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

Marco Freire, Manas Bhargava, Camille Schreck, Pierre-Alexandre Hugron, Bernd Bickel, Sylvain Lefebvre

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

MARCO FREIRE*, Université de Lorraine, CNRS, Inria, LORIA, France
MANAS BHARGAVA*, ISTA (Institute of Science and Technology Austria), Austria
CAMILLE SCHRECK, Université de Lorraine, CNRS, Inria, LORIA, France
PIERRE-ALEXANDRE HUGRON, Université de Lorraine, CNRS, Inria, LORIA, France
BERND BICKEL, ISTA (Institute of Science and Technology Austria), Austria
SYLVAIN LEFEBVRE, Université de Lorraine, CNRS, Inria, LORIA, France

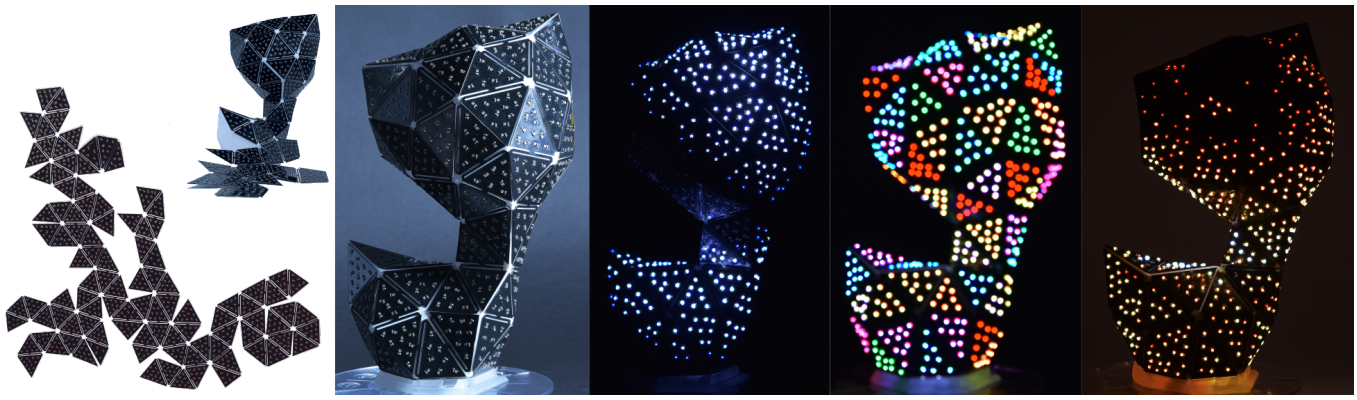


Fig. 1. Starting from a 3D mesh our method automatically generates design files to produce an on-surface display composed of individually addressable RGB LEDs. The circuit board is manufactured through standard PCB production services, including component soldering. The user then folds the fabricated 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.

We propose a computational design approach for covering a surface with individually addressable RGB LEDs, 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 sub-problem 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.

The code and data for this paper are available at https://github.com/mfremer/pcbend.

CCS Concepts: • **Hardware** → *PCB design and layout*; • **Computing methodologies** → **Mesh geometry models**.

Additional Key Words and Phrases: PCB design, PCB bending, automated placement and routing, 3D electronics

ACM Reference Format:
Marco Freire, Manas Bhargava, Camille Schreck, Pierre-Alexandre Hugron, Bernd Bickel, and Sylvain Lefebvre. 2023. PCBend: Light Up Your 3D Shapes With Foldable Circuit Boards. *ACM Trans. Graph.* 42, 4, Article 142 (August 2023), 16 pages. https://doi.org/10.1145/3592411

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

*Joint first authors.

Authors' addresses: Marco Freire, marco.freire@inria.fr, Université de Lorraine, CNRS, Inria, LORIA, France; Manas Bhargava, manas.bhargava@ist.ac.at, ISTA (Institute of Science and Technology Austria), Austria; Camille Schreck, camille.schreck@inria.fr, Université de Lorraine, CNRS, Inria, LORIA, France; Pierre-Alexandre Hugron, pierre-alexandre.hugron@inria.fr, Université de Lorraine, CNRS, Inria, LORIA, France; Bernd Bickel, bernd.bickel@ist.ac.at, ISTA (Institute of Science and Technology Austria), Austria; Sylvain Lefebvre, sylvain.lefebvre@inria.fr, Université de Lorraine, CNRS, Inria, LORIA, France.

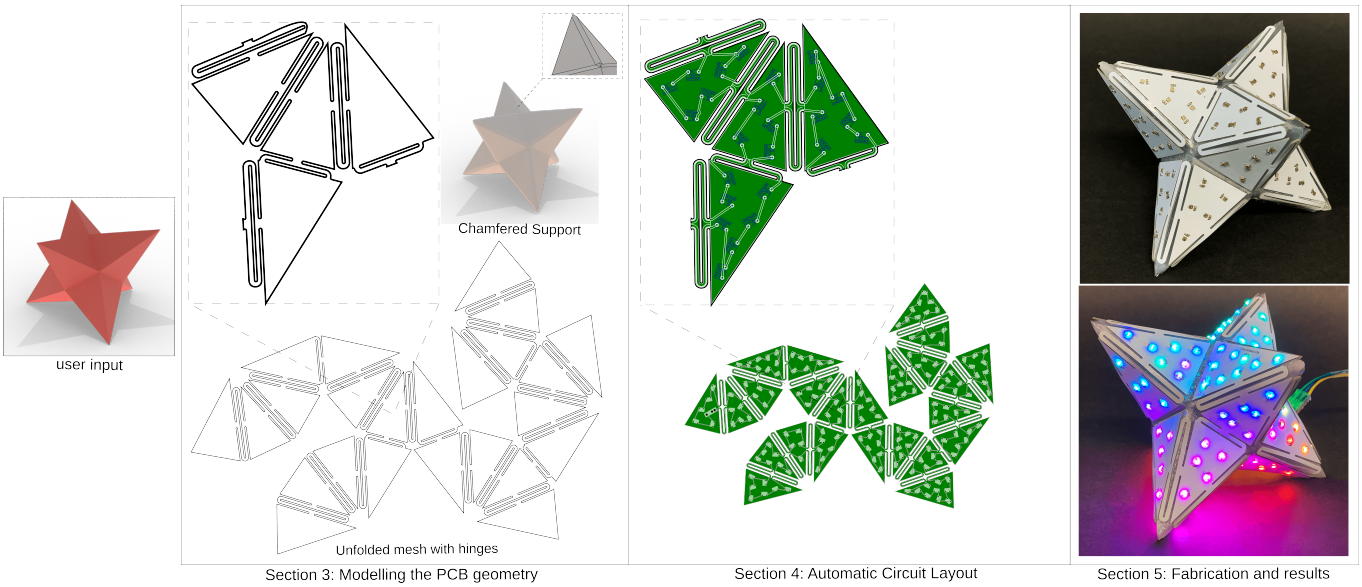


Fig. 2. From the user input mesh, our method starts by modeling the PCB geometry, making it foldable with hinge patterns (Section 3). LEDs are then placed, the circuit optimized and routed to obtain fabrication-ready PCB blueprints (Section 4). A chamfered support mesh is also generated. The PCB is then folded onto the 3D printed support. The assembled model is finally ready to display visual patterns (Section 5).

