Topology-Aware Routing of 3D-Printed Circuits

Florens Wasserfall, Norman Hendrich, Daniel Ahlers, Jianwei Zhang

Department of Informatics, University of Hamburg, Germany

Abstract

Hybrid manufacturing of 3D-printed parts with integrated electronics has made significant progress in recent years. The general idea of depositing conductive material and electronic components during or after the additive manufacturing step was successfully demonstrated for different process types. However, efficient 3D arrangement and wire routing of circuits for seamless integration into the printing process requires the support of appropriate design software. In this paper, we introduce an approach to integrate the wire routing into the slicing software for Fused Filament Fabrication (FFF). This allows us to consider process parameters for each specific print, e.g. extrusion width, layer thickness or shell thickness. The volumetric object model is first converted into a graph data structure which is then used to apply a routing algorithm. The resulting G-code with embedded wires can be directly executed by the printer. The G-code also includes instructions for a pick and place system to automatically produce the entire object, including electronic components, without operator intervention.

Keywords: Hybrid Manufacturing, Printed Electronics, Wire Routing

1. Introduction

Additive manufacturing of complex objects with additional functionality beyond the rigid physical shape, currently receives an increasing amount of active research [1]. Several approaches towards integration of electronics into additive manufacturing processes have been proposed over the past two decades. A recent survey is provided in [2]. First experiments with low-temperature eutectic alloy [3] and silver filled polymer ink [4] as conductive material were conducted as early as 2004. Direct Write (DW) application of conductive ink or paste inside or at the surface of additively manufactured objects is a very common technique to create electric contacts [5, 6, 7, 8, 9]. Other approaches include direct embedding of wires [10, 11], aerosol jetting [12, 13] and inkjet application of low viscosity conductive inks [14, 15].

The additive electronics manufacturing technology is maturing and increasingly adopted by industry vendors. The Harvard-based startup Voxel8 [16] started distribution of a low-cost printer, combining FFF with a pneumatic ink dispenser in early 2015 but ceased development and support of the printer in 2017. A similar technological approach is taken by nScrypt [17] and Neotech AMT [18]. Both are developing FFF printers with ink dispensers and pick and place capability, based on high-grade mechanical components, aiming at professional, industrial applications.

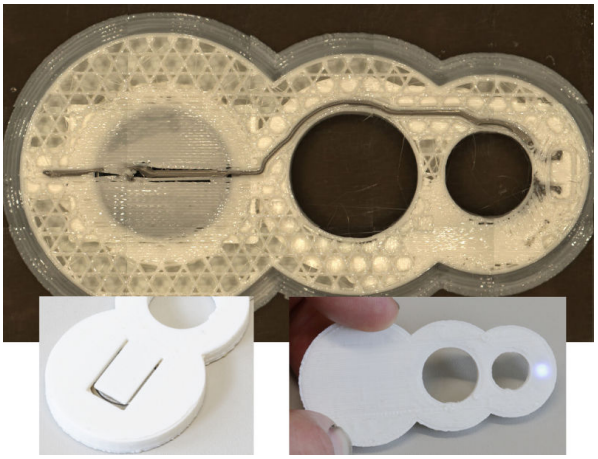

Figure 1: Printed wire, connecting a battery (left) to an LED (right, not inserted yet), following the contour over multiple layers without interrupting the shell. The main image was taken during the printing process and shows an internal layer, final result at the bottom.

Email address: wasserfall@informatik.uni-hamburg.de (Florens Wasserfall, Norman Hendrich, Daniel Ahlers, Jianwei Zhang)

For complex applications beyond the stage of a technology demonstrator, design- and routing-software becomes increasingly relevant to aid the rapid development of new parts. So far, most of the printed demonstrator objects have been carefully designed and prepared with a substantial amount of manual work, which is not a feasible option for the broader commercial application of additively fabricated electronics. For both, mechanical (mCAD) and electrical (eCAD) design software, solutions are well understood and established. However, the integration of an existing schematic into a 3D-body is a mostly unsolved challenge. This involves positioning of components inside or at the surface of a model and wiring and routing of electric connections, both adjusted to the specifics of a given additive process and the individual geometry of each model.

A straightforward but very labor-intensive solution is the manual design of two solid models: one volume for the structural plastic and another representing the conductive material. Channels and cavities must be explicitly modeled, making subsequent redesigns time-consuming. A semantic model of the circuit is not supported. This approach is often used to demonstrate the feasibility of a printing technique, e.g. in [19] or in our own work [20].

MacDonald et al. used a common PCB layout software for aided design and routing of a planar circuit which was then wrapped around an object [21]. While this approach allows to reuse established algorithms and design methods, it is bound to the surface of simple geometric objects. Placing of components inside of the material is not possible. As the circuit is modeled as a 2D plane, collision avoidance by routing wires on different layers inside of the printed object is also not supported.

