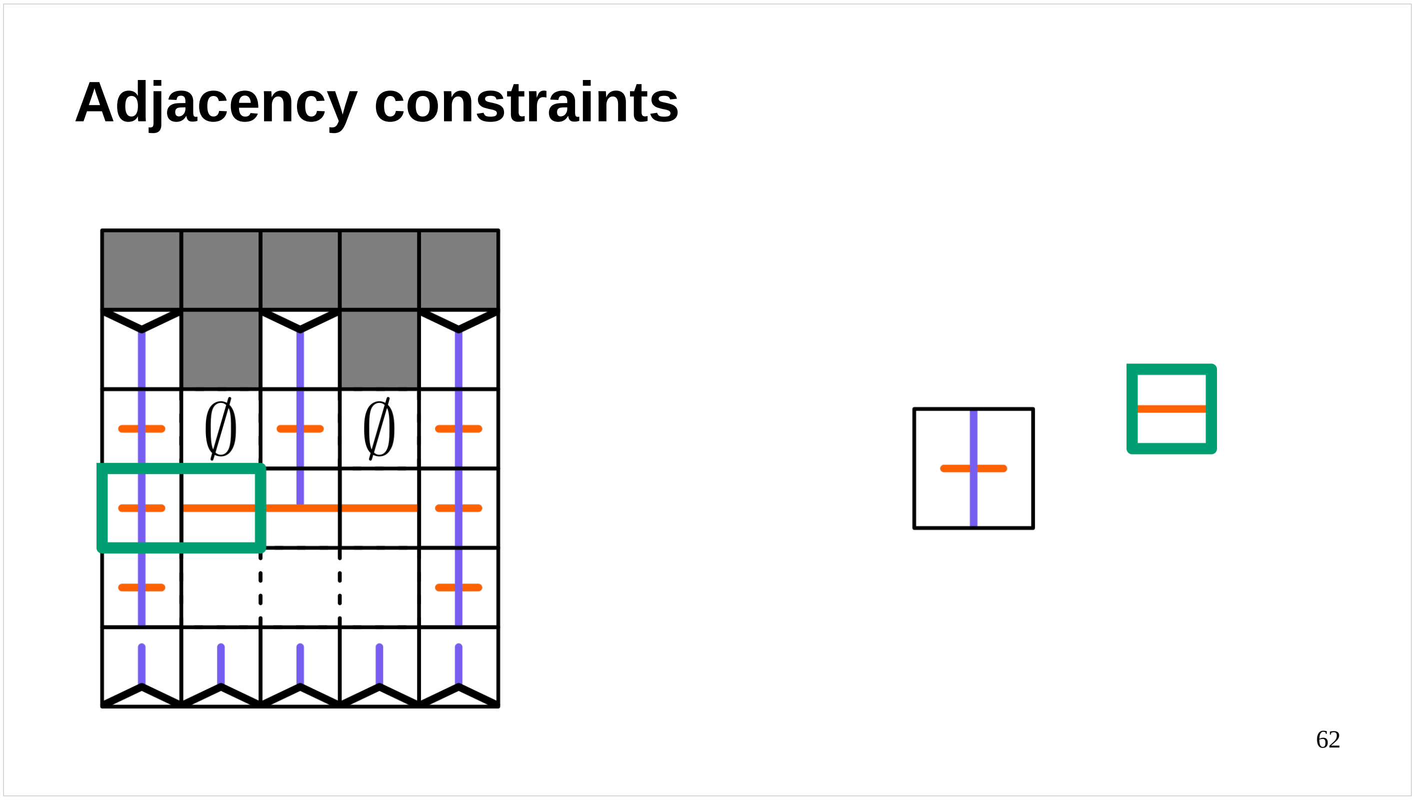
What labels occur around ┼ in the template?

Template

This information is provided by a set of adjacency constraints defined by our template. This template is a compact representation of all structures that can be synthesized by the algorithm.

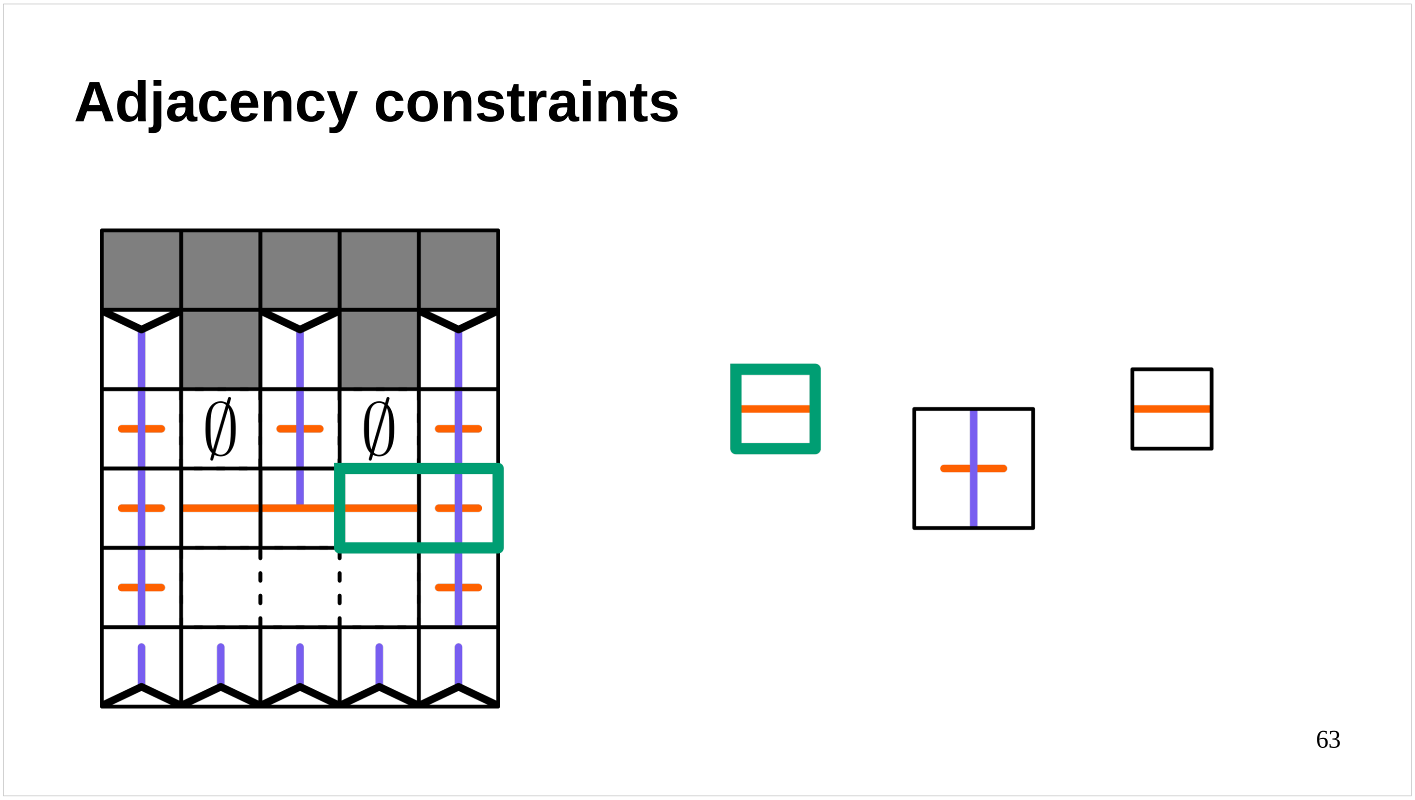
The constraints are extracted by looking at every adjacent pair of labels in every direction and marking them as allowed.

[CLICK] As an illustration, let’s find what labels are allowed next to a pillar label.