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. While promising methods are being explored to prototype circuits along curved surfaces (see Section 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. Please refer to Section 3.4 for details.

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 2.

Fabrication. To ensure fabrication *scalability* and *availability*, we target traditional (non-flexible) printed circuit boards (PCBs). 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 PCB — 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 1 and 2. To this end, we design specialized *kerfing* patterns [Kalama et al. 2020; Lee et al. 2018] that we call *hinges* (Section 3.1). These allow the PCB to bend along specific cut patterns, while enabling required electric signals to go through. We seek to

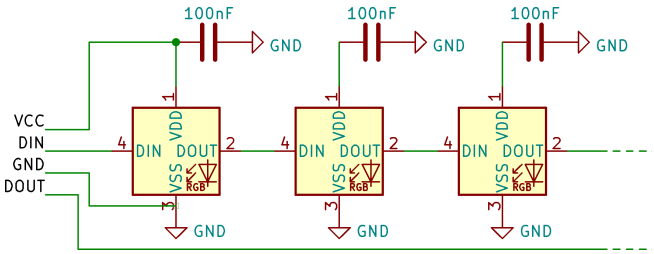


Fig. 3. Schematics of a chain of LED pixels, Each LED is connected to gnd and vcc, with a decoupling capacitor next to it. LEDs are chained through their `data-in`, `data-out` pins, passing information along the chain. The connector `DOUT` pin can be used to connect multiple chains together.

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 (Section 3.2).

This setup naturally leads to an unfolding problem (Section 3.3): 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. The unfolder gives us the PCB geometry. We next densely cover the board with LEDs connected by a valid circuit satisfying all manufacturing constraints (Section 4).

We rely on RGB LEDs that can be 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. 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 across the entire design. This path is not abstract: it has to go around components and has to satisfy geometric constraints from PCB manufacturing (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.

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.

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. We discuss requirements on the input mesh in Section 3.4.

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.

2 RELATED WORKS

Electronics design typically starts from a circuit schematic after which a printed circuit board (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 [Lienig and Scheible 2020]. *Vias* are inserted to connect traces across layers.

PCB manufacturing is a mature technology, ubiquitously used across many industries. Components range from millimetric surface-mounted devices (SMD) 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. This is a task under strict geometric and electrical requirements, and professional tools [Autodesk 1988; Charras 1992] 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. [Rich et al. 2021; Wu et al. 2020].

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 [MID 1992; Toriz-Garcia et al. 2013; Umetani and Schmidt 2017] and additive manufacturing techniques [Flowers et al. 2017; He et al. 2022; Swaminathan et al. 2019; Zhu et al. 2020a]. A variety of materials are used such as carbon, graphene or copper enriched filaments [Flowers et al. 2017] and silicons [Zhu et al. 2020a].

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 [Hong et al. 2021], 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 [Angel et al. 2018; Kim et al. 2019].

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 [Hong et al. 2021; Plovie et al. 2017b], self-morphing under heat [Wang et al. 2020], wearable textile substrates [Paredes et al. 2021] or manual folding [Olberding et al. 2015]. Other techniques produce planar transfer stamps [Groeger and Steimle 2018; Hodges et al. 2014; Zhu et al. 2020b] to help a user place conductive traces on a target surface. Rigid-flexible PCBs can also be used to produce curved designs [Altium 2014]. For instance, Muscolo et al. [2019] connects ten triangular circuit elements through flexible areas to produce a curved sensor. Such PCBs are however significantly more expensive than regular PCBs [Bob Burns 2020; Wang et al. 2020] 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 [Yamaoka et al. 2019] with traces connected across [Olberding et al. 2015]. Circuits may contain several conductive layers with connections across isolating layers [Hong et al. 2021; Yan et al. 2022].

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 [Jason et al. 2015; Kawahara et al. 2013; Russo et al. 2011; Wang et al. 2018] or laser cutting of specially prepared laminated copper–kapton sheets [Yan et al. 2022]. This led to the development of interactive tools assisting users in creating foldable designs with sensors, actuators and display elements [Oh et al. 2018; Olberding et al. 2015; Paredes et al. 2021; Qi and Buechley 2010]. Curvature can also be facilitated by introducing spatially varying kerfing patterns [Groeger and Steimle 2019].

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 – in terms of manual steps – to circuits with hundreds of components.

Curved displays. Among the aforementioned approaches, a number of techniques prototype displays as applications.

The approach of Torres et al. [2017] 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. [2014], Lee et al. [2020] and Hanton et al. [2020] 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 [Olberding et al. 2014; Plovie et al. 2017a,b] to a grid of 25×16 standard LEDs manually glued on paper [Russo et al. 2011]. 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.

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 [Polthier 2009]. Origami and kirigami, traditional Japanese art forms of folding and cutting paper, have been extensively studied [Callens and Zadpoor 2018] and are strongly related to unfolding.

Obtaining a non-overlapping unfolding might require cutting the surface into multiple disconnected *patches*. The question of unfolding with a minimal number of disconnected patches is computationally hard [Demaine and O'Rourke 2007; Shephard 1975]. Thus, algorithms typically rely on heuristics. Straub and Prautzsch [2011] use a minimum perimeter heuristic [Schlickenrieder 1997]. 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 [Straub and Prautzsch 2011]). This approach is however known to still create several patches on complex cases. Several improvements have been explored, using for instance genetic algorithms [Takahashi et al. 2011] or simulated annealing [Korpitsch et al. 2020].

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 PCBs but provides a strong foundation on which our algorithm is based.

2.4 Background on shape packing

Distributing components on the surface is a case of packing shapes inside a bounded domain, with additional strict constraints of non-overlapping components and routability of the circuit. A general approach of tiling shapes in a bounded 2D domain is a hard problem [El-Khechen et al. 2009; Fowler et al. 1981] leading most algorithms to resort to heuristics.

A typical approach is to use a Centroidal Voronoi Tesselation (CVT) to distribute points and spawn tiles of various shapes in a

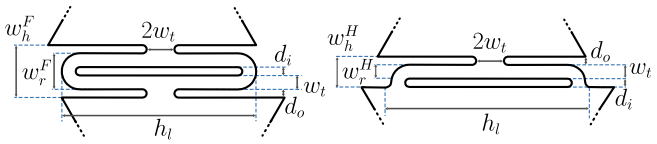


Fig. 4. Hinge designs and parameters. Left: full hinge, right: half hinge. Track width (w_t) is 1.7 mm, inner diameter (d_i) and outer diameter (d_o) are 1.0 mm each. The rigid width is w_r^F = 4.4 mm for full hinges and w_r^H = 1.7 mm for half hinges. The total hinge width is w_h^F = 6.4 mm for full hinges and w_h^H = 3.7 mm for half hinges.

bounded domain, as in for instance [Dalal et al. 2006; Hausner 2001; Smith et al. 2005]. These works however do not guarantee a non-overlapping result. Others explore the layout of tiles directly on a surface, for filigrees [Chen et al. 2016], mosaics [Hu et al. 2016], volumetric elements [Fanni et al. 2022] and fabricable tilings [Chen et al. 2017]. However, tiles are deformed to ensure that they fit compactly, which cannot be done with components. Xu et al. [2020] 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, see Section 4.3.1.

3 MODELING THE PCB GEOMETRY

Our method proceeds in two main steps: PCB geometry modeling and circuit synthesis (Figure 2). In this Section we detail the first part of the process.