In 2015 Autodesk and Voxel8 offered Project Wire [22] as a web-based design and slicing tool, tailored for the Voxel8 printer. Only predefined components were supported. Conductive traces were represented as a series of boxes, their thickness matching the layer thickness of the object, and generated by a sequence of mouse clicks. The final design was exported as a multi-material model, represented by two tessellated objects. It was then converted into G-code by a custom slicing tool which translated the conductive extrusions into combined axis and PWM commands to control the pressure-driven ink dispenser. The service was shut down at the end of 2017.

Baily et al. proposed a concept to integrate component placement and wire routing for the wire embedding process into both, the mCAD and slicing software [23]. They use Dassault's SolidWorks [24] for CAD modeling and Ultimakers Cura [25] as slicing tool. Both were extended with custom plug-ins to provide the additional functionality. The electronics specification is imported from an EAGLE [26] schematic into SolidWorks, where 3D representations of the components are rendered and cut out from the object to form cavities. Electrical connections are then routed by creating a "3D sketch" and exported into an auxiliary DXF file. The generation of channels is not required, as the wires are thermally submerged into the plastic surface. In a second step, the object is imported into the Cura software, where a second plug-in generates the trajectories for wire embedding from the DXF file. Insertion of components and joining of wires are executed manually. The approach is currently limited to planar circuits within a single layer.

Carranza et al. [27] introduced a routing solution based on the open source 3D-animation software Blender. Similar to our approach, their extension can import existing netlists and component geometries from dedicated electronic design software. Components are manually placed at arbitrary positions and orientations in the object model. Blender's spline implementation is utilized to generate wire paths by manually manipulating the spline parameters for each wire. The result is exported as a multi-material STL, one mesh representing plastic, the other conductive material. They used a nScrypt printer to successfully fabricate a 555 timer circuit. All components had to be placed manually in a postprocessing step and are therefore arranged at the surface. While this approach mostly attempts to solve the same design challenges as our work, it does not take into account the process parameters of the manufacturing system. The authors state that they had to follow certain design rules, particularly respecting the extrusion-width to achieve printable designs. Furthermore, automated routing is currently not possible.

Ankenbrand et al. [28] used a fully integrated 5-axis system, combining an FFF extruder for structural material, a piezo jet dispenser for contactless application of conductive ink, and a vacuum gripper for camera assisted pick and placing of electronic components. The additional degrees of freedom allow to print wires along free-form surfaces and place components in arbitrary rotations. To control the system, they integrated slicing algorithms into a commercial CAM processor. The model can be split into several subvolumes with different build directions. Cavities are created upon positioning of a component and wires are defined manually by point to point trajectories.

In this study, we investigate the integration and routing of electronic circuits, particularly for 3-axis FFF machines. We propose to integrate the mCAD and

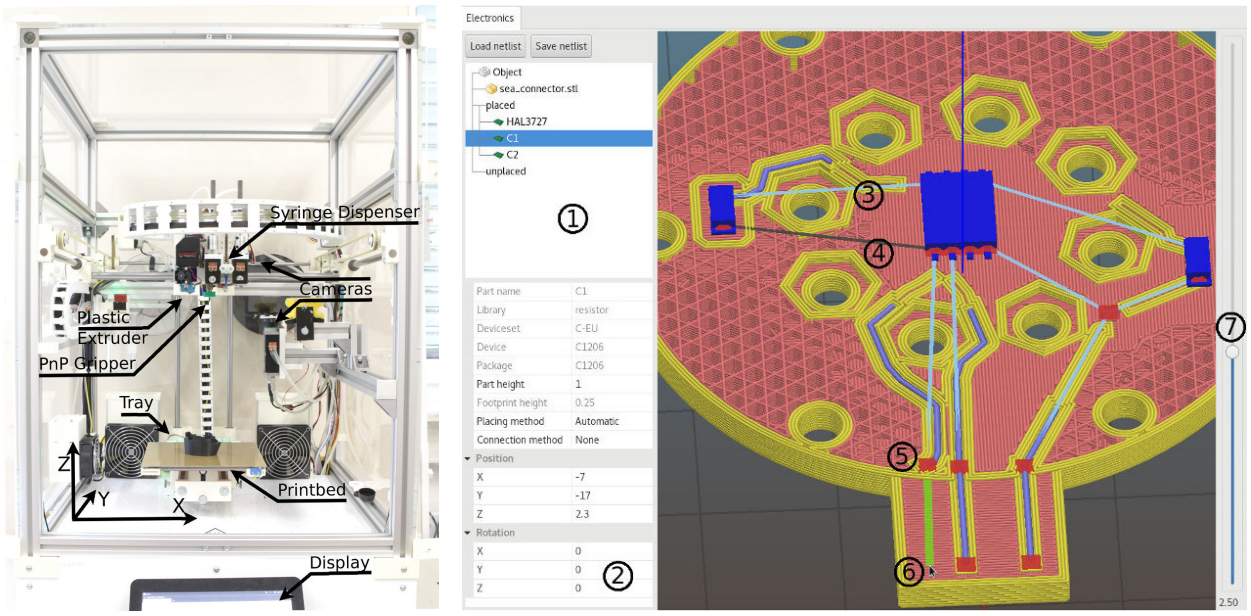

Figure 2: **Left**: modified 3D printer with cameras, syringe driven dispenser for conductive paste and vacuum gripper for automatic component placing. **Right**: screenshot of the augmented slicing software for 3D-electronics design. The sidebar (1) contains a list of all components and their properties (2). Routed (3) and unrouted (4) connections are represented by rubberbands. Waypoint (5) based rubberbands are created with drag and drop (6). A slider on the right (7) allows browsing through the object layers.