Adjacency constraints

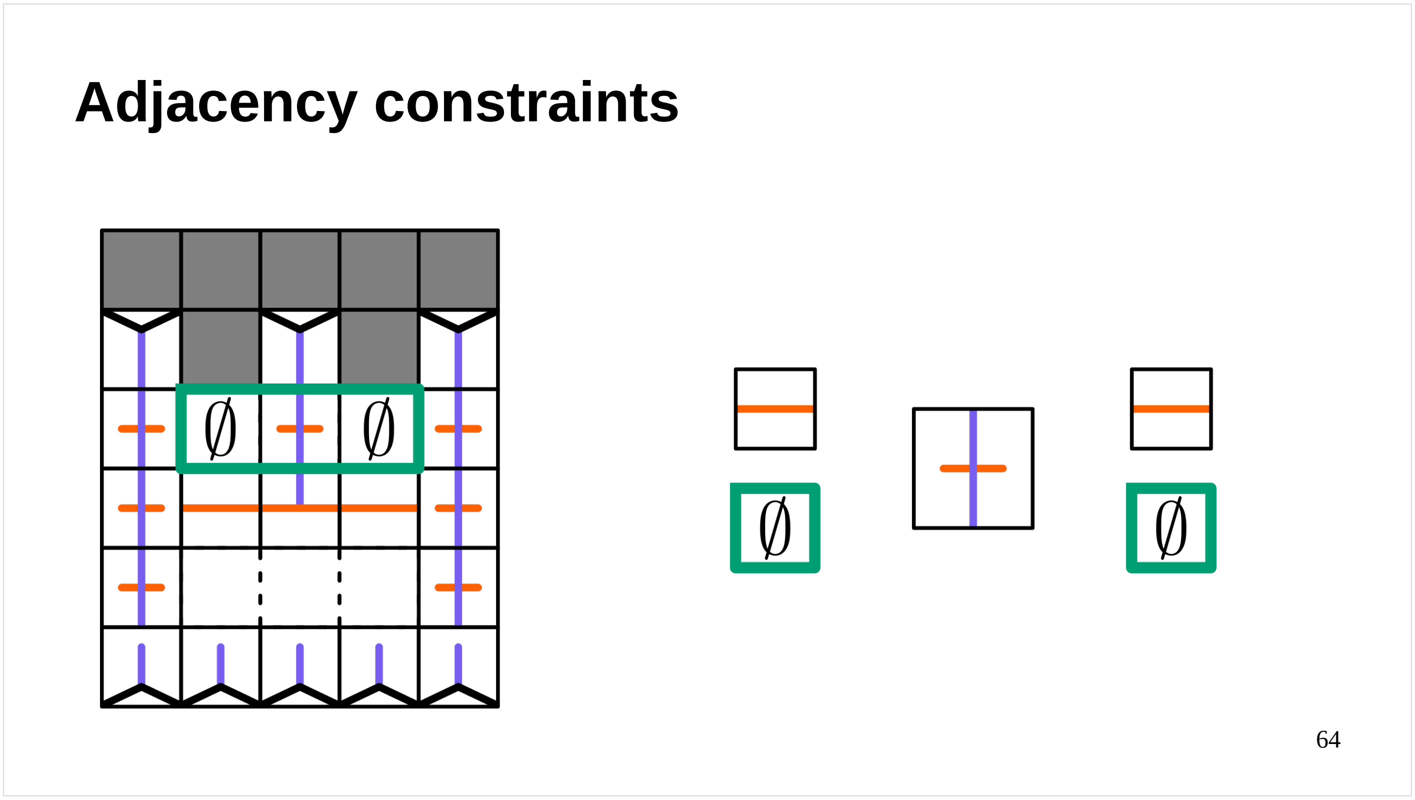
We see that a bar label is adjacent to a pillar on its right. We add this combination to the set of adjacency constraints.



Adjacency constraints

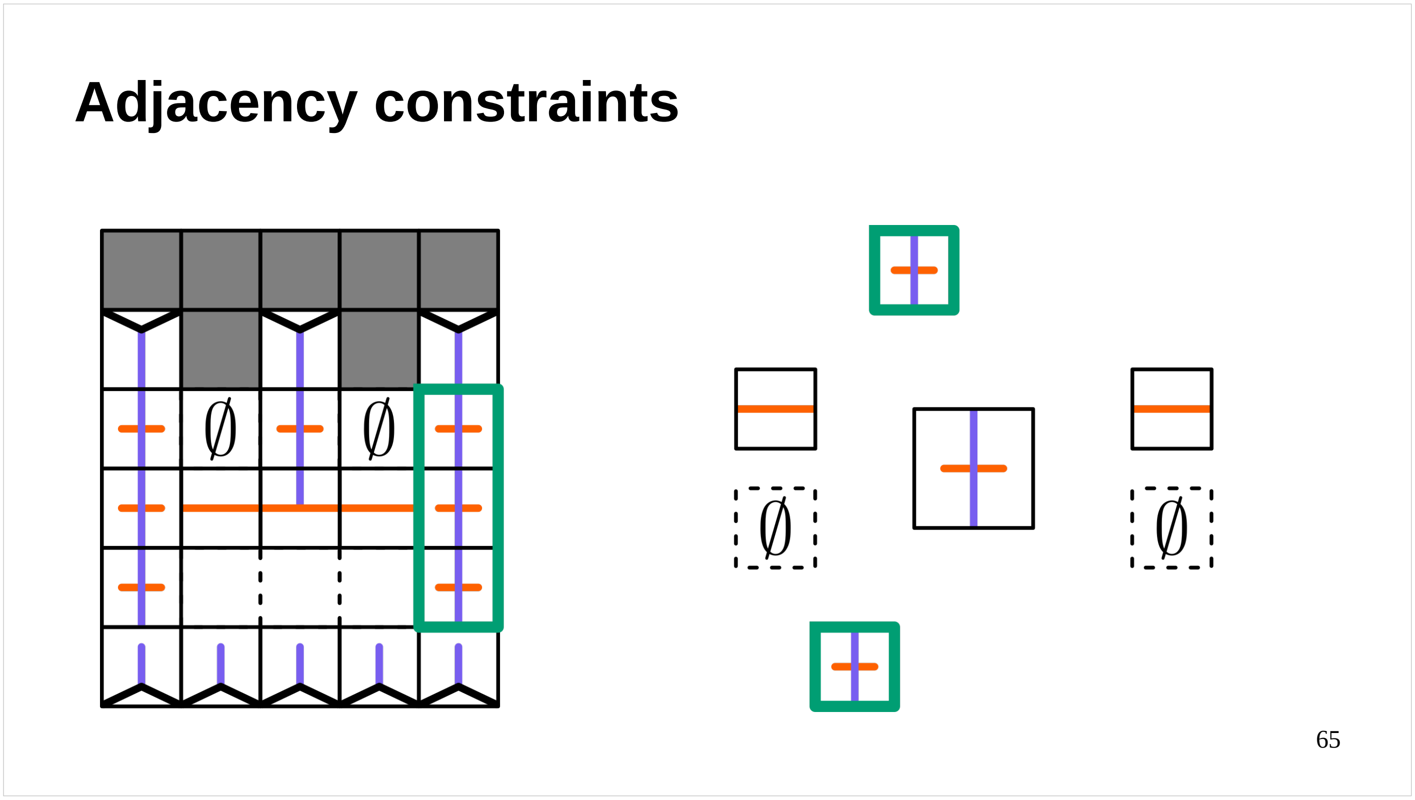
63

This is also true on the left.



