An Open-Source Tool for FEM Modeling of Bent Flexible Circuits

Nico Arnold, Andreas Krinke, Manfred Dietrich, *Member, IEEE*, and Jens Lienig, *Senior Member, IEEE*

***Abstract*—Predicting the properties of flexible printed circuits (FPCs) before production is a significant challenge in their development. Traditional design flows model FPCs in a flat state, yet simulations of the actual bent shape of the circuits are required in forecasting their final, real-world behavior. Currently, only proprietary tools are available for simulating these 3-D geometries, with no open-source alternatives. This article introduces a novel open-source software tool that bridges this gap by transforming 2-D layout data from KiCad files into 3-D bent meshes ready for simulation. Initially, retrieved geometries are rendered according to a predefined stackup. To enable a seamless simulation workflow, the geometries must then undergo preprocessing, meshing, and labeling. This article significantly extends an earlier published algorithm by: 1) supporting more bend shapes and 2) eliminating dependencies on external programs for geometry generation, omitting the necessity to do the meshing manually, and generating physical groups. These steps are now integrated in our tool, facilitating automatic mesh generation from layout files in a single step. Consequently, the software generates ready-to-use finite element method (FEM) meshes for external simulators, thus supporting a comprehensive open-source design and verification workflow. This work not only enhances the accessibility of FPC design and simulation but also underscores the broader applicability of open-source solutions for printed circuit board (PCB) design.**

***Index Terms*—Finite element method (FEM), flexible circuit, flexible printed circuit (FPC), open source, simulation.**

I. Introduction

A. Motivation

FLEXIBLE printed circuits (FPCs) provide significant advantages over traditional rigid printed circuit boards (PCBs). Particularly beneficial in projects with complex geometries or limited space, FPCs use a bendable substrate [often polyimide (PI)], in contrast to the fiberglass-reinforced epoxy (FR4) [2] used in rigid PCBs. The inherent flexibility of FPCs enables circuits that can adapt to curved surfaces or be folded to reduce their spatial footprint. In addition, the material properties of PI are advantageous in high-frequency (HF) applications [3].

Despite these benefits, designing FPCs poses substantial challenges. Conventional layout designs are executed in 2-D, yet the final form of these circuits is inherently 3-D. This introduces challenges in visualizing and planning the final product. Effective design processes must preemptively address potential issues such as component interference and undesirable electrical or thermal interactions, e.g., crosstalk or indirect heating via external parts. Furthermore, precise component placement during the design phase enables seamless assembly and optimal functionality in the final 3-D structure (Fig. 1).

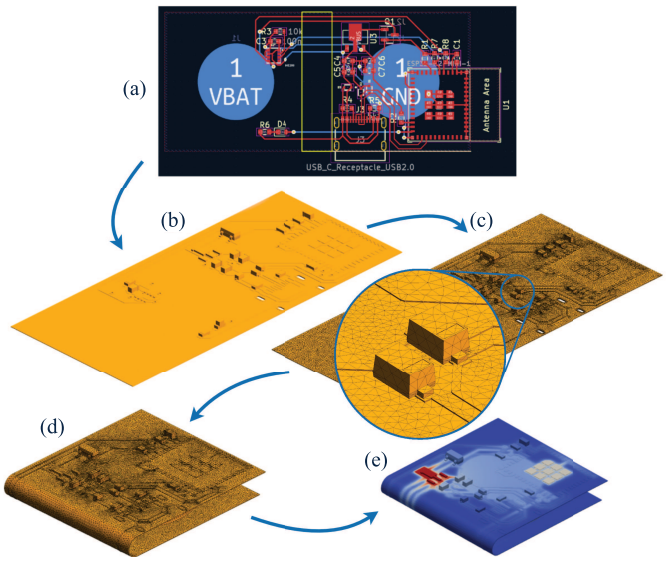

Fig. 1. Individual steps in our workflow. (a) Layout design, (b) geometry generation, (c) meshing, (d) bending, and (e) simulation of an FPC.

B. State of the Art

To date, 3-D-simulation software is only available under proprietary licenses from companies like *Altium* and *Cadence*. For smaller companies and projects, the licensing cost is often too high. Moreover, users are constrained to the functionality the software offers and lack possibilities to enhance capabilities, raising demand for a free or open-source alternative.

Open-source tools benefit from the fact that their code is open for access and modification by any individual. This enables teams to adapt the workflows involved to their specific requirements and create interfaces to specialized tools. Those factors lead to an increased flexibility and versatility of a program.

In addition, the software's user community can contribute to its testing, further development, and refinement. It facilitates

Received 24 June 2025; accepted 16 July 2025. Date of publication 31 July 2025; date of current version 7 October 2025. This work was supported by German Federal Ministry of Education and Research (BMBF) under Grant 16ME0473. An earlier version of this paper was presented at the IFETC [DOI: 10.1109/IFETC61155.2024.10771875]. *(Corresponding author: Nico Arnold.)*

The authors are with the Institute of Electromechanical and Electronic Design, Dresden University of Technology, 01069 Dresden, Germany (e-mail: nico.arnold@tu-dresden.de).

Digital Object Identifier 10.1109/JFLEX.2025.3594658

Please cite as: N. Arnold, A. Krinke, M. Dietrich and J. Lienig, "An Open-Source Tool for FEM Modeling of Bent Flexible Circuits," in IEEE Journal on Flexible Electronics, vol. 4, no. 8, pp. 333-341, Aug. 2025

the extension of the tool's scope to other aspects of simulation and modeling and enables a wider range of tests. In addition, the transparency of open-source software allows for increased security because it can be examined by the community. With many eyes reviewing the code, bugs and security vulnerabilities can be identified and addressed quickly. Consequently, this leads to more robust, accurate, and secure software.