eCAD models in the slicing software, where layer thickness, extrusion widths and other important process parameters are known for each particular machine and print-job. Known parameters allow the slicer to create channels with a diameter matching the extrusion width of the conductive material, generate solid, aligned surfaces under wires, or to align adjacent wires with distances chosen as a multiple of the plastic extrusion width. The 3D circuit layout is stored in a separate file. This way, both the 3D model and the schematic can be modified later and the routing algorithm automatically adopts existing connections to the new object shape. The fundamental idea to first slicing an object and then converting the outline of all layers into a graph representation that is suitable for the application of search algorithms was introduced in our previous work [29]. This article is based on a dissertation [30], particularly on the advances described in chapter 5.

The rest of this paper is structured as follows. Section 2 reviews our previous work, including the hardware of the printer we developed to conduct experiments and the basic concepts of our 3D-design- and routing-software, which is integrated into a slicer. In section 3, the transformation of the tessellated object model into a graph representation is described as a prerequisite for the routing step. The execution of the routing algorithm is explained in section 4. During the search, the graph is dynamically expanded with a grid and z-connections in regions where the wire is likely to be placed. A selection of different applications and printed objects is presented in section 5, along with a discussion of the results. The paper concludes with an outlook on future work.

2. Own Previous Work

2.1. *Manufacturing System*

Figure 2 shows the modified, open source FFF 3D printer which we used to develop and test our algorithms. The printer is based on a Kühling & Kühling HT500 system [31]. It consists of a 3-axis gantry system, where the x and y axis are mounted at a fixed height, the printbed serves as z-axis. The carriage is equipped with:

- a standard Bondtech extruder for plastic filament,
- a screw-driven syringe extruder for conductive paste,
- a pivot-mounted vacuum nozzle for pick and place handling of SMD-components,
- an industrial camera (Basler acA2500-14gc).

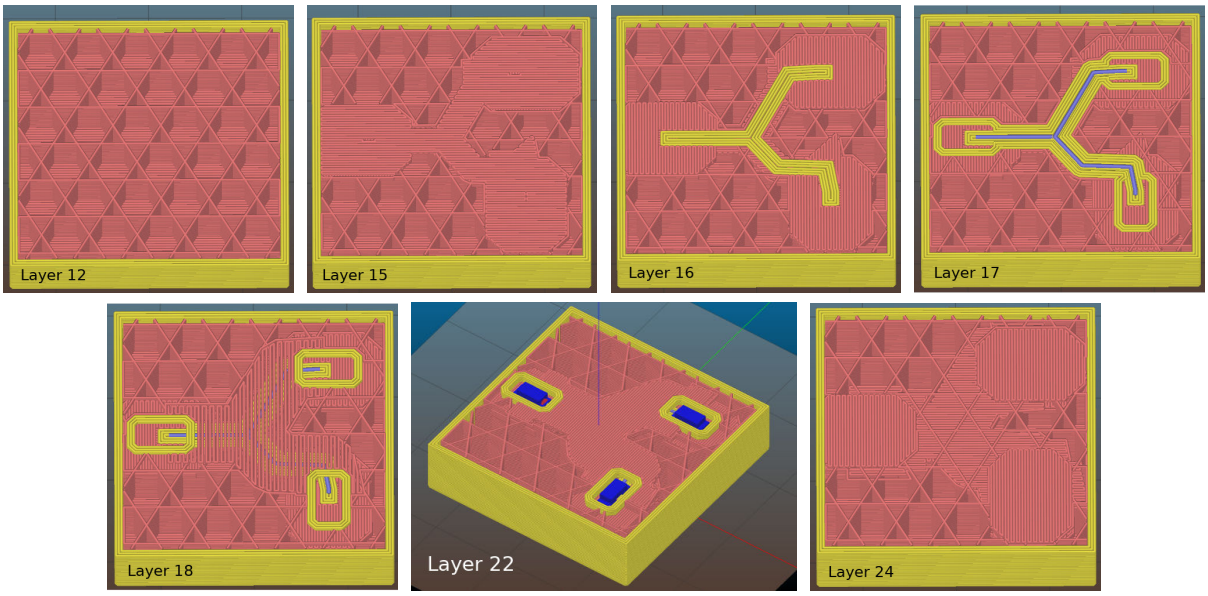

Figure 3: Automatic generation of a tool path with integrated electronic components and wires, illustrating the steps for channel and cavity generation. **Layer 12**: Internal layers with sparse infill (density 20%). **Layer 15**: Several layers