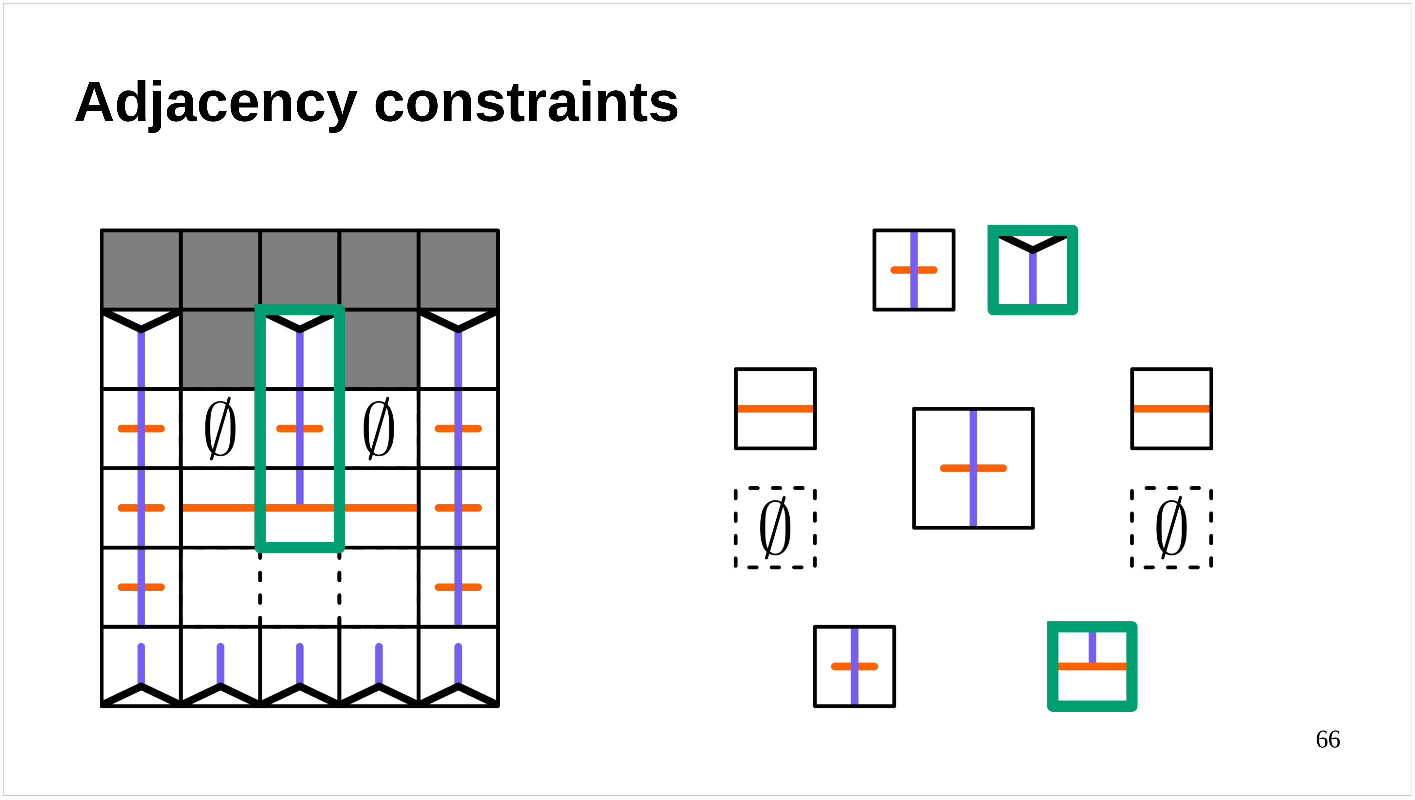
Adjacency constraints

A pillar can also be adjacent to an empty label on both left and right.



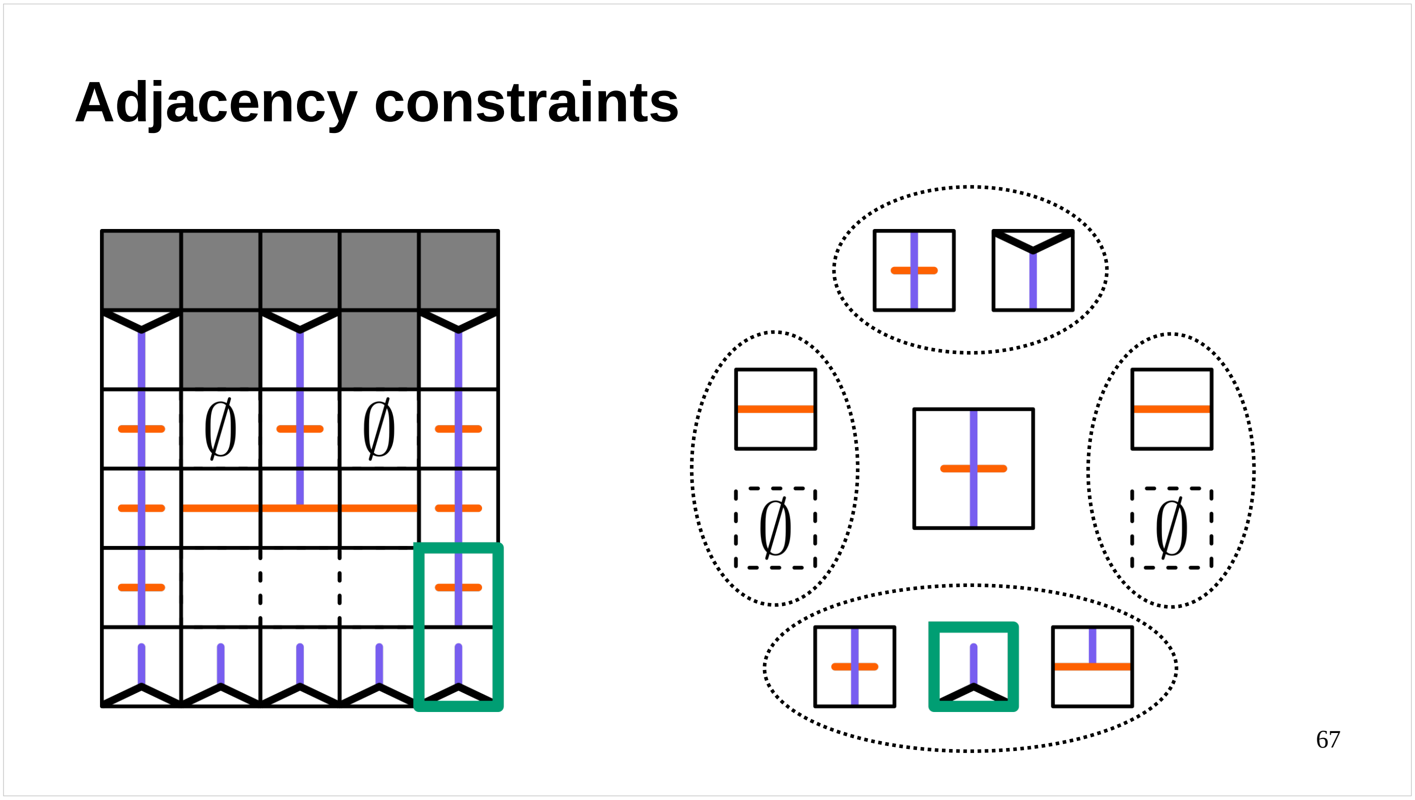
Adjacency constraints

Pillars can be adjacent to other pillars vertically.