While the existing open-source PCB design tools like *KiCad* [4] are widely used to design rigid PCBs, they lack built-in capabilities for the design, modeling, and simulation of FPCs. Therefore, several plugins have been developed, e.g., to implement finite element method (FEM) simulation in KiCad in the past years. One example is the *Kicad-nikfemm* [5] plugin, which was developed to simulate and plot power densities and voltages on PCB geometries. It was introduced in 2024 and is based on the author's own FEM implementation *nikfemm* [6] published a year earlier. However, *nikfemm* is constrained to 2-D geometries, and the corresponding plugin assumes the stackup to comprise two copper layers of 35 μm.

Kicad_pdn [7] is a similar piece of software. As with the previous example, it possesses good visualization capabilities, but also lacks support for specifying the thickness of copper layers.

While aforementioned projects are smaller plugins for simulating current density in power delivery networks (PDNs), the *KiCad-sparselizard* branch [8] appears to have a more ambitious objective. In addition to current density and voltage drop simulations, it facilitates capacitance simulations on a native KiCad interface. This integration ensures a seamless workflow within the main software.

Despite the enhanced support for FEM simulations facilitated by aforementioned software, KiCad continues to lack tools for defining bending shapes or visualizing and simulating FPCs in their final, 3-D form. This is where our contribution comes in.

C. Our Contribution

In a previous paper [1], we presented the workflow depicted in Fig. 2 that uses exclusively open-source software to visualize and simulate flexible circuits. We developed the application Fold the Line (FTL) based on this workflow, allowing users to bend circuits designed in the open-source program KiCad by applying specific bending parameters, filling the gap between FPC design and simulation. Still, the application discussed there had several major limitations.

1) It supported only .kicad_pcb input files, creating a dependency on KiCad.
2) It relied on external tools — FreeCAD [9] and fcad_pcb [10] — to generate 3-D geometries.
3) Meshing was performed in Salome Meca [11], requiring extensive manual intervention.
4) Bending was restricted to directions parallel to the x- or y-axis.

To enhance the efficiency and memory utilization of our application, as well as to increase its functional adaptability, it was necessary to eliminate unnecessary external dependencies and to address and improve the following aspects:

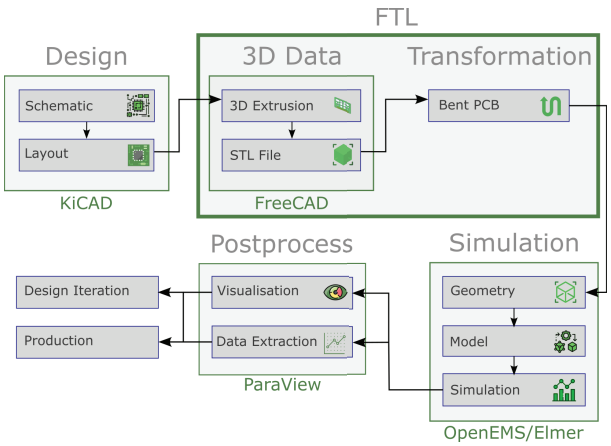

Fig. 2. Workflow for the simulation of FPCs. Our tool for transforming geometries (FTL) is highlighted.

1) Support for alternative input data formats like Gerber files and DXF, enabling the use of most PCB layout tools.
2) Integrated meshing functionality supporting automatic meshing with minimal manual effort.
3) Capabilities for automatic model structuring and labeling to assign material parameters and constraints for simulation,
4) Support for bending the FPC along any line.

These extensions are presented in this article. The updated workflow, still named *FTL*, diverges from the previous one in that it no longer relies on *FreeCAD* and *Salome* and instead uses *gmsh* [12] for the generation, annotation, and meshing of the 3-D geometries. In addition, we added new transformation schemes to support bending along any line and, thus, enabling more design flexibility. To illustrate the capabilities of this workflow, we use an example circuit described in Section III to demonstrate the transformation into its bent configuration and the thermal simulation of the generated geometry.

The source code of our new and extended program FTL is licensed as open-source software and available on GitHub [13] for anyone to use and modify.

II. Method

In this section, we first describe our workflow for designing an FPC as shown in Fig. 3 (Section II-A), and then explain the detailed implementation of our new tool FTL (Section II-B).

A. Workflow

Our workflow begins with the design of the circuit as a flat PCB in either KiCad or any program that can export DXF or GDSII files. This ensures the generation of a valid 2-D input geometry that can be modeled for subsequent simulations.

1) PCB Design With KiCad: The most straightforward approach to designing the FPC uses the software *KiCad*. This software offers a variety of features, such as design rule check (DRC), electrical rule check (ERC), layout versus schematic check (LVS), and SPICE simulation, facilitating a correct design through powerful checks to mitigate potential

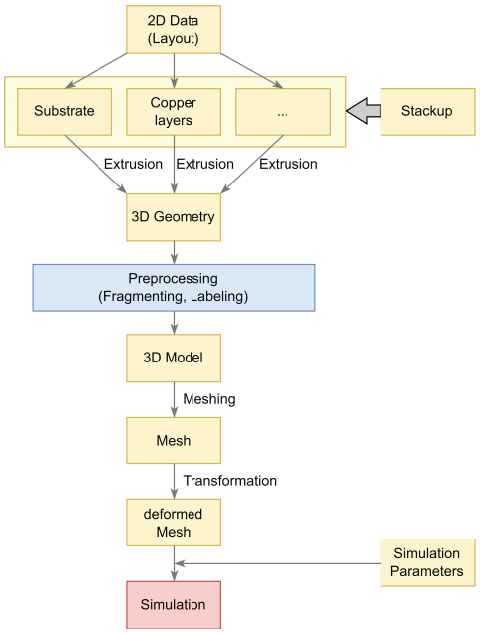

Fig. 3. Workflow for processing geometries in FTL and their applicability for simulations.