In Section 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 3.2.

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 and the modeling of the support structure in Section 3.5.

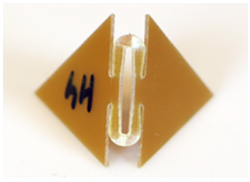
3.1 Hinge designs

We consider two types of hinges, shown in Figure 4. We refer to them as *full hinges* (Figure 4, left) and *half hinges* (Figure 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 11). Thus, both our hinge designs are parameterized by just their hinge lengths h_l.

3.2 Hinge bending

We use 0.6 mm thick PCBs 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 obtain the admissible angles, we conduct an experiment where we bend hinges of different lengths. We observe that during progressive bending a visible yellowing gradually appears on the PCB material before breakage. This yellowing effect serves as an indication of stress accumulation.

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. Figure 5 shows the resulting maximum angle plot. Given this experimental data for both full and half hinges, we can consider how to obtain the overall PCB layout.

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. [2011] 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 4.

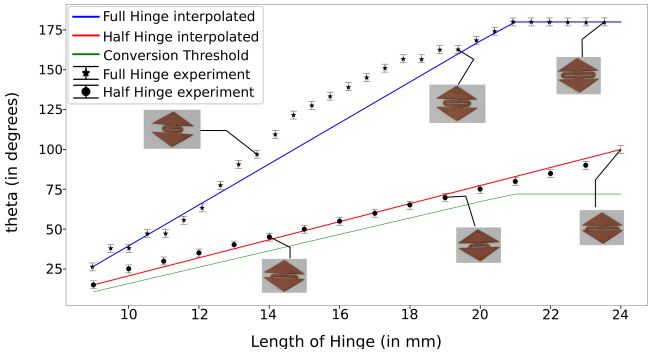


Fig. 5. Safe bending angle for varying hinge length for both full hinges and half hinges. In green, we see the safety conversion threshold below which we safely replace a full hinge with a half hinge.

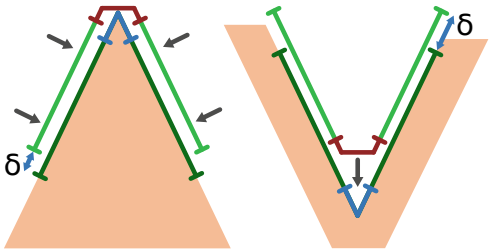


Fig. 6. Consequences of ignoring the hinge curvature as seen from the side on convex and concave edges. The assumed folding behavior (crease line) is shown in blue, the actual one in red, with the triangles attached to the hinge in green. This moves the triangles away from the opposite edge by an offset δ, introducing a global discrepancy.

In practice, the middle section bends slightly but we found this to be negligible.

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 6. 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. We overcome this problem by trimming the triangles neighboring an edge by an amount that depends on the edge dihedral angle θ and the hinge parameters. Similar to [An et al. 2018], 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 7):

$$\delta^*_\theta = d_* + d_\theta = d_* + \frac{w_r}{2\sin(\theta/2)}, \qquad * \in \{i, o\} \tag{1}$$

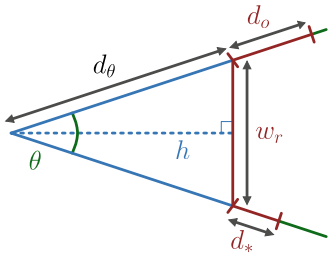


Fig. 7. Illustration for the dihedral offset computation for both hinges. $d_* = d_o$ for full hinges, $d_* = d_i$ for half hinges.

The total amount trimmed from the triangles in the unfolding is $2 \cdot \delta^o_\theta$ for a full hinge and $\delta^i_\theta + \delta^o_\theta$. 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.

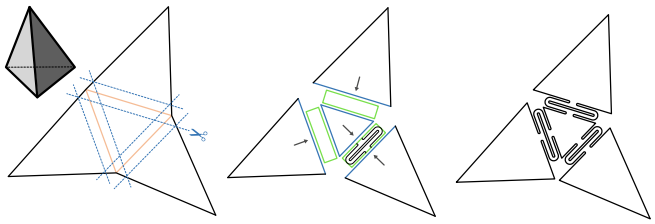


Fig. 8. Hinge insertion. On the left, triangles are trimmed according to the dihedral offsets. In the middle, hinges are inserted between adjacent triangles, which are then snapped to remove gaps. On the right, the final geometry is shown.

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 on rare occasions: 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 8. 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 paper.

Checking feasibility of hinges. After inserting the hinges in our unfolded mesh, we check for feasibility of all the hinges by using the bending data information (refer to section 3.2). For every hinge, we compute its hinge length and consult the data to check whether the hinge can bend within the safety margins to its desired bending angle. If a hinge is not feasible we report the mesh as incompatible and suggest a scale up value to be applied to the mesh.

Adding half hinges. We also use the experimental data to select the hinge type. We favor half hinges as they require less PCB area than full hinges. We choose half hinges whenever the required bending angle is below the threshold of safe bending angle for a half hinge (See Figure 5). Table 2 reports the usage of half hinges and full hinges on our test models. Naturally, a smoother model with smaller dihedral angles has fewer (zero) full hinges, whereas models with acute angles have to use full hinges.

Choice of unfolder heuristic. Since unfolding a mesh with minimal patch count is computationally hard, algorithms rely on various heuristics depending on application requirements. In this work we rely on minimum dihedral angle and minimum perimeter heuristics. The minimum dihedral angle heuristic penalizes edges that have high dihedral 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 tends to cut shorter edges where hinges would bend less and empirically results in fewer patches. The minimal dihedral angle heuristic results in a single patch unfolding for most of our examples (see also Section 5.5). We

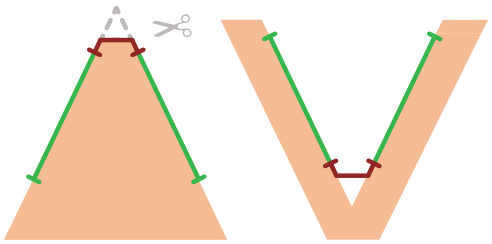


Fig. 9. Effect of hinge insertion after trimming and chamfering, side view of convex and concave edges. After folding the hinge is now flush with the support. Triangles are no longer displaced from their intended position.

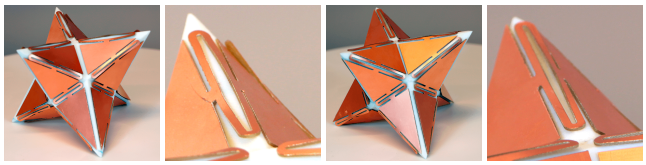


Fig. 10. Impact of chamfering on hinges. Left: No chamfering. Right: With chamfering. Although chamfering is barely noticeable on the mesh, it greatly reduces hinge deformation.

expose the choice of heuristic as a design parameter, so the user can choose the one that gives the best result for each model.

3.4 Compatible input mesh

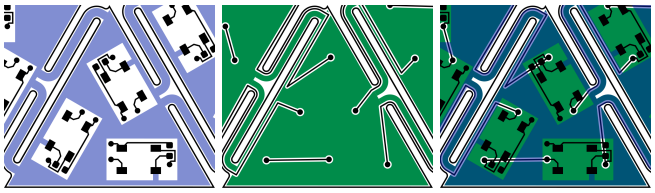
One key limitation of our approach are the constraints on the input mesh for it to be compatible with our system. The input mesh is incompatible if any of the hinges is too short for the required bending angle. In that case, the mesh is rejected and will need to be scaled up before using it as an input 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 22, 24 and the corresponding SVGs in the supplemental material).