Adjacency constraints

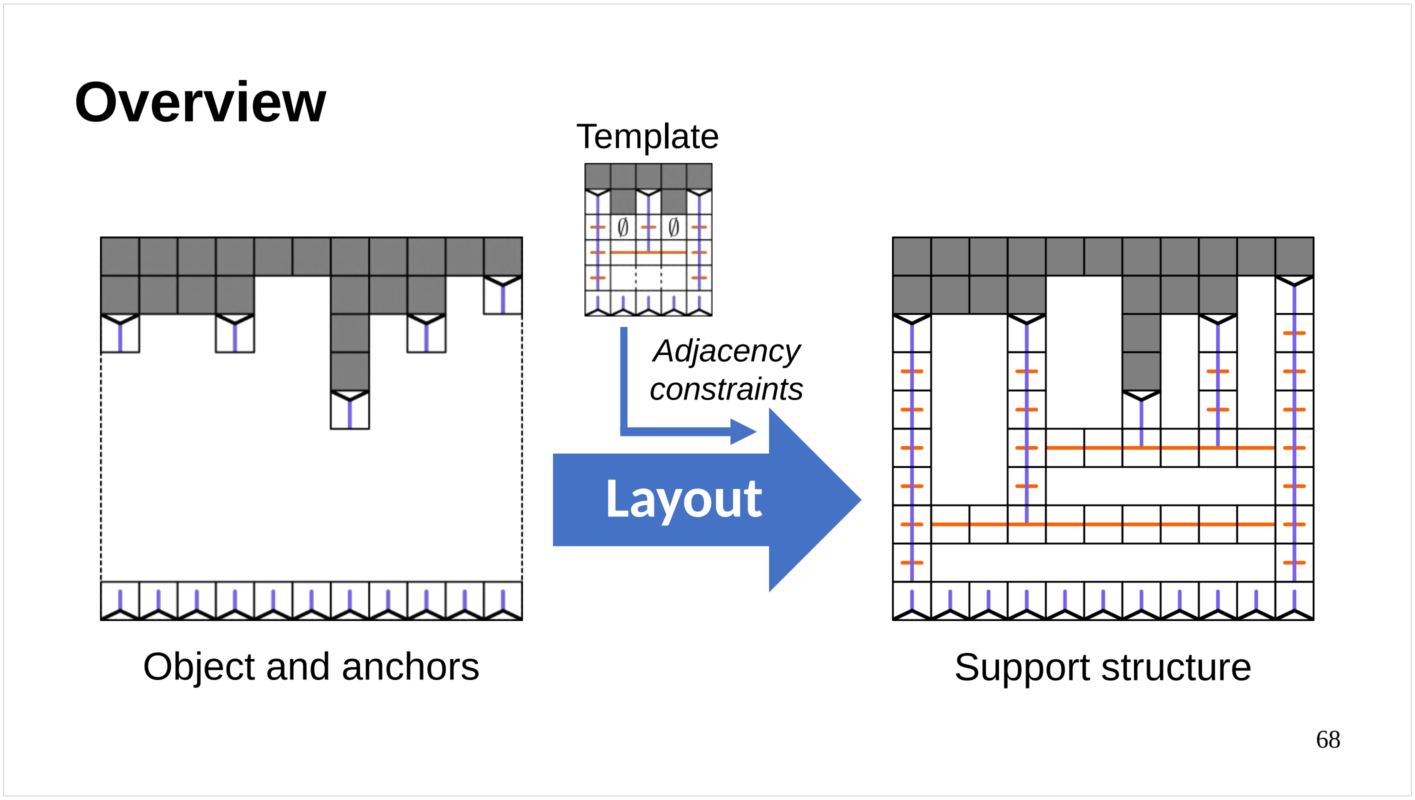
They can be right below anchors and right above junctions.



Adjacency constraints

And finally, pillars can connect to the ground.

[CLICK] This gives us every label a pillar can be adjacent to in every direction. Constraints for other labels are extracted the same way.



Overview

Template

Adjacency constraints

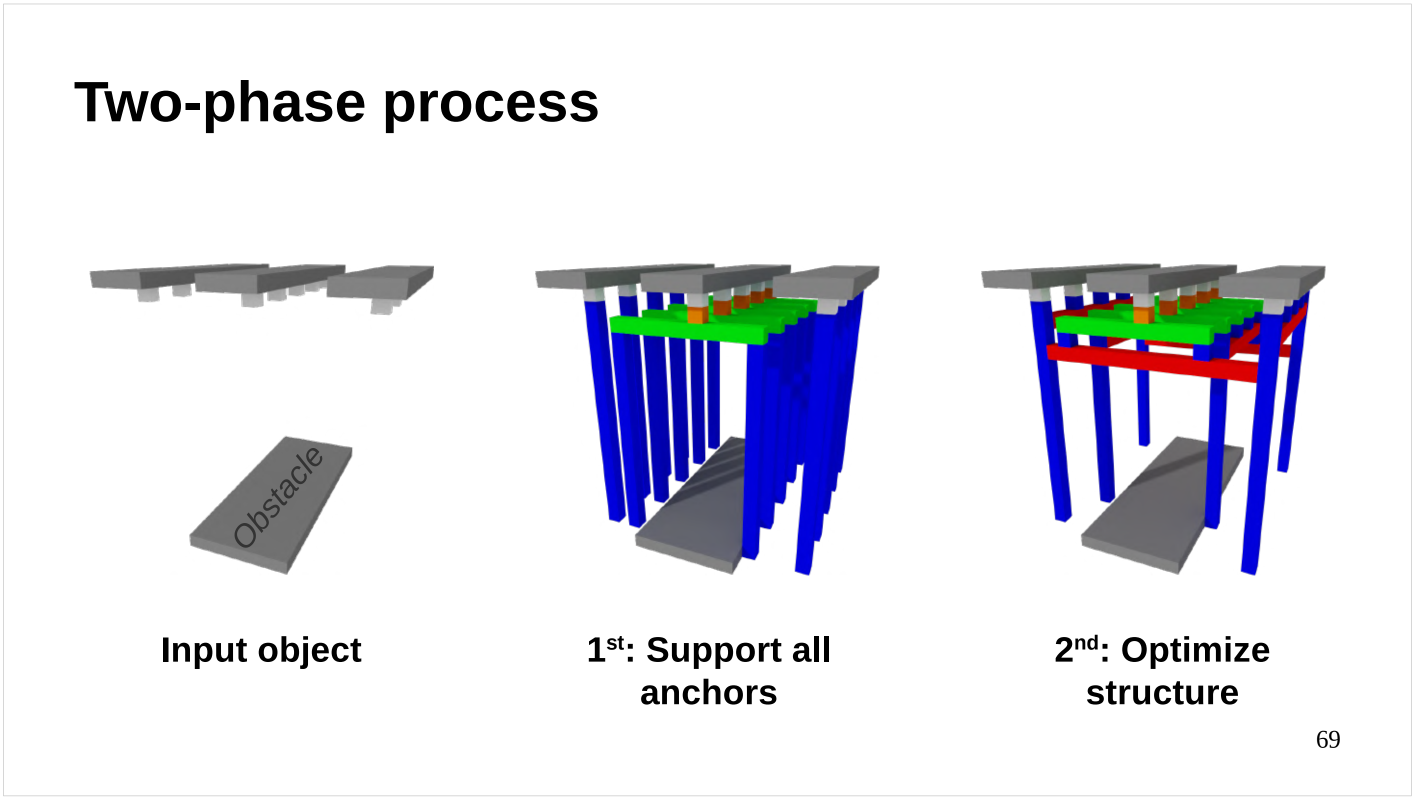
Layout

Object and anchors

Support structure

The algorithm starts with a voxelized input object mainly consisting of object and anchor labels. The rest of voxels are either empty or ground. Computing the anchors is a relevant but orthogonal problem that we do not tackle here.

We then use [CLICK] the template and the adjacency constraints to synthesize [CLICK] a support structure that properly supports the object while avoiding collisions with it.