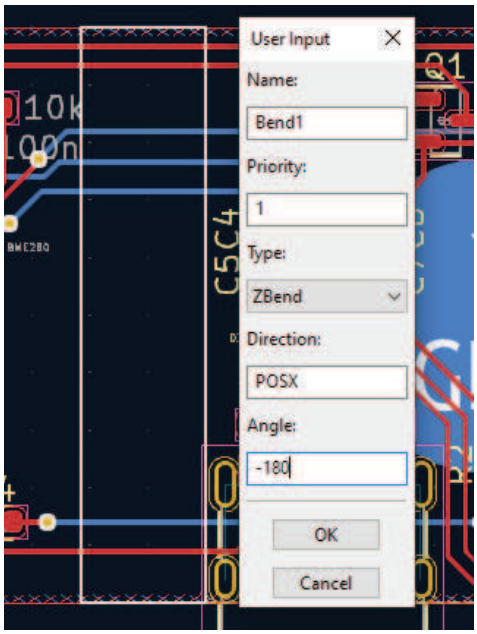

Fig. 4. Example configuration of a bend in *KiCad* with the bending area highlighted in white.

issues from the beginning of the design process. Following the schematic design, layout is designed. In this step, the involved electronic components are arranged and wired. Care must be taken to ensure correct placement to avoid collision of the components in the bent state and to treat the bending area as keep-out zones.

We have developed a KiCad plugin to enable users to specify the bending areas and adjust their parameters, as pictured in Fig. 4. This is achieved by drawing rectangles and polygons on a dedicated layer. The scope of these geometries is always defined by their geometric boundaries; only areas of the PCB confined within the shape will be transformed. By selecting a shape and clicking on the plugin icon, parameters can be adjusted. These include the maximum bending angle at the end of the transformation and the direction (positive/negative *x*-/*y*-axis). Depending on the latter parameter, the first edge of the rectangle found in the corresponding direction will be defined as the start line for the bending, while the latter will be the end line corresponding to the maximum bending angle. The transformation data is then exported to a JSON file, making it accessible for our tool *FTL*.

As explained in our previous paper [1], KiCad layout files (`.kicad_pcb`) were used to generate the 3-D substrate and copper trace geometries as STL files using the fcad_pcb [10] macro in FreeCAD [9]. This macro extruded the 2-D polygons representing the copper routes into a 3-D stack of copper wires and substrate layers.

This outdated approach had the disadvantage of requiring the macro and FreeCAD software, which added heavy computational overhead. An additional disadvantage was that geometric groups could not be defined on the mesh in this way and the creation of the mesh took longer. Geometric groups are required to assign material properties and constraints for simulation.

To address these limitations, we created a custom geometry engine, allowing users to use python to generate geometries, extrude them, and perform Boolean operations. This resulted in a significantly reduced memory footprint, faster generation times, and an efficient method for automatically creating geometry groups to identify parts and pads during simulations. We go into the details of this implementation in Section II-B.1.

2) DXF File Import: Alternatively, a DXF file can be used to generate the model. This import uses a custom built-in DXF reader module that enables the generation of gmsh [12] geometries from the individual layers of the DXF file. These geometries are inherently 2-D as defined in the file, requiring extrusion and stacking of layers to form a 3-D model of the FPC. This can be done by defining the stackup in a YAML file, where each layer must be defined with its corresponding layer in the DXF file, its thickness, and material.

The primary benefit of DXF files is that they can be exported by a wide range of programs. This contributes to the format's widespread support and ease of use. This even enables users to draw circuit structures using a mechanical CAD program.

However, those programs do not offer the advanced functionality that KiCad offers. Conformity checks must be performed manually, and electric simulations on SPICE level are not available. Nevertheless, the DXF format can serve as a practical interface to other electrical CAD programs that are only capable of exporting other formats not supported by FTL. Often, these ECAD programs also offer utilities for analysis and checks, making FTL independent of *KiCad*.

3) GDSII File Import: GDSII files are the most widely used data format for layout files of integrated circuits and can be accessed and edited with the open-source tool *KLayout* [14]. Internally, they are structured similar to DXF files: A GDSII file contains a collection of named layers that each contain a variety of geometric entities. Contrary to the DXF format, in the GDSII format those layers' names are numeric codes

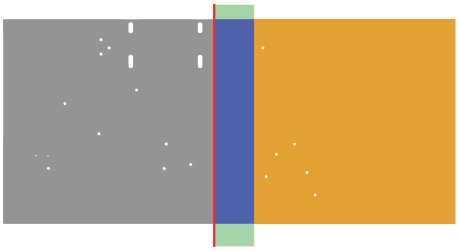

Fig. 5. Bending area (green), affected FPC area (blue), start line (red), and residual area (orange) as visualized in the application. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

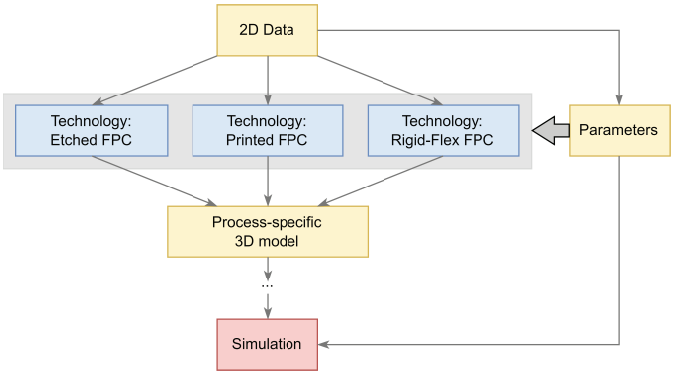

Fig. 6. Process-specific flow for rendering FPC geometries, in which 3-D geometry generation and preprocessing are conducted according to the respective PCB technology. The further steps toward simulation (cf. Fig. 3) are not shown.

instead of strings. Rendering, however, is the same as in DXF files. Support for GDSII files is a crucial requirement for a later step in the project: the simulation of IC structures on an FPC. This will allow more precise simulations of application specific integrated circuits (ASICs) in their designated environment. Moreover, vice versa, it allows for a more precise prediction of the impact of heat generation within the ASIC on the FPC.