3.5 Chamfered support structure

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 of the support structure are chamfered such that the hinges can rest of them properly, see Figure 9. The convex edges of the support structures are offset by $h = d_\theta \cos(\theta/2)$. Figure 10 shows the impact of the chamfer on the state of the hinges after assembly.

4 AUTOMATIC CIRCUIT LAYOUT

We have seen how to fully determine the PCB outline in Section 3. We 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. Note that each LED is paired with a 100nF decoupling capacitor connected to ground (gnd) and power (vcc).



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

Fig. 11. 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.

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 [Lienig and Scheible 2020], 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 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 connected respectively 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 are *carved out* from these planes: the component and via footprints, the data traces. This is illustrated in Figure 11. 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 long enough (multiple meters) to have a non-negligible resistance.

Modules. We group a LED and its accompanying capacitor into a *module*: a fixed circuit with a rectangular outline, having predefined solder pads and internal traces, as well as connection points where external traces should connect. Figure 12 shows LED modules for

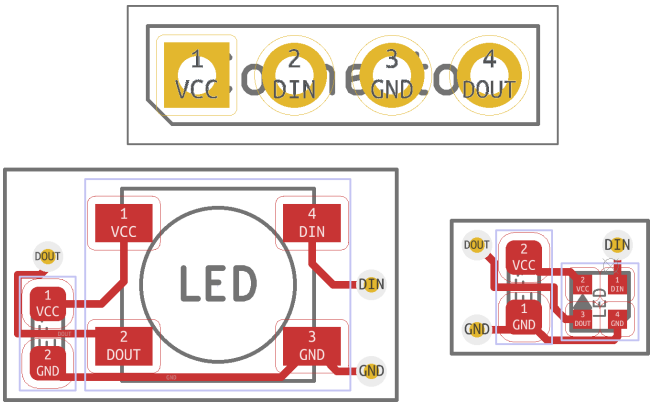


Fig. 12. Connector and modules containing an LED and a capacitor, shown at the same scale. Top: Connector (11.2×3.6 mm^2). Left: Big module (5.0×5.0 mm^2 WS2812B LED) with dimensions 10.2×6.0 mm^2. Right: Small module (1.5×1.5 mm^2 SK6805-EC15 LED) with dimensions 5.11×3.46 mm^2. Data traces enter a 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.

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.

4.1 Overall strategy

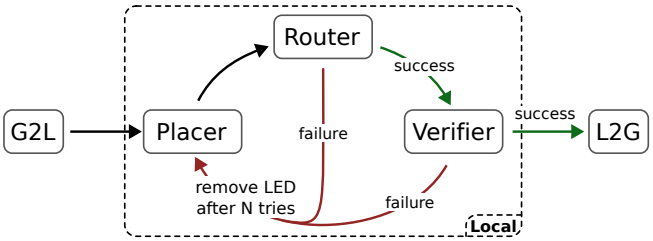


Fig. 13. PCB blueprint generation pipeline. *G2L* and *L2G* are short for *global-to-local* and *local-to-global* respectively. Everything in the dashed box is done locally per-triangle.

Generating the circuit requires solving for both the *placement* of the modules, their *ordering* 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 ordering 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 13. 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*.

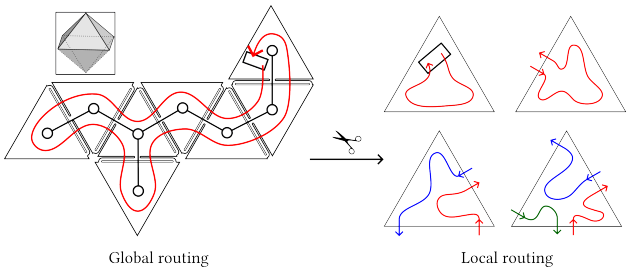


Fig. 14. Each unfolded patch is traversed in depth-first order starting at the connector (left), traversing hinges right-side first. This counter-clockwise order defines a cycle on the unfolding and fixes the inputs and outputs of each triangle (right).

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.

4.2 Global to local

We split the place-order-route problem into per-triangle, independent sub-problems. 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 14. 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 as defined by the traversal order. This facilitates routing as explained in Section 4.3.2.

4.3 Per-triangle circuit layout

Each triangle is processed independently, making parallel computations possible if needed. Within each triangle we first place modules (Section 4.3.1) and then route the signals (Section 4.3.2). The resulting routing is then verified to ensure that additional design and electrical constraints are satisfied. (Section 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 4.3.4).

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 et al. 2020], with strict boundary and overlap constraints as well as a uniform distribution requirement. We propose a custom placer combining a classical Centroidal Voronoi Tessellation (CVT) with a rigid body collision solver (*Box2D*).

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 a fixed number of iterations (50, 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 distribution forces. These forces attract each module towards the center of its Voronoi cell, computed at each iteration from the module centers. 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. We use 800 iterations (experimentally determined).

The entire placement process is illustrated in Figure 15. 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 4.3.4). The number of modules to place is then given to the placer.

Placement efficiency and timings. We evaluate the efficiency of placement for a variety of triangle areas and shapes in Figure 16. Timings are given in Table 2 for different models. Step 4 of the placement algorithm dominates the timing. Based on the desired quality, the user may choose to end this step earlier thereby reducing the time spent in placement. We show the effect of the number of iterations in step 4 on the result in Figure 17. We always use 800 iterations in our results, which gives a good compromise between execution time and placement quality.

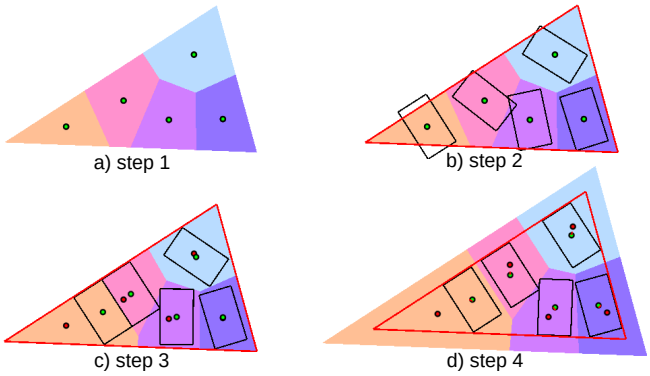


Fig. 15. Placement algorithm. Step 1: Placing points. Red points specify the Voronoi cell centers and green points specify the center point of modules. Step 2: Spawning rigid bodies. Step 3: Collision resolution. Step 4: Even distribution.