Two-phase process

Input object

1st: Support all anchors

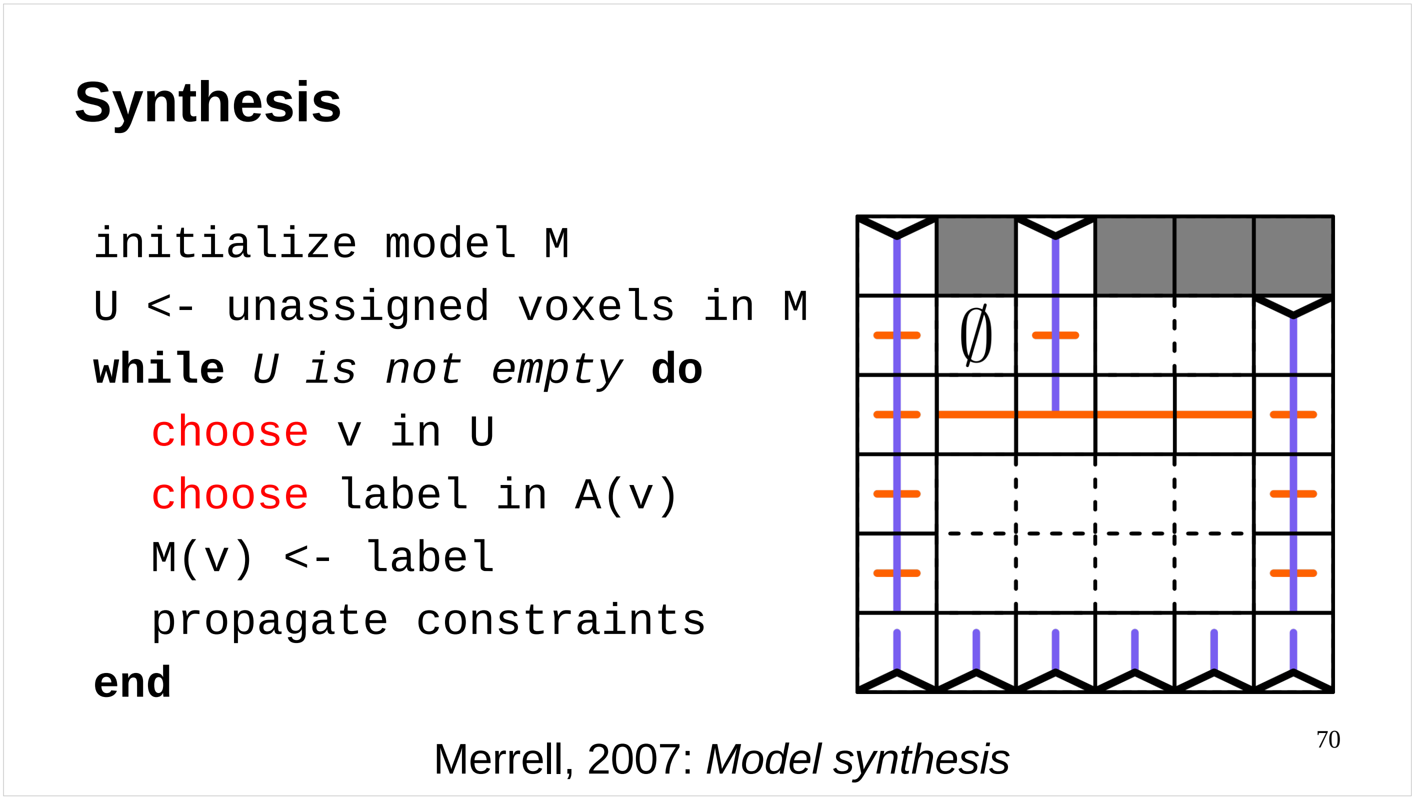
2nd: Optimize structure

69

There are two separate synthesis phases in our method. First, [CLICK] we take the input model and synthesize a solution supporting every anchor with a restricted set of simple structures.

As you can see, we generate supports that avoid the obstacle for the middle slab (green), while it generates simple vertical pillars for the outer slabs instead. This first solution is valid but inefficient, and can be improved.

This is the role of the second step, which starts from the previous solution and optimizes it by allowing the creation of bridges (red) between pillars. This makes the structure more robust and material-efficient.



Synthesis

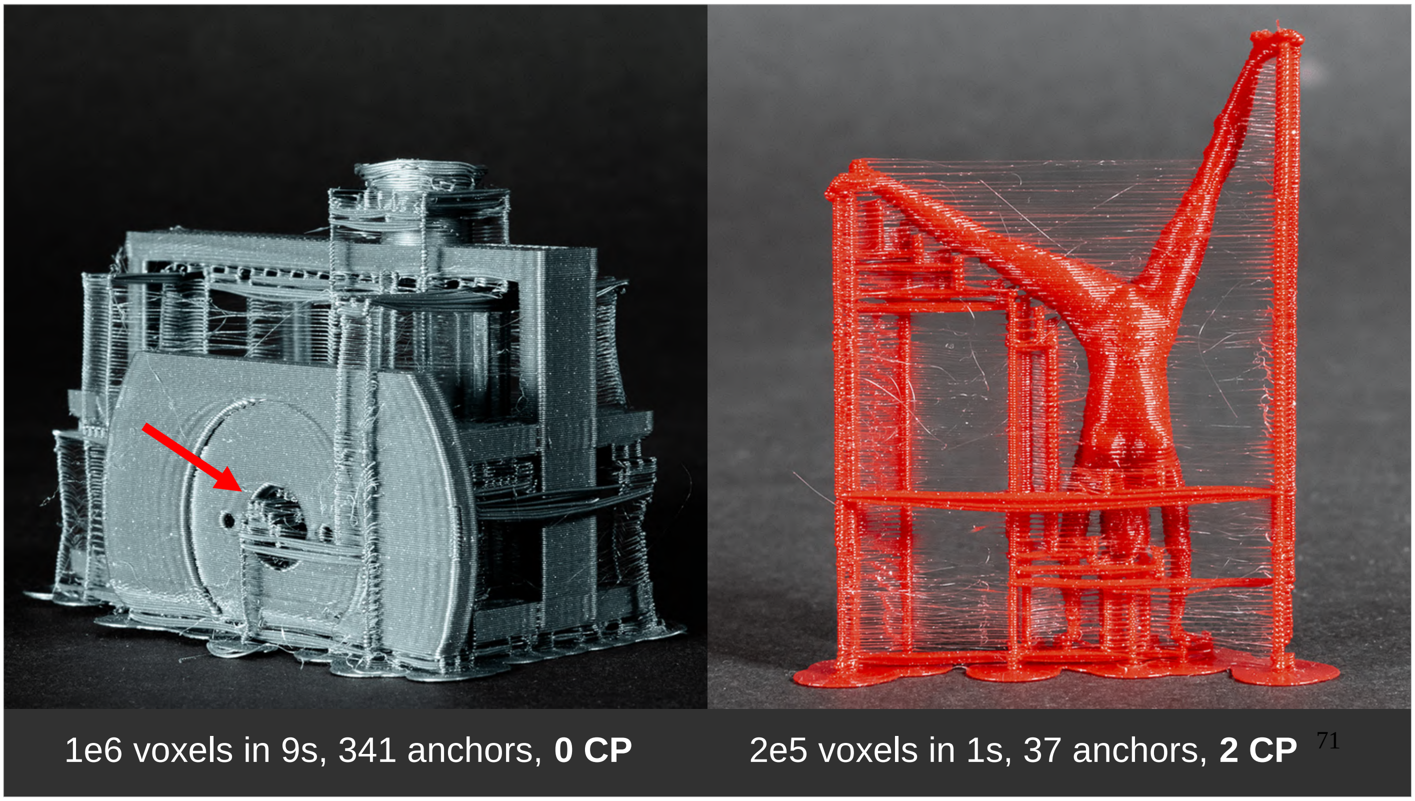
```
initialize model M
U <- unassigned voxels in M
while U is not empty do
   choose v in U
   choose label in A(v)
   M(v) <- label
   propagate constraints
end
```

Merrell, 2007: *Model synthesis*

I will not go too deeply on the synthesis algorithm itself, since it is pretty similar in concept to Model Synthesis.

The important elements are highlighted in red. We designed specific heuristics to avoid the contradictions typical of Model Synthesis.

This is all explained in detail in my dissertation.