Our open-source program FTL is able to create, mesh, and bend the 3-D circuit geometry created during the steps in Sections II-A.1–II-A.3. Subsequent to the geometry creation process that is explained in more detail later in Section II-B.1, the gmsh interface is used to create a mesh from the geometries that can be used as a data source for the transformation algorithm. The term *transformation* refers to the process of bending (transforming) the flat PCB into its bent shape and thus the displacement of the nodes according to the bending properties defined by the designer.

Prior to the transformation, the area to be bent is highlighted, as illustrated in Fig. 5. The individual areas of the PCB are highlighted in distinct colors here: transformations appear as green shapes, with the affected portion of the FPC shown in blue. A red line indicates the start line, marking the position at which the transformation begins to bend the FPC from its flat (horizontal) form. As the transformation progresses toward the opposite end of the shape, the bending angle increases, resulting in the end being bent at the maximum angle specified in the parameters of the corresponding transformation. The *residual*, as described later in Section II-B, denotes the portion of the FPC that is located beyond the end of the transformation. This is indicated by the orange coloring in Fig. 5. Nevertheless, it is necessary that this part is rotated and moved in a way that ensures it is tangential to the end of the transformed mesh. Said part extends to the edge of the FPC, or the beginning of a subsequent transformation. After confirmation, the algorithm is executed, and the bent circuit board is visualized and can be exported.

The mesh generated by our program FTL can be used for simulations. For this, the bent geometries can be further processed for advanced mesh operations and, for example, imported into *Elmer* [15] or *Palace* [16] to enable FEM simulations of the transformed circuit. Alternatively, straight geometries can be used in *OpenEMS* [17] to execute finite difference time domain (FDTD) simulations, as the FDTD method is not suited for bent geometries.

B. Implementation

Our program FTL has been developed in Python. This decision ensures a seamless usability across all the platforms to make it accessible for a wide user base. In addition, it offers access to a vast ecosystem of libraries, streamlining the development workflow and offering good interfaces to other programs, algorithms, and file formats. This facilitates the implementation of complex features with minimal effort. By leveraging these capabilities, the application can be deployed on various operating systems and integrated seamlessly with other applications without the need for significant modifications.

1) 3-D Geometry Generation: As shown in Fig. 3, the program starts by importing the information from the project-specific configuration file. This includes a variety of parameters and the location of the source file containing the geometry. For a KiCad PCB file, the stackup information is read from the corresponding location in the file, including the material, thickness, and order of the layers. Conversely, when dealing with a DXF file, it is necessary to define the stackup parameters within the configuration file itself.

Once the information about the stackup is available, the geometries are rendered. For DXF files, the layers are systematically traversed from bottom to top (according to the data in the stackup). For each layer, the geometries are extracted and rendered in gmsh. In contrast, the rendering of KiCad files is more intricate, as parts are rendered as “containers” holding geometries from multiple layers that correspond to the stackup layers. These parts must be rendered with an offset relative to the part’s position. The position of each part is specified in the respective entry, facilitating the calculation of the necessary coordinate offset. Geometries are then aggregated to be rendered in the associated main stackup layers.

An important point to note is that our workflow uses what we call *technology classes* to render a process-specific model as illustrated in Fig. 6. These files contain information about the thicknesses and materials involved in the manufacturing process and additional parameters that must be

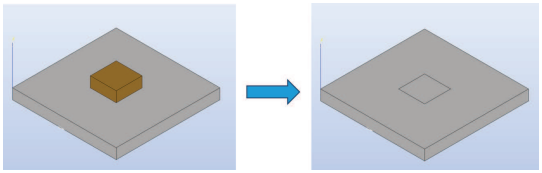

Fig. 7. Shared geometries in fragmented components.

considered during the generation of models for the given process. If a technology is provided, manual specification of the corresponding parameters in the configuration file becomes redundant. However, if these parameters are specified, the values configured in the technology file will be overridden.

Examples for additional parameters that can be specified are as follows.

1) *Via filling*: this is applicable for FPCs that are printed with conductive ink, leading to filled vias. For etched FPCs, vias will be hollow.
2) *Metallization thickness*: this is due to the fact that different technologies may be used, each using different methods and times for via metallization, resulting in different metal thicknesses for different technologies.

The primary benefit of this procedure is that it offers a more compact and comprehensive approach, because it

1) eliminates the need to define parameters for each individual model,
2) improves the management of technological parameters by ensuring they are consistent across all the projects, and
3) enables rendering a model for multiple technologies from a single input geometry, enabling effective comparison and analysis.

This proves to be advantageous in scenarios where simulations must be prepared for multiple technologies using the same input files.

2) Preprocessing: After geometry generation, careful preparation for the meshing process is crucial. To ensure the model's integrity and to prevent issues during the simulation process, cautious attention must be directed toward the interfaces between the individual components. A rigorous verification process is essential to ensure the nodes are consistent at the interface between all the geometries, thereby ensuring the creation of a both conforming and also uninterrupted mesh. This objective can be achieved using the *Boolean fragments* function of the gmsh Python interface. This operation replaces the tangential surfaces present in both the components with a single new surface belonging to each of the adjacent components, as shown in Fig. 7. Later, upon the subsequent mesh generation, this ensures that the boundary surfaces will be comprising identical surface elements and nodes.

In the context of early manual simulations, a prevalent challenge is the necessity of assigning geometries to *physical groups*. These allow the relevant geometries to be used in simulations by combining a number of volume or surface elements into groups, thus enabling the definition of parameters for specific parts of the model. This method ensures the accurate assignment of materials, initial conditions, or forces to components or surfaces. Within the gmsh framework, *physical groups* are enumerated in the order in which they are created. This necessitates maintaining precise records for each group, as the final model may contain a substantial number of them for large FPCs with higher amounts of components. Using said records, the subsequent assignment of groups is eased by mapping all the geometries to their corresponding group ID. This method enables referencing components with terms like "PartX.PadY" instead of the number of the associated physical group. It is imperative this is done before starting the meshing process; otherwise, changes will not be included in the mesh.