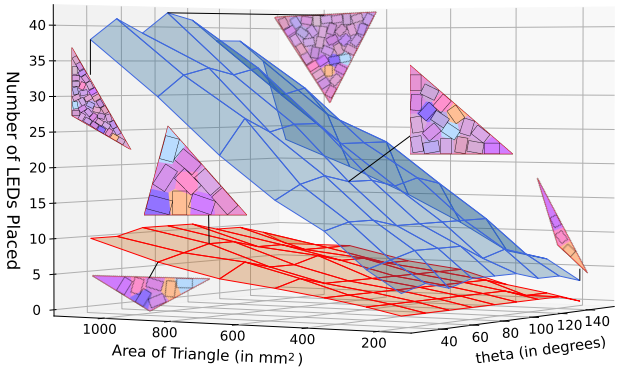


Fig. 16. Effect of the shape and area of a triangle on the number of placed LEDs. We place LED modules (small in blue, big in orange) in triangles with varying shapes and sizes. Triangle size is determined by its area and its shape is determined by the angle θ between two of its sides.

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 LED distribution corresponding to the hinges, while cut edges in the unfolding meet seamlessly. This results in noticeable hinge gaps (see Figure 18, left). With a proper choice of target spacing, the user can control the density so that the gaps are no longer noticeable (see Figure 18, right).

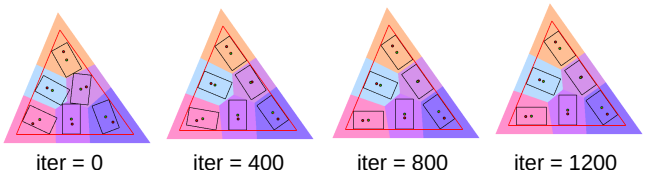


Fig. 17. Impact of the number of iterations spent on step 4 of the placer.

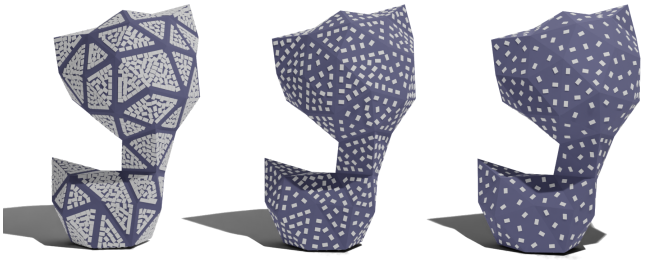


Fig. 18. The user controls the density of LEDs on the result by specifying a target spacing: (left) 5.0 mm (middle) 7.5 mm (right) 10 mm. The results show the renderings of *cat* model with different LED densities.

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

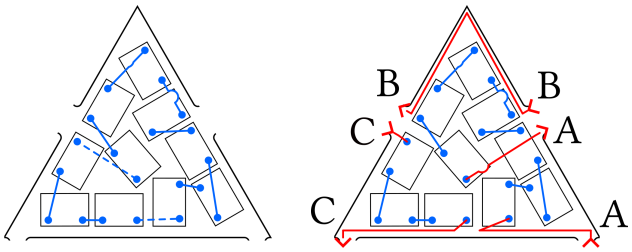
We design our router with the following overall principle, illustrated in Figure 19:

(1) Connect all modules to form a cycle, adding non-overlapping data segments between their data in/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 4.3.3) may detect issues and re-run the process with a different choice of parameters (Section 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 in-out 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.



(a) Module cycle (b) Hinge cuts

Fig. 19. 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.

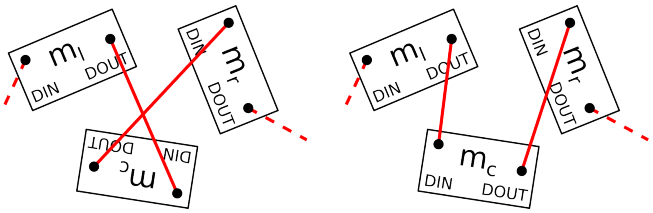


Fig. 20. Both orientations of m_c for given orientations of m_l and m_r. A bad orientation leads to an intersection (left) while a good orientation produces no intersection (right).

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 given that the non-intersection check 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 [Applegate et al. 2001]. 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. 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 20. 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, on the placement we produce stable modules tend to be spread out along the TSP sequence, with only short unstable sequences in-between. 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 small 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, see Figure 19. 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 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 as detailed in Section 4.3.4. 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 19).

If a valid set of hinge cuts is found, we proceed to the layout phase that performs additional verifications.

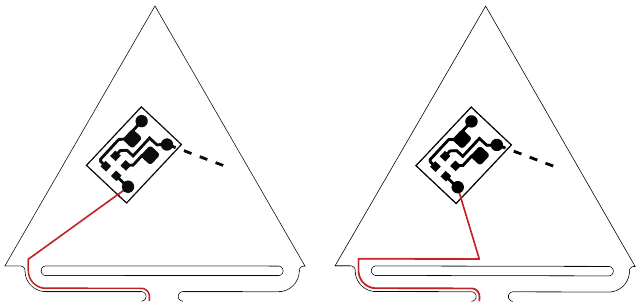
Half hinge parametric connections. Contrary to full hinges, the wide part of a half hinge makes the connection points for data traces be very far apart from one another (see Figure 4). This makes it more likely that direct traces from a module to this type of connection point will cause intersections 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 21. 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 4.2.

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. As such the generated layouts can sometimes be invalid. We thus have to run additional verifications on the generated geometry.



(a) Direct connection (b) Narrowing connection

Fig. 21. 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.

Verifications are done on the bottom layer to ensure that every the gnd plane contains all gnd vias in the triangle and is properly connected to the hinges, i.e. there are no disconnects or narrowings below a threshold. This is done with a morphological opening of the gnd plane, using the minimal trace width (0.2 mm) as 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.

4.3.4 Placement-routing loop. The routing process may fail either during routing itself (Section 4.3.2) or verification (Section 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.

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).

5 RESULTS

We demonstrate our approach on physical models and virtual examples. We first discuss the fabrication and assembly process (Section 5.1), then explain how we design the lighting effects along the surfaces (Section 5.2) and showcase our results on physical models

Table 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

(Section 5.3). We use our pipeline to produce additional virtual examples (Section 5.4). We conclude by analyzing the performance of the pipeline on our results (Section 5.5). The dimensions of all models are listed in Table 1.

We provide in supplemental material previews of the circuit blueprints for all results (SVG files). We also provide a video showcasing 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. SVGs illustrating the maximal density can be found in the supplemental material.

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 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.

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 – wrapping – 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.

5.1.1 Optional light diffuser. 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 housing for the LEDs. It folds easily onto the PCB with LED housings snapping in place onto the soldered components. A diffuser is shown before and after assembly in Figure 22.

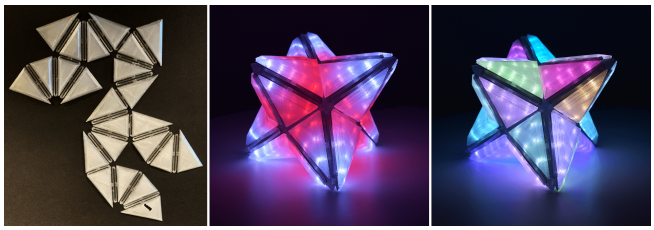


Fig. 22. 3D printed light diffuser for the *star* model (left). Light effects on the design: inside-out pulse (middle) and per triangle colors (right).

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, the LED id for which to compute the color. It returns the color as a 24 bits RGB triple directly sent to the LED. The shader has access to the entire mesh information (vertices, normals, topology). Sample shaders are provided with our code.

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 highly dynamic range of brightness. 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. The data is streamed through UART (1MBd) to an external LED controller implemented on an FPGA (Lattice ECP5). The controller runs at 100MHz and can drive several boards in parallel with up to 2048 LEDs per board in its current implementation.