1e6 voxels in 9s, 341 anchors, **0 CP**

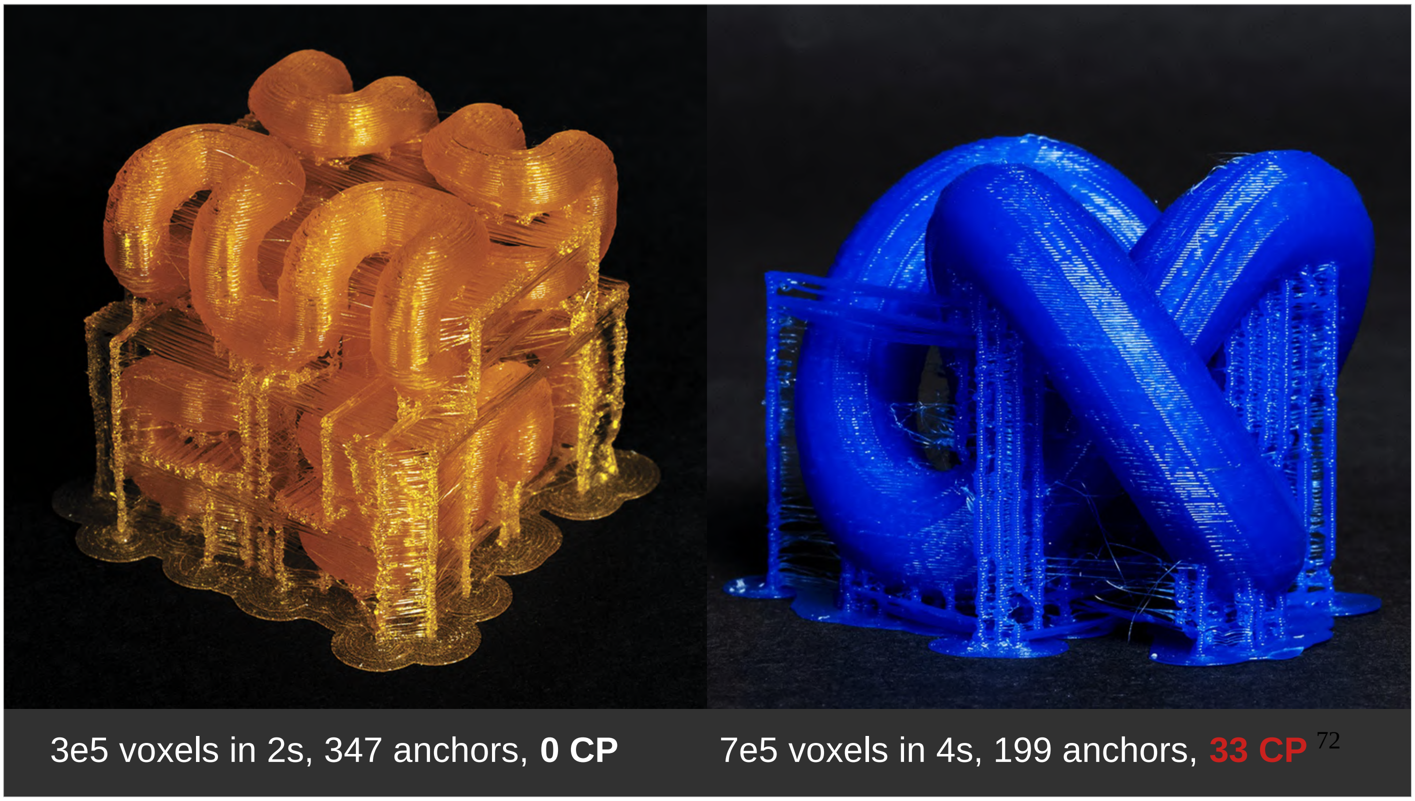
2e5 voxels in 1s, 37 anchors, **2 CP**

Here are some printed results w/ supports. The stringing is a result of slicer optimizations, and does not touch the object itself, only the supports.

All objects here have a few 1e5 to a few 1e6 voxels, and supports are always generated within 1 to 10 seconds.

Our algorithm occasionally generates new CPs between the supports and the object (denoted CPs), where a pillar stands on the object instead of the ground. These are not intersections, and are easy to remove when printed. As you can see, very few new CPs are created.

[CLICK] Also note how on the left, the supports go through the small hole to support the arch without intersecting the object.

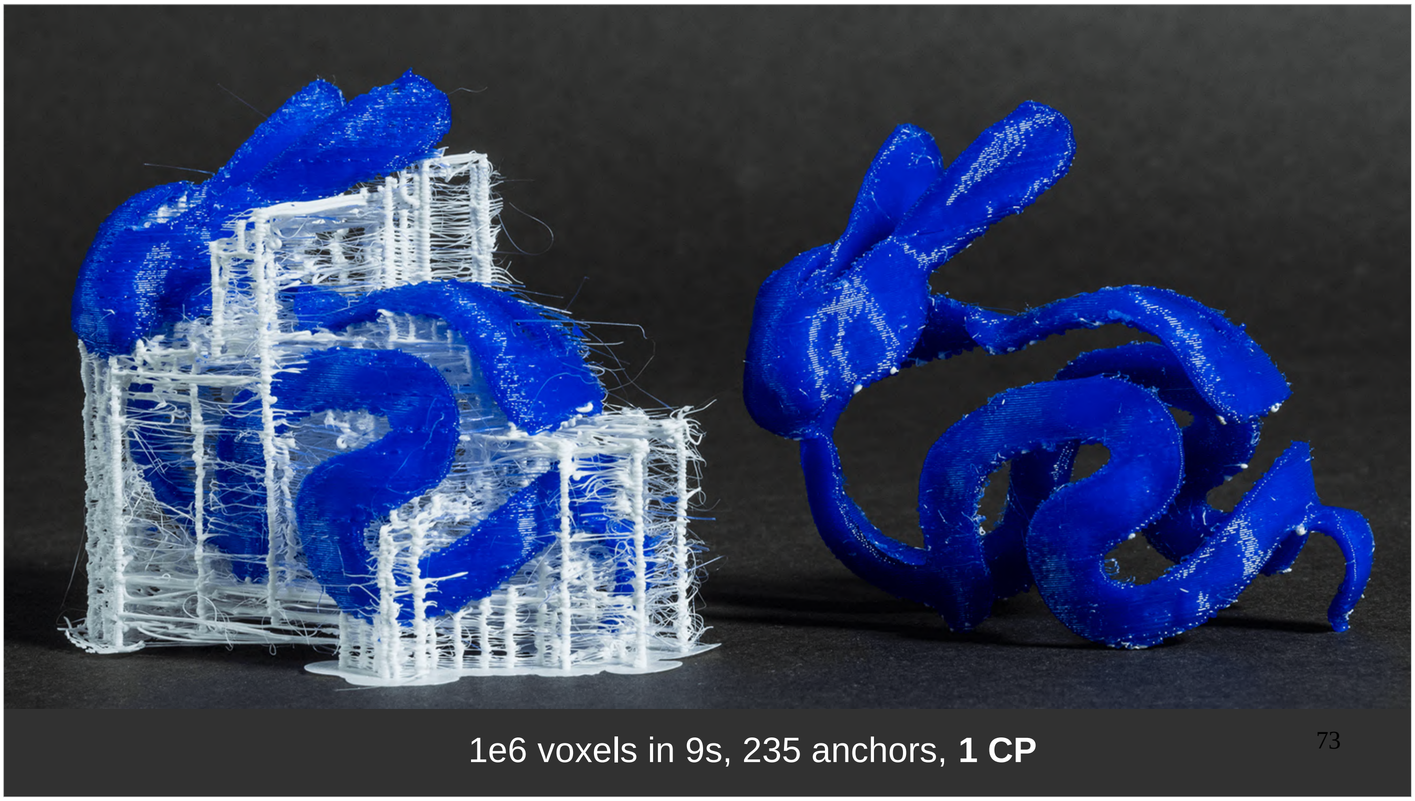


3e5 voxels in 2s, 347 anchors, **0 CP**

7e5 voxels in 4s, 199 anchors, **33 CP**

These examples showcase how our algorithm performs in tight geometries, where there is little to no room for supports.