3) Meshing: The meshing of the 3-D model is conducted using gmsh which provides functions to define the grid size and shape of the generated mesh. This improves mesh generation for models that consist of both small and large dimensions. The mesh coarseness is automatically interpolated between areas of each kind, easing a seamless integration of those distinct dimensions. This feature is particularly useful in scenarios involving the simulation of electronic systems of higher complexity comprising smaller parts, facilitating the potential of ASIC simulations on their final FPC.

Following the aforementioned meshing process, the geometries are processed into a mesh consisting of smaller surface and volume elements according to their respective coarseness. Despite the conversion of the geometry, physical groups are maintained with the same IDs as previously assigned during the generation of geometry. This is vital to ensure accurate mapping to the respective geometric objects (bodies, surfaces) during the simulation. For simulation, the mesh can be exported into any file format supported by gmsh. Most commonly, this is `.msh` for Palace or `.unv` for Elmer.

At this point, it is noteworthy that all the functionality demonstrated is completely integrated within our tool. Contrary to the workflow exhibited in our previous paper [1], there is no dependency on FreeCAD and Salome Meca anymore in our workflow, as the geometry and mesh generation is processed internally by facilitating the direct gmsh Python interface. This integration streamlines the workflow, leading to reduced computing times and a minimized memory footprint.

4) Transformations: Transformations perform the transition from a flat FPC to its final shape. They define the regions where the circuit will undergo bending and the configuration of the resulting shape. In addition, they provide the possibility to restrict the deformation to specific regions without affecting neighboring areas, as shown in Figs. 4 and 5. The transformation is computed by an algorithm that iterates through all the nodes in the mesh confined inside the boundaries of the transformation. The position of each node between the start and end line is then mapped to an angle; nodes on the start line remain unaltered and will be assigned an angle of 0°, while nodes on the end line will undergo a bending by the maximum angle specified by the parameters for the transformation.

Transformations are hereby characterized as rectangles with coordinates $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$. Beginning at the start line (red line in Fig. 5) with a bending angle of 0°, the algorithm then iterates over all the nodes of the mesh between there and the end line of the specific transformation area. For

every node with position $\vec{p} = [p_x, p_y, p_z]^\intercal$, the position factor t is calculated [see (1)]. Based on the maximum angle α_{max} configured in the JSON file and the value of t, the bending angle α is calculated for $\vec{p}$ so that the last points on the end line are transformed with α_{max} [see (2)]. Moreover, the bending radius r is determined by the width of the transformation rectangle [see (3)]. For each node, the bend is then performed using a coordinate transformation matrix T to calculate the new position $\vec{p'}$ of the node [see (4)]. Equation (5) shows an example transformation matrix T for a bend in positive z-direction while moving in positive x-direction relative to the flat PCB

$$t = \frac{p_x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

$$\alpha = \alpha_{max} \cdot t \tag{2}$$

$$r = \frac{x_{max} - x_{min}}{\alpha_{max}} \tag{3}$$

$$\vec{p'} = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = T \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{4}$$

$$T = \begin{bmatrix} 0 & 0 & -\sin(\alpha) & x_{min} + r\sin(\alpha) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & r(1-\cos(\alpha)) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

Occasionally, there are still nodes remaining beyond the end of the transformation. This requires a so-called *residual transformation* as visualized in Fig. 5 (highlighted in orange). These areas must be kept in a straight plane, but it is still necessary to move and rotate them so that they tangentially extend the transformed FPC at the end line of the transformation. This is facilitated by a rotation of all the nodes just as the last nodes on the end line so they form a straight plane tangentially to the transformed area. Residual transformations are described by a transformation matrix R as calculated in (6). The transformed nodes are then calculated as in (4)

$$R = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & -x_{max}\cos(\alpha) + x_{min} + r\sin(\alpha) \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & -x_{max}\sin(\alpha) + r(1-\cos(\alpha)) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

More complex cases are transformations that bend the geometry along a line that is not parallel to either the x- or y-axis. In this case, the line connecting the first two points $\vec{p}_a$ and $\vec{p}_b$ of the transformation outline is used as the bending line, with the other points specifying the boundary of the transformation. Since the conventional x or y coordinates are inadequate for the distance to the start or end line, a more elaborate approach is required.

As a preliminary step, the start line of the transformation is converted into a linear equation, as illustrated in Fig. 8. Given the coordinates of the start line, specified as $\vec{p}_a = (x_a, y_a)$ and $\vec{p}_b = (x_b, y_b)$, along with the minimum and maximum dimensions of the boundary of the transformation scope, designated as x_{min}, x_{max}, y_{min}, and y_{max}, the slope m with the inclination angle α_z of the start line can be calculated by (7). Subsequently, this parameter can be used to retrieve the

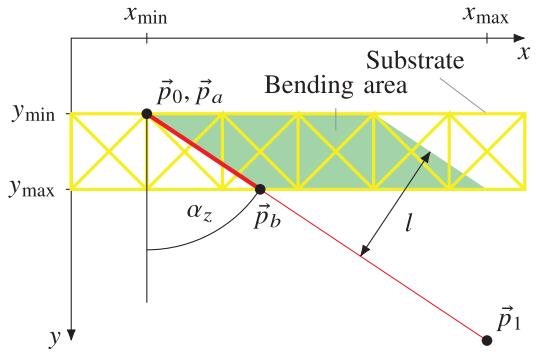

Fig. 8. Advanced bending with transformation boundary highlighted in green.