5.3 Fabricated and lit results

We fabricated four designs, two small ones (*icosa*, *star*) and two larger ones (*cat*, *sqtorus*). The price of a single PCB (without components and assembly) ranges from $10 per unit (*icosa*) to $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.

icosa is a small sphere-like shape shown in Figure 23. 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 2 and 22). Our unfolder produces a single patch with mostly full hinges, featuring 166 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 the teaser (Figure 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

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

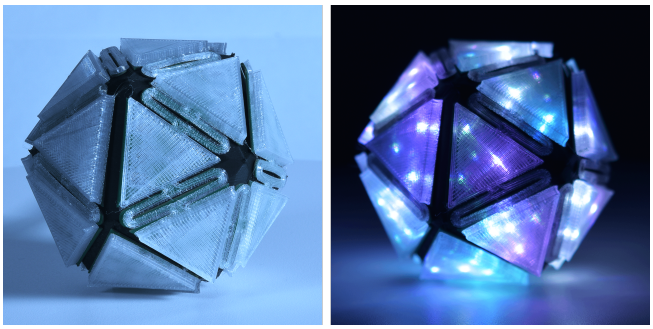


Fig. 23. *icosa* model with its diffuser (left), lit by a sparkling effect (right).

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.

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 24. 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 directional effects, and is sufficiently powerful to light a space – drawing upwards of 3 A on full power. Lighting effects are shown in Figure 25. The tileable pieces can be arranged in different, possibly larger shapes – an example appears in the accompanying video. Figure 30 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.

5.4 Virtual examples

We run our pipeline on additional models, for which statistics are given in Table 2.

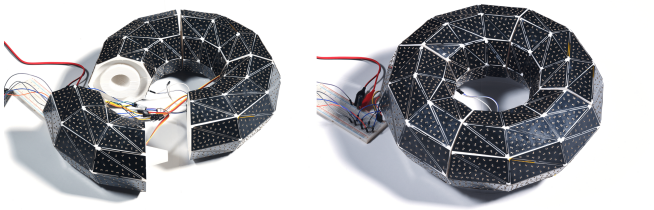


Fig. 24. A torus assembled from four *sqtorus* tiles. Note that different assemblies from *sqtorus* tiles are possible. See supplemental material for another virtual configuration.

batman is a mask covered with LEDs, shown in Figure 26. 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, see Figure 28.

Our method allows to choose between different LED modules and to preview the expected outlook after fabrication. Figure 29 shows the usage of our pipeline on big and small 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 18. The preview of the final appearance is faster than running the whole pipeline to generate the real blueprints, since we can skip the routing and verification steps.

5.5 Statistics on different models

Key statistics regarding the performance of our pipeline are given in Table 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

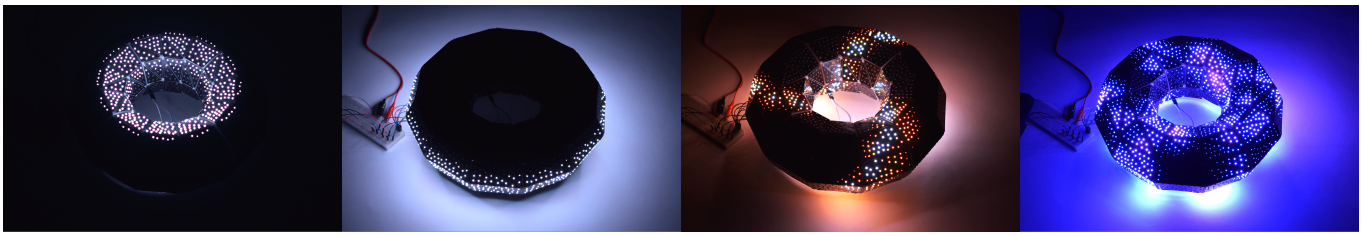


Fig. 25. The torus produces directional light patterns that strongly reflect on its surroundings. These effects are best visualized in the accompanying video.

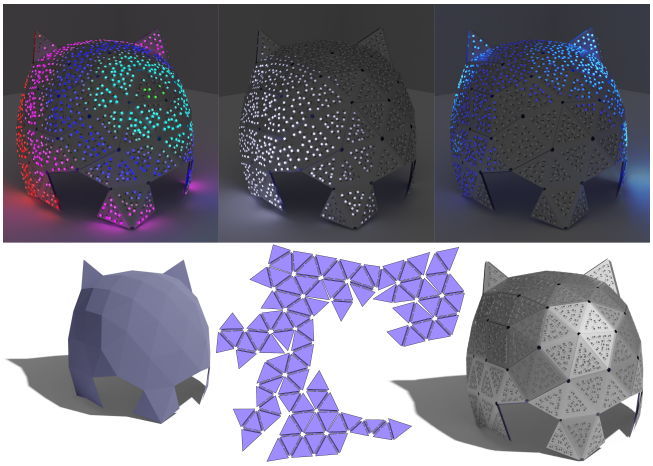


Fig. 26. *batman* (virtual result). (top) Different light effects, (bottom) original mesh, unfolding, and folded PCB.

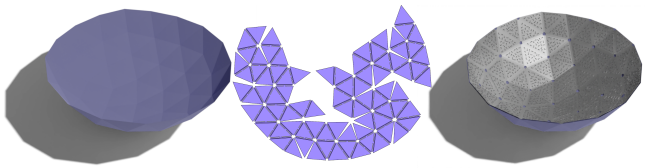


Fig. 27. *dome* (virtual result). Original mesh, unfolding, folded PCB.

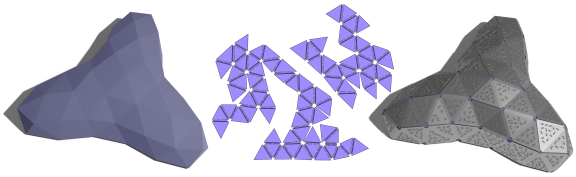


Fig. 28. *archi* (virtual result). Mesh, unfolding (2 patches), folded PCB.

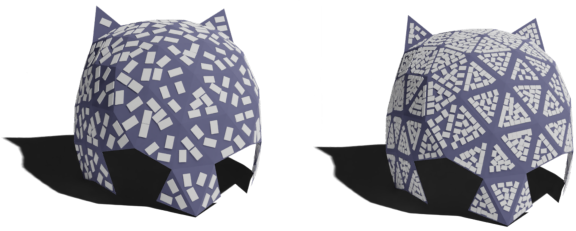


Fig. 29. *batman* model covered with with big LED modules (left) and small LED modules (right).

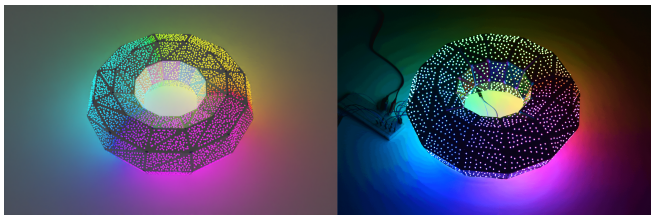


Fig. 30. Preview (left) compared to physical model (right).

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 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* (big LEDs) 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*.

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 3.4. 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 trade-offs 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.

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.