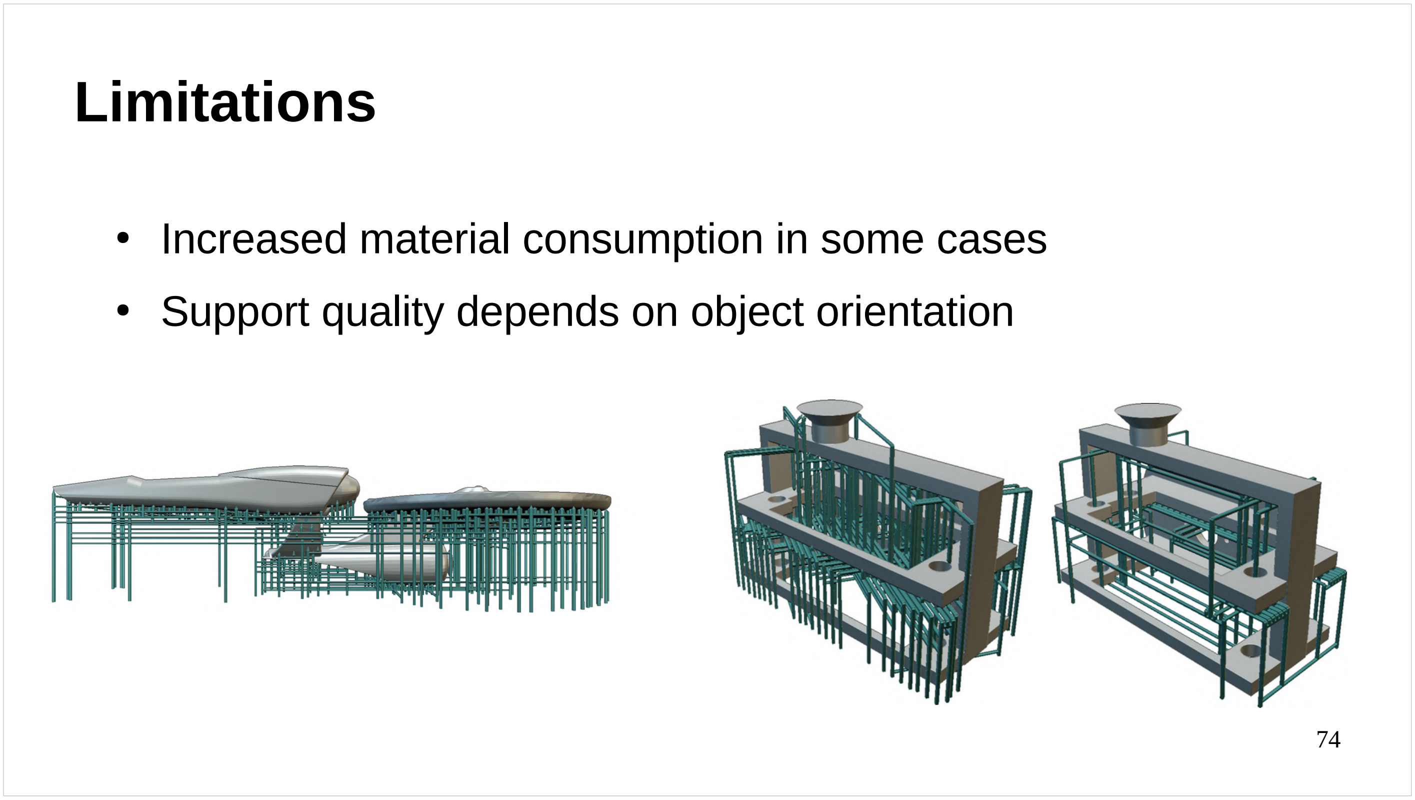
The knot (right) is significant, since the algorithm created 33 new CPs, which is a lot relative to the number of anchors. This is due to the three-fold rotational symmetry of the shape, which clashes with our axis-aligned generation method. Still, these CPs are easy to remove, leaving little to no traces.



1e6 voxels in 9s, 235 anchors, **1 CP**

Finally, for the bunny peel model from MeshMixer, we printed the supports in a different color. We see after removal that the area of contact between the object and the supports is pretty small.

We also tested our algorithm on a thousand objects from Thingiverse, and the algorithm never encountered any contradiction.



Limitations

- Increased material consumption in some cases
- Support quality depends on object orientation

Our method has some limitations. Compared to the previous algorithm, we generate more redundant structures and thus may consume more material on average (hard to evaluate). This is especially the case with long horizontal objects such as the one on the left.

(This redundancy allows us to print thinner supports, making it hard to compare ourselves to the previous method in terms of material efficiency.)

Also, support quality depends on object orientation, since we work on a 3D voxel grid. This is rarely a problem, since the best orientation is usually easy to find. For the object on the right, it is clear that the supports will be better if the object is axis-aligned.

Perspectives

[~40min]

To conclude, let me spend some time talking about avenues for future research in the topics I have discussed.

Future work

- Support more diverse inputs
- Rendering on generic surfaces
- New possibilities for electronics design

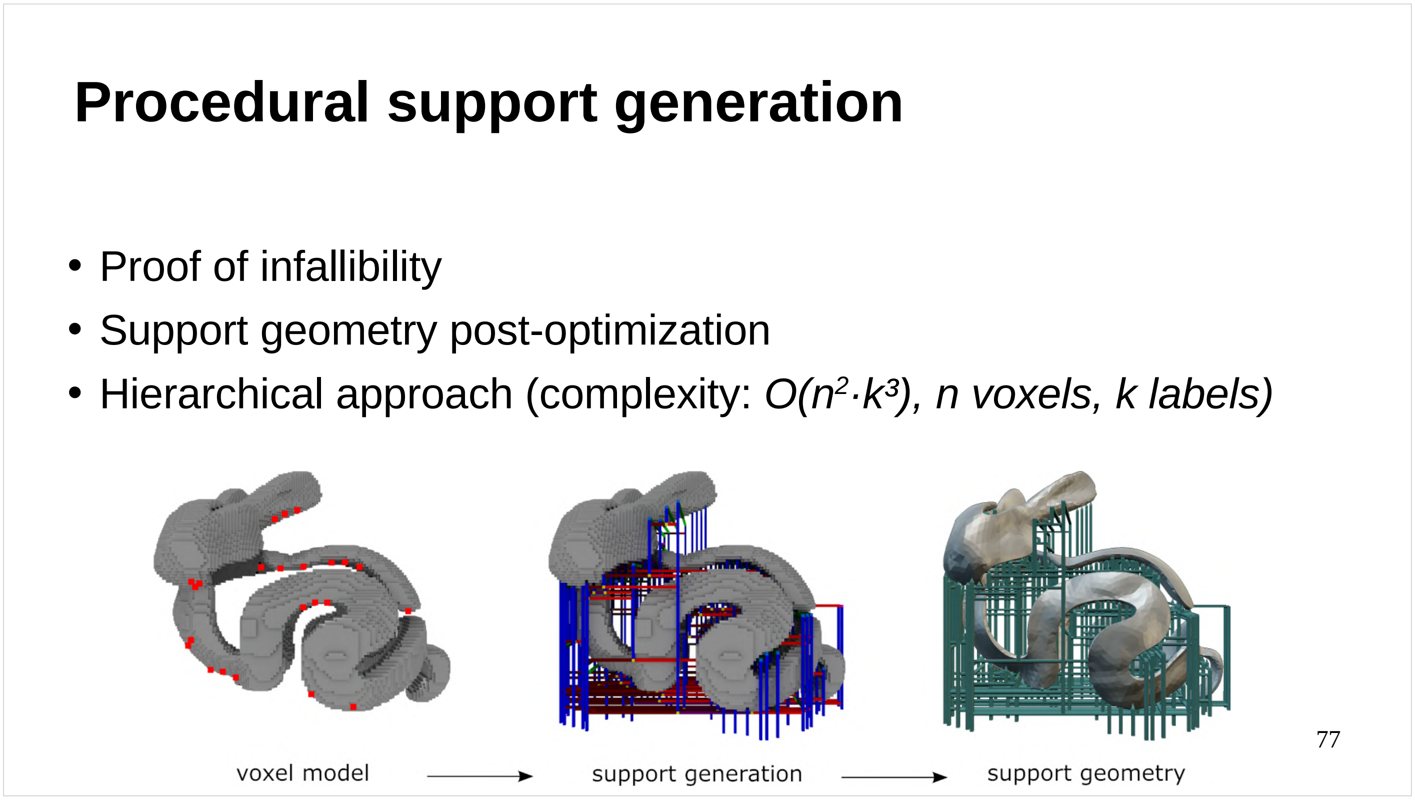
Project page

Let’s start with PCBend.

I said that the displays could have an arbitrary shape, but right now, the input mesh provided to the pipeline has to be “nice enough” (large enough triangles to fit components in and edges long enough to bend). Designing a targeted remeshing algorithm to fix problematic inputs, or expanding the range of meshes accepted by our system would allow using it for a wider variety of applications.

Rendering something on an arbitrary display also poses a challenge, since rendering algorithms are not designed for this. Right now, lighting effects use the position and neighborhood information from LEDs as an input, but a same lighting effect shows very differently in different shapes.

Finally, we only worked with LEDs in this project, but foldable circuit boards are a generic framework. Being able to design low-cost electronic circuits with a complex shape using existing fabrication methods opens new doors in electronics design in my opinion.