intercept n [see (8)] to create the line equation [see (9)]. From there, the y values at the lowest and highest x coordinate of the transformation need to be calculated by (10) and (11), respectively. The result is a line from $\vec{p}_0 = (x_{min}, y_0)$ to $\vec{p}_1 = (x_{max}, y_1)$ that elongates the start line across the entire transformation scope and enables the distance of individual points to be calculated

$$m = \frac{y_a - y_b}{x_a - x_b} \tag{7}$$

$$n = y_a - m \cdot x_a + (y_b - y_a) \tag{8}$$

$$y = m \cdot x + n \tag{9}$$

$$y_0 = m \cdot x_{min} + n \tag{10}$$

$$y_1 = m \cdot x_{max} + n \tag{11}$$

$$\alpha_z = \arctan m \tag{12}$$

$$M_{tr1} = \begin{bmatrix} \cos\alpha_z & -\sin\alpha_z & 0 & y_a\sin\alpha_z - x_a\cos\alpha_z \\ \sin\alpha_z & \cos\alpha_z & 0 & -x_a\sin\alpha_z - y_a\cos\alpha_z \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$$M_{tr2} = \begin{bmatrix} \cos -\alpha_z & -\sin -\alpha_z & 0 & x_a \\ \sin -\alpha_z & \cos -\alpha_z & 0 & y_a \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$T = \begin{bmatrix} 0 & 0 & -\sin(\alpha) & r\sin(\alpha) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & r(1-\cos(\alpha)) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

$$R = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & -x_{max}\cos(\alpha) + r\sin(\alpha) \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & -x_{max}\sin(\alpha) + r(1-\cos(\alpha)) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

$$\vec{p'} = M_{tr2} \cdot T \cdot M_{tr1} \cdot \vec{p}. \tag{17}$$

This formula can be used to get the distance of the transformation boundary's most remote point. This is assumed to be the length l of the transformation that corresponds to the maximum bending angle α_{max}. For each point in the transformation domain, its distance to the start line can be determined and mapped to a bending angle α similar to (1)–(3) by substituting $x_{max} - x_{min}$ with the maximum distance.

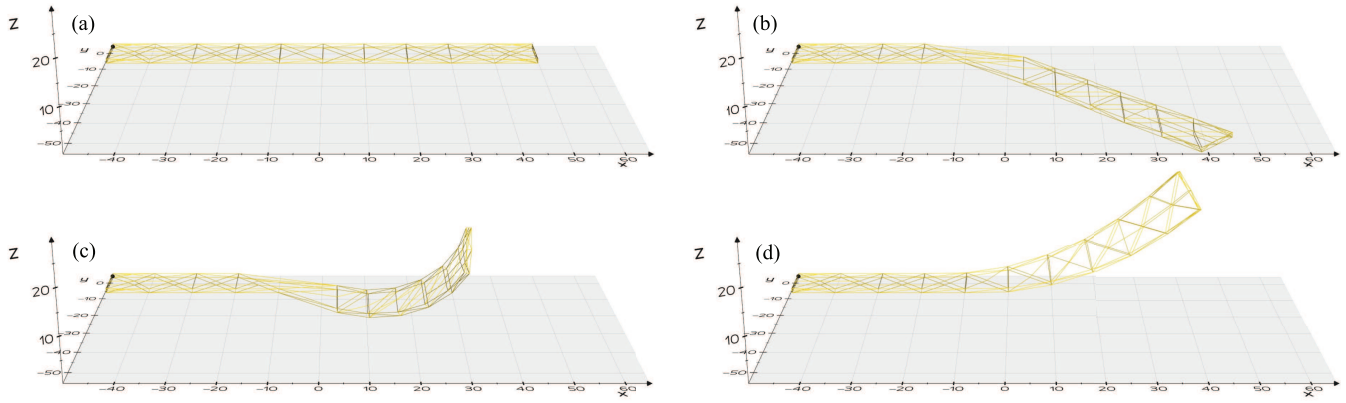

Fig. 9. Individual steps for a bend along lines not parallel to the x- or y- axis (a) original shape/mesh, (b) rotated geometry, (c) bent geometry, and (d) final shape (rotated back to original position).

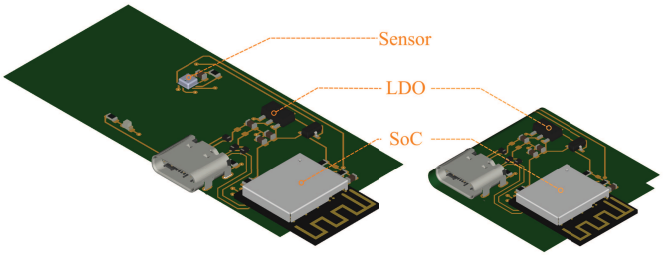

Fig. 10. Rendering of the FPC in its straight and bent shape.

The transformation itself is executed in the same manner as an axis-parallel bend with deformation along the positive x-axis outlined in (4) and (5). However, prior to this, the affected geometry must be rotated around the point $\vec{p}_a$ by the inclination angle α_z. The same procedure is repeated again once the deformation is computed to rotate the bent geometry back to the start line of the transformation as depicted in Fig. 9.

This objective is accomplished by considering the rotation around a point as a translation of the entire affected geometry, such that $\vec{p}_a$ lies at the origin of the coordinate system, followed by a subsequent translation back. To minimize computing time, the subsequent translation back and the translation to the origin for the second rotation can be omitted if the bending matrix and the residual matrix are adjusted accordingly like in (15) and (16), leading to the pre- and postbending transformation matrices given in (13) and (14), derived from the dot product of the corresponding transformation and rotation matrices.

The final coordinates of each node are calculated according to (17).