ACKNOWLEDGMENTS

We thank the reviewers for the valuable feedback. We also thank the Miba Machine Shop at ISTA, PCBWay, and PragoBoard for helping us with fabrication and assembly. This project was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 715767 -- MATERIALIZABLE).

REFERENCES

Altium. 2014. Designing a Rigid-Flex PCB in Altium Designer. https://www.altium.com/documentation/altium-designer/designing-rigid-flex-pcb.

Byoungkwon An, Ye Tao, Jianzhe Gu, Tingyu Cheng, Xiang 'Anthony' Chen, Xiaoxiao Zhang, Wei Zhao, Youngwook Do, Shigeo Takahashi, Hsiang-Yun Wu, Teng Zhang, and Lining Yao. 2018. Thermorph: Democratizing 4D Printing of Self-Folding Materials and Interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173834

Kristin Angel, Harvey H. Tsang, Sarah S. Bedair, Gabriel L. Smith, and Nathan Lazarus. 2018. Selective electroplating of 3D printed parts. *Additive Manufacturing* 20 (2018), 164–172. https://doi.org/10.1016/j.addma.2018.01.006

David Applegate, Robert E. Bixby, Vašek Chvátal, , and William J. Cook. 2001. Concorde TSP. https://www.math.uwaterloo.ca/tsp/concorde/index.html.

Autodesk. 1988. EAGLE. https://www.autodesk.com/products/eagle/overview.

Bob Burns. 2020. How to Reduce Rigid Flex PCB Costs. `https://www.printedcircuits.com/blog/reduce-rigid-flex-pcb-costs/` (Oct. 2020).

Sebastien JP Callens and Amir A Zadpoor. 2018. From flat sheets to curved geometries: Origami and kirigami approaches. *Materials Today* 21, 3 (2018), 241–264.

Jean-Pierre Charras. 1992. KiCad. https://www.kicad.org/.

Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and wenping wang. 2017. Fabricable Tile Decors. *ACM Trans. Graph.* 36, 6, Article 175 (nov 2017), 15 pages. https://doi.org/10.1145/3130800.3130817

Weikai Chen, Xiaolong Zhang, Shiqing Xin, Yang Xia, Sylvain Lefebvre, and Wenping Wang. 2016. Synthesis of Filigrees for Digital Fabrication. *ACM Trans. Graph.* 35, 4, Article 98 (jul 2016), 13 pages. https://doi.org/10.1145/2897824.2925911

Ketan Dalal, Allison W. Klein, Yunjun Liu, and Kaleigh Smith. 2006. A Spectral Approach to NPR Packing. In *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) *(NPAR '06)*. Association for Computing Machinery, New York, NY, USA, 71–78. https://doi.org/10.1145/1124728.1124741

Erik D. Demaine and Joseph O'Rourke. 2007. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, Cambridge ; New York.

Dania El-Khechen, Muriel Dulieu, John Iacono, and Nikolaj van Omme. 2009. Packing 2×2 unit squares into grid polygons is NP-complete. In *Canadian Conference on Computational Geometry.*

Filippo Andrea Fanni, Fabio Pellacini, Riccardo Scateni, and Andrea Giachetti. 2022. PAVEL: Decorative Patterns with Packed Volumetric Elements. *ACM Trans. Graph.* 41, 2, Article 19 (jan 2022), 15 pages. https://doi.org/10.1145/3502802

Patrick F. Flowers, Christopher Reyes, Shengrong Ye, Myung Jun Kim, and Benjamin J. Wiley. 2017. 3D printing electronic components and circuits with conductive thermoplastic filament. *Additive Manufacturing* 18 (2017), 156–163. https://doi.org/10.1016/j.addma.2017.10.002

Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. 1981. Optimal Packing and Covering in the Plane are NP-Complete. *Inf. Process. Lett.* 12 (1981), 133–137.

Daniel Groeger and Jürgen Steimle. 2018. ObjectSkin: Augmenting Everyday Objects with Hydroprinted Touch Sensors and Displays. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4, Article 134 (jan 2018), 23 pages. https://doi.org/10.1145/3161165

Daniel Groeger and Jürgen Steimle. 2019. LASEC: Instant Fabrication of Stretchable Circuits Using a Laser Cutter. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300929

Ollie Hanton, Michael Wessely, Stefanie Mueller, Mike Fraser, and Anne Roudaut. 2020. *ProtoSpray: Combining 3D Printing and Spraying to Create Interactive Displays with Arbitrary Shapes.* Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376543

Alejo Hausner. 2001. Simulating Decorative Mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 573–580. https://doi.org/10.1145/383259.383327

Liang He, Jarrid A. Wittkopf, Ji Won Jun, Kris Erickson, and Rafael Tico Ballagas. 2022. ModElec: A Design Tool for Prototyping Physical Computing Devices Using Conductive 3D Printing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 4, Article 159 (2022), 20 pages.

Steve Hodges, Nicolas Villar, Nicholas Chen, Tushar Chugh, Jie Qi, Diana Nowacka, and Yoshihiro Kawahara. 2014. Circuit Stickers: Peel-and-Stick Construction of Interactive Electronic Prototypes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1743–1746. https://doi.org/10.1145/2556288.2557150

Freddie Hong, Connor Myant, and David E Boyle. 2021. *Thermoformed Circuit Boards: Fabrication of Highly Conductive Freeform 3D Printed Circuit Boards with Heat Bending.*

W. Hu, Z. Chen, H. Pan, Y. Yu, E. Grinspun, and W. Wang. 2016. Surface Mosaic Synthesis with Irregular Tiles. *IEEE Transactions on Visualization and Computer Graphics* 22, 03 (mar 2016), 1302–1313. https://doi.org/10.1109/TVCG.2015.2498620

Naveen Noah Jason, Wei Shen, and Wenlong Cheng. 2015. Copper nanowires as conductive ink for low-cost draw-on electronics. *ACS applied materials & interfaces* 7, 30 (2015), 16760–16766.

Andriani-Melina Kalama, Danai Tzoni, and Ioanna Symeonidou. 2020. Kerf Bending: A Genealogy Of Cutting Patterns For Single And Double Curvature.

Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang, and Gregory D. Abowd. 2013. 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*. 363–372.

Myung Jun Kim, Mutya A. Cruz, Shengrong Ye, Allen L. Gray, Gabriel L. Smith, Nathan Lazarus, Christopher J. Walker, Hjalti H. Sigmarsson, and Benjamin J. Wiley. 2019. One-step electrodeposition of copper on conductive 3D printed objects. *Additive Manufacturing* 27 (2019), 318–326. https://doi.org/10.1016/j.addma.2019.03.016

Thorsten Korpitsch, Shigeo Takahashi, Eduard Gröller, and Hsiang-Yun Wu. 2020. Simulated annealing to unfold 3d meshes and assign glue tabs. (2020).

Dongchi Lee, Kazuya Saito, Takuya Umedachi, Tung D Ta, and Yoshihiro Kawahara. 2018. 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*. 392–395.

Yu-Ki Lee, Zhonghua Xi, Young-Joo Lee, Yun-Hyeong Kim, Yue Hao, Hongjin Choi, Myoung-Gyu Lee, Young-Chang Joo, Changsoon Kim, Jyh-Ming Lien, and In-Suk Choi. 2020. Computational wrapping: A universal method to wrap 3D-curved surfaces with nonstretchable materials for conformal devices. *Science Advances* 6, 15 (2020), eaax6212.