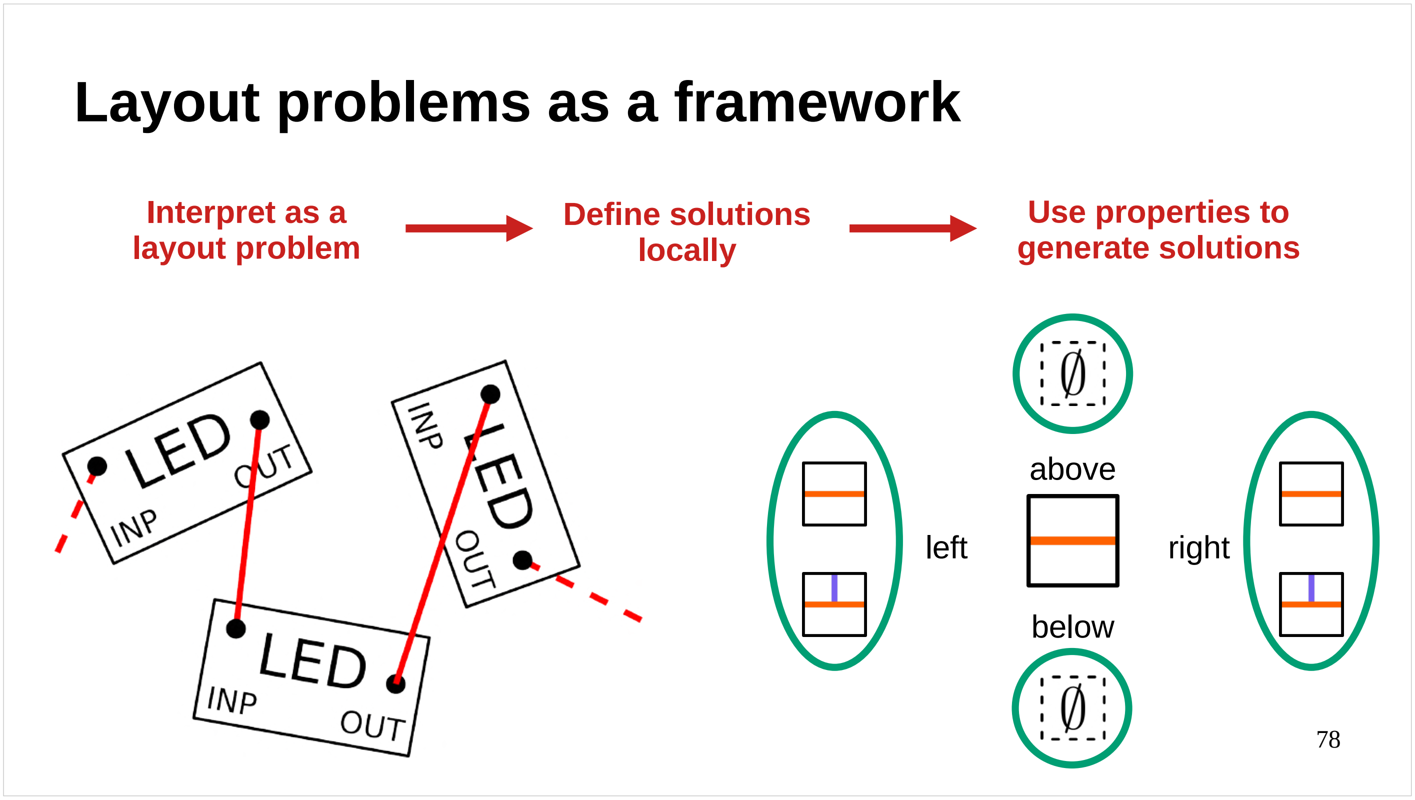
Procedural support generation

- Proof of infallibility
- Support geometry post-optimization
- Hierarchical approach (complexity: *$O(n^2 \cdot k^3)$, n voxels, k labels)*

For our procedural supports, a logical extension would be to formally prove that indeed the synthesis process always succeeds, which we have only verified empirically. I hope that this leads to a better understanding of this intricate system, allowing it to be generalized to other applications.

Another detail I have glossed over is that the algorithm actually just generates a list of horizontal and vertical segments, defined by their voxel endpoints. Then, an extra step converts the list of segments into actual support geometry. There is a lot of potential in optimizing the support structure before generating the geometry: by merging redundant structures, converting horizontal bridges into diagonal beams, and many more.

Finally, our approach scales linearly with the number of voxels in a grid, which is a problem for very large objects. I see potential in adopting a hierarchical approach to generation, where subregions of the domain are generated independently, while making sure that adjacent subdomains generate coherent structures.



Layout problems as a framework

Interpret as a layout problem → Define solutions locally → Use properties to generate solutions

To conclude, in my opinion, the main contribution of this thesis is showing that adopting the layout framework can help breaking down a problem and solve it efficiently.

Analyzing structures to derive local properties can be a great asset when confronted to a challenging situation. Additionally, due to the ubiquity of layout problems across many different domains, a huge variety of methods have already been successfully implemented for solving them. Drawing on this existing state of the art can be very useful when tackling new problems.

The main challenge is that there is currently little crossover between different fields dealing with layout problems: industrial facility layouts, architectural building layouts, electronic circuit layouts papers barely reference each other. They deal with similar constraints that feel different due to them originating from different causes. I feel there is unexploited potential that could be tapped into by drawing bridges between these different fields.

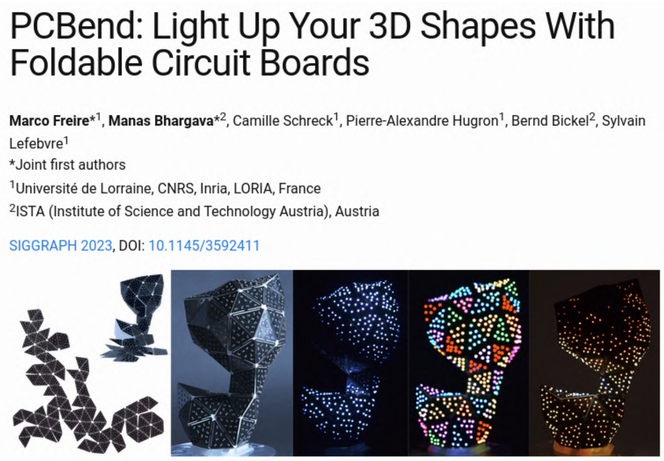
Fabrication is messy :)

Finally, I would like to share some insights about research in fabrication. The main thing that I have learned in the last 4 years is that fabrication is very messy. Actually producing functioning objects with finicky machines takes a lot of tries. You constantly need to take the fabrication process and constraints in consideration.

Article deadlines become extra terrifying when you take into account unexpected shipping delays, last minute electronic failures, or 3D printer malfunctions.

There are many failed and successful design iterations for both projects that I was not able to discuss during this defense, and I simply wanted to shine a light on them by including them here!

Thank you for your attention!



PCBend: Light Up Your 3D Shapes With Foldable Circuit Boards

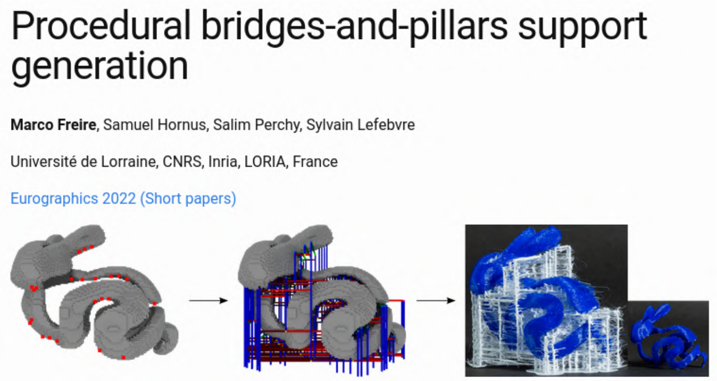
Marco Freire*[1], **Manas Bhargava***[2], Camille Schreck[1], Pierre-Alexandre Hugron[1], Bernd Bickel[2], Sylvain Lefebvre[1]

*Joint first authors

[1]Université de Lorraine, CNRS, Inria, LORIA, France

[2]ISTA (Institute of Science and Technology Austria), Austria

SIGGRAPH 2023, DOI: 10.1145/3592411



Procedural bridges-and-pillars support generation

Marco Freire, Samuel Hornus, Salim Perchy, Sylvain Lefebvre

Université de Lorraine, CNRS, Inria, LORIA, France

Eurographics 2022 (Short papers)

Questions?

This concludes my presentation. Thank you all for your attention. I am happy to answer your questions now.