III. Experimental Results

A use case was developed to assess the capabilities of the workflow. This involves an FPC containing an *ESP32-H2* system-on-a-chip (SoC) with several peripheral components. Besides passive components, these include a low dropout (LDO) voltage converter for supplying the processor with power via USB, a *BME280* temperature/humidity sensor, and a *CR2032* lithium coin cell as primary power supply. The circuit is designed for collecting data concerning heat and humidity while being operated from a coin cell battery, with subsequent wireless transmission of the acquired data to a base station via ZigBee. However, it should be noted that the processor will generate waste heat, leading in turn to further heating of the sensor. This is detrimental to the precision of the sensor as it might heat it to a temperature higher than the ambient temperature. Furthermore, when powered via USB, the LDO will lead to an additional temperature increase at the sensor as it has to reduce the USB bus voltage of 5V to 3.3V needed by the ESP32-H2. The distinct feature of this scenario is the FPC's ability to bend around the CR2032 battery as illustrated in Fig. 10, thus eliminating the requirement for a dedicated battery holder and minimizing spatial dimensions. However, it is detrimental to sensor precision. This originates from the coin cell acting as a heat conductor, effectively providing a low-heat-resistance path from the parts generating heat to the sensor.

To minimize the heating effect, the temperature rise caused by heat dissipation from the processor and the LDO is of interest. In the flat form (Fig. 10), the only way for the heat to reach the sensor is via the copper traces. As a significant amount of heat will be conducted by the battery when the FPC is bent around it, it is best to do a simulation, as the temperature at the sensor is expected to increase.

For the purpose of this demonstration, an FEM simulation was set up in Elmer. The UNV format was chosen when exporting the mesh to guarantee a seamless import into Elmer. The pads underneath the LDO regulator and the SoC were configured to act as heat sources with a constant temperature of 30° C and 25° C, respectively. This was done to simulate the waste heat generated from these components. The substrate and copper traces were configured to emit heat with a thermal coefficient of 100 $\mathrm{Wm^{-2}\ K^{-1}}$ to the surrounding air. The ambient temperature and also the initial temperature of each component were set to 20° C. The materials used in this simulation included copper for the traces and pads, and PI for the substrate. As the electric components are conducting heat not as efficiently as copper but more efficiently than the substrate, they were modeled via the steel material for simplicity, just as the CR2032 battery.

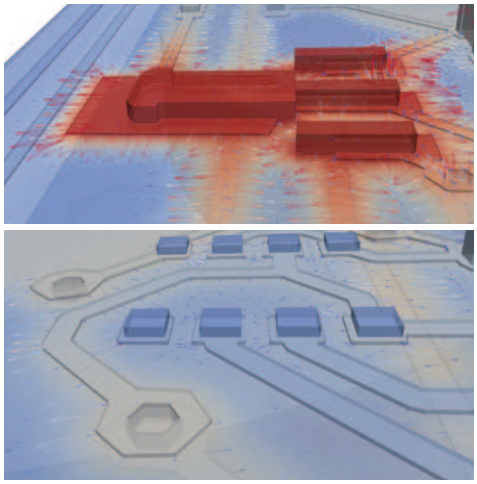

Fig. 11. (Top) Heat flux at the LDO and (bottom) sensor.

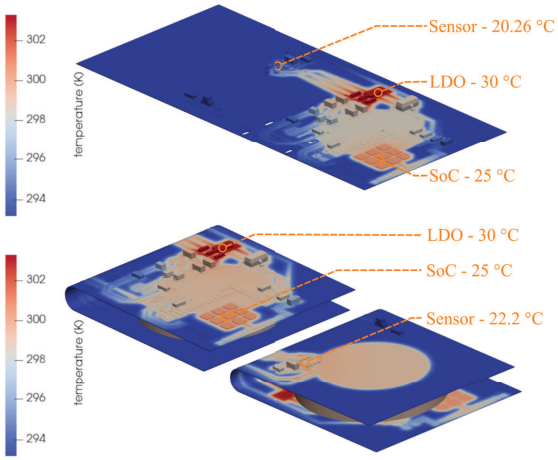

Fig. 12. Thermal FEM simulation — straight board (top) versus bent board (bottom).

Upon completion of the simulation, the `.vtu` file generated by the Elmer solver was then imported into *ParaView* [18] to plot the temperatures on the mesh. To further augment visualization, a gradient operation was created in ParaView to illustrate the vector of the heat flux using arrows with proportional length and direction as depicted in Fig. 11. It turned out that the temperature at the BME280 sensor was 20.3° C in the flat configuration (Fig. 12 top). In the bent configuration with the battery as an additional thermal bridge (see Fig. 12 bottom), the temperature increased by nearly 2K to 22.2° C. The circular pad area heated by the battery was clearly identified as the cause for the additional heating of the sensor.

IV. Conclusion

We presented a comprehensive workflow for designing, modeling, and simulating FPCs using exclusively open-source tools. Our extended tool FTL is free of dependencies on external programs when transforming the 2-D layout of a circuit board into a detailed 3-D model. The latter can be bent according to user-defined parameters. The outcome is available for visualization and simulation purposes, creating powerful possibilities for analyzing the behavior of flexible electronics prior to production.

The integration of a process-dependent renderer enables the utilization of the same 2-D geometry file for the generation of a variety of different models. This approach eliminates the necessity to redefine technology or stackup parameters within any individual model. In addition, it facilitates the generation of multiple different models from a single input geometry, enabling users to execute comparative simulation studies.

By rendering these models in 3-D, users gain a more profound understanding of the final shape of flexible circuits and their interaction with mechanical components, bridging the gap to mechanical CAD programs for further development of casings or mechanical parts. The generated geometries facilitate the creation of accurate models for simulating crucial electrical, thermal, and mechanical parameters. This advancement empowers small companies and open-source projects to design, visualize, and simulate complex flexible circuits, thereby enhancing design accuracy and fostering innovation in FPC development.

Acknowledgment

The authors gratefully appreciate the support of Susann Rothe for useful discussions on this article.

References

[1] N. Arnold, A. Krinke, M. Dietrich, and J. Lienig, "Please, fold the line: Designing flexible electronics using open-source software," in *Proc. IEEE Int. Flexible Electron. Technol. Conf. (IFETC)*, Sep. 2024, pp. 1–3, doi: 10.1109/IFETC61155.2024.10771875.

[2] C. Fu, R. C. Brown, and C. Ume, "Temperature-dependent material characterizations for thin epoxy FR-4/ E-glass woven laminate," in *Proc. IEEE 43rd Electron. Compon. Technol. Conf. (ECTC)*, Jun. 1993, pp. 560–562, doi: 10.1109/ECTC.1993.346789.

[3] M. Wagih, Y. Wei, and S. Beeby, "Flexible 2.4 GHz node for body area networks with a compact high-gain planar antenna," *IEEE Antennas Wireless Propag. Lett.*, vol. 18, pp. 49–53, 2019, doi: 10.1109/LAWP.2018.2880490.

[4] *KiCad*. [Online]. Available: https://www.kicad.org/

[5] Kicad-NIKFEMM. [Online]. Available: https://github.com/nikisalli/Kicad-nikfemm

[6] *Nikfemm*. [Online]. Available: https://github.com/nikisalli/nikfemm

[7] *Kicad_PDN*. [Online]. Available: https://gitlab.com/the_floe/kicad_pdn

[8] *KiCad-sparselizard Branch*. [Online]. Available: https://gitlab.com/kicad/code/kicad/-/merge_requests/1210

[9] *FreeCAD*. [Online]. Available: https://www.freecad.org/

[10] *cad_pcb*. [Online]. Available: https://github.com/realthunder/fcad_pcb

[11] *SALOME Platform*. [Online]. Available: https://www.salome-platform.org/

[12] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *Int. J. Numer. Methods Eng.*, vol. 79, no. 11, pp. 1309–1331, Sep. 2009, doi: 10.1002/nme.2579.

[13] N. Arnold. *Fold the Line*. Accessed: Aug. 21, 2024. [Online]. Available: https://github.com/IFTE-EDA/ftl

[14] *KLayout*. [Online]. Available: https://www.klayout.de/

[15] *Elmer FEM*. Accessed: Aug. 21, 2024. [Online]. Available: https://www.elmerfem.org/

[16] *Palace*. Accessed: Feb. 3, 2025. [Online]. Available: https://github.com/awslabs/palace

[17] T. Liebig. *OpenEMS–Open Electromagnetic Field Solver. General and Theoretical Electrical Engineering (ATE), Univ. Duisburg-Essen*. Accessed: Aug. 21, 2024. [Online]. Available: https://www.openEMS.de

[18] *ParaView*. [Online]. Available: https://www.paraview.org/

Nico Arnold received the M.Sc. (Diploma) degree in electrical engineering from Dresden University of Technology (TU Dresden), Dresden, Germany, in 2022.

He is currently working on the project "HyPerStripes" at the Institute of Electromechanical and Electronic Design (IFTE), TU Dresden. His research field includes FEM simulation and modeling, FPC design, and geometry processing.

Andreas Krinke received the M.Sc. (Diploma) and Ph.D. (Dr.-Ing.) degrees in electrical engineering from Dresden University of Technology (TU Dresden), Dresden, Germany, in 2010 and 2019, respectively.

He currently heads the group for electronic design automation at the Institute of Electromechanical and Electronic Design (IFTE), TU Dresden. His current research interests include the design of integrated circuits and flexible electronics using open-source software.

Manfred Dietrich (Member, IEEE) received the Ph.D. degree from the University of Applied Sciences, Technology, Business and Design, Wismar, Germany, in 1982.

He worked for a variety of companies in the microelectronic industry, including the German Aerospace Center. From 1978 to 1991, he was the Leader of the Group "Development of Design Tools," Halbleiterwerk Frankfurt (Oder), Frankfurt (Oder), Germany. Since 2016, he is an independent Dealer and a Consultant. He founded the company "TexEDA Design GmbH," Frankfurt (Oder), in 2006, where he held a role as the CEO. His competences include modeling of electronic circuits and systems and layout design of integrated circuits.

Dr. Dietrich is a member of various sections of VDE and GI. He was a reviewer of Ph.D. at Technical University in Dresden. He received the Life Achievement Award of the Edacentrum in 2017 and the Award of the Memory Medal of the HTW University Dresden in 2018. He was the General Chair of the IEEE DDECS conference in 2017.

Jens Lienig (Senior Member, IEEE) received the M.Sc. (Diploma), Ph.D. (Dr.-Ing.), and Habilitation degrees in electrical engineering from Dresden University of Technology (TU Dresden), Dresden, Germany, in 1988, 1991, and 1996, respectively.

He is currently a Full Professor of electrical engineering at TU Dresden, where he is also the Director of the Institute of Electromechanical and Electronic Design (IFTE). From 1999 to 2002, he worked as the Tool Manager at Robert Bosch GmbH, Reutlingen, Germany, and from 1996 to 1999, he was with Tanner Research Inc., Pasadena, CA, USA. From 1994 to 1996, he was a Visiting Assistant Professor with the Department of Computer Science, University of Virginia, Charlottesville, VA, USA, and from 1991 to 1994, a Post-Doctoral Fellow at Concordia University, Montreal, QC, Canada. He also authored eight books on EDA and physical design topics published by Springer. His current research interests are in physical design automation, with a special emphasis on electromigration avoidance, 3-D design, and constraint-driven design methodologies of analog circuits.

Prof. Lienig was the General Chair of ISPD 2021. He has served on the Technical Program Committees for the DATE, SLIP and ISPD conferences.

Please cite as: N. Arnold, A. Krinke, M. Dietrich and J. Lienig, "An Open-Source Tool for FEM Modeling of Bent Flexible Circuits," in IEEE Journal on Flexible Electronics, vol. 4, no. 8, pp. 333-341, Aug. 2025