Jens Lienig and Juergen Scheible. 2020. *Fundamentals of Layout Design for Electronic Circuits.* Springer International Publishing. https://doi.org/10.1007/978-3-030-39284-0

Research Association Mechatronic Integrated Devices 3-D MID. 1992. 3D-MID Technology. https://www.3d-mid.de/en/technology/.

Giovanni Gerardo Muscolo, Giacomo Moretti, and Giorgio Cannata. 2019. SUAS: A Novel Soft Underwater Artificial Skin with Capacitive Transducers and Hyperelastic Membrane. *Robotica* 37, 4 (2019), 756–777. https://doi.org/10.1017/S0263574718001315

Hyunjoo Oh, Tung D. Ta, Ryo Suzuki, Mark D. Gross, Yoshihiro Kawahara, and Lining Yao. 2018. *PEP (3D Printed Electronic Papercrafts): An Integrated Approach for 3D Sculpting Paper-Based Electronic Devices.* 1–12.

Simon Olberding, Sergio Soto Ortega, Klaus Hildebrandt, and Jürgen Steimle. 2015. 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 and Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 223–232. https://doi.org/10.1145/2807442.2807494

Simon Olberding, Michael Wessely, and Jürgen Steimle. 2014. PrintScreen: Fabricating Highly Customizable Thin-Film Touch-Displays. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 281–290. https://doi.org/10.1145/2642918.2647413

Luis Paredes, Sai Swarup Reddy, Subramanian Chidambaram, Devashri Vagholkar, Yunbo Zhang, Bedrich Benes, and Karthik Ramani. 2021. FabHandWear: An End-to-End Pipeline from Design to Fabrication of Customized Functional Hand Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 2, Article 76 (jun 2021), 22 pages. https://doi.org/10.1145/3463518

Bart Plovie, Maarten Cauwe, Frederick Bossuyt, and Jan Vanfleteren. 2017a. One-Time Deformable Thermoplastic Devices Based on Flexible Circuit Board Technology. ESA, 1.

Bart Plovie, Yang Yang, Joren Guillaume, Sheila Dunphy, Kristof Dhaenens, Steven Van-Put, Björn Vandecasteele, Thomas Vervust, Frederick Bossuyt, and Jan Vanfleteren. 2017b. Arbitrarily Shaped 2.5D Circuits using Stretchable Interconnects Embedded in Thermoplastic Polymersθ. *Advanced Engineering Materials* 19, 8 (2017), 1700032.

Konrad Polthier. 2009. Imaging maths - Unfolding polyhedra.

Jie Qi and Leah Buechley. 2010. 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)*. Association for Computing Machinery, New York, NY, USA, 121–128.

Steven I. Rich, Zhi Jiang, Kenjiro Fukuda, and Takao Someya. 2021. Well-rounded devices: the fabrication of electronics on curved surfaces – a review. *Materials Horizons* 8 (2021), 1926–1958. Issue 7.

Analisa Russo, Bok Yeop Ahn, Jacob J. Adams, Eric B. Duoss, Jennifer T. Bernhard, and Jennifer A. Lewis. 2011. Pen-on-Paper Flexible Electronics. *Advanced Materials* 23, 30 (2011), 3426–3430.

Wolfram Schlickenrieder. 1997. Nets of polyhedra. *Unpublished. Technische Universitat Berlin* (1997).

Geoffrey C. Shephard. 1975. Convex polytopes with convex nets.

Kaleigh Smith, Yunjun Liu, and Allison Klein. 2005. Animosaics. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 201–208. https://doi.org/10.1145/1073368.1073397

Raphael Straub and Hartmut Prautzsch. 2011. *Creating optimized cut-out sheets for paper models from meshes.* Citeseer.

Saiganesh Swaminathan, Kadri Bugra Ozutemiz, Carmel Majidi, and Scott E. Hudson. 2019. *FiberWire: Embedding Electronic Function into 3D Printed Mechanically Strong, Lightweight Carbon Fiber Composite Objects.* 1–11.

Shigeo Takahashi, Hsiang-Yun Wu, Seow Hui Saw, Chun-Cheng Lin, and Hsu-Chun Yen. 2011. Optimized topological surgery for unfolding 3d meshes. In *Computer graphics forum*, Vol. 30. Wiley Online Library, 2077–2086.

J J Toriz-Garcia, J J Cowling, G L Williams, Q Bai, N L Seed, A Tennant, R McWilliam, A Purvis, F B Soulard, and P A Ivey. 2013. Fabrication of a 3D electrically small antenna using holographic photolithography. *Journal of Micromechanics and Microengineering* 23, 5 (mar 2013), 055010. https://doi.org/10.1088/0960-1317/23/5/055010

Cesar Torres, Jasper O'Leary, Molly Nicholas, and Eric Paulos. 2017. Illumination Aesthetics: Light as a Creative Material within Computational Design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6111–6122. https://doi.org/10.1145/3025453.3025466

Nobuyuki Umetani and Ryan Schmidt. 2017. SurfCuit: Surface-Mounted Circuits on 3D Prints. *IEEE Computer Graphics and Applications* 37, 3 (2017), 52–60.

Guanyun Wang, Fang Qin, Haolin Liu, Ye Tao, Yang Zhang, Yongjie Jessica Zhang, and Lining Yao. 2020. MorphingCircuit: An Integrated Design, Simulation, and Fabrication Workflow for Self-Morphing Electronics. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4 (2020). https://doi.org/10.1145/3432232

Tianyi Wang, Ke Huo, Pratik Chawla, Guiming Chen, Siddharth Banerjee, and Karthik Ramani. 2018. Plain2Fun: Augmenting Ordinary Objects with Surface Painted Circuits. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems.* 1–6.

Hao Wu, Yu Tian, Haibo Luo, Hui Zhu, Yongqing Duan, and YongAn Huang. 2020. Fabrication Techniques for Curved Electronics on Arbitrary Surfaces. *Advanced Materials Technologies* 5, 8 (2020), 2000093. https://doi.org/10.1002/admt.202000093 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/admt.202000093

Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. 2020. TilinGNN: Learning to Tile with Self-Supervised Graph Neural Network. *ACM Trans. Graph.* 39, 4, Article 129 (jul 2020), 16 pages. https://doi.org/10.1145/3386569.3392380

Junichi Yamaoka, Mustafa Doga Dogan, Katarina Bulovic, Kazuya Saito, Yoshihiro Kawahara, Yasuaki Kakehi, and Stefanie Mueller. 2019. FoldTronics: Creating 3D Objects with Integrated Electronics Using Foldable Honeycomb Structures. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300858

Zeyu Yan, Anup Sathya, Sahra Yusuf, Jyh-Ming Lien, and Huaishu Peng. 2022. 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* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 12, 13 pages. https://doi.org/10.1145/3526113.3545652

Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Kevin Shum, Jessica Ayeley Quaye, and Stefanie Mueller. 2020a. *CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form.* 1–13.

Junyi Zhu, Yunyi Zhu, Jiaming Cui, Leon Cheng, Jackson Snowden, Mark Chounlakone, Michael Wessely, and Stefanie Mueller. 2020b. *MorphSensor: A 3D Electronic Design Tool for Reforming Sensor Modules.* Association for Computing Machinery, New York, NY, USA, 541–553. https://doi.org/10.1145/3379337.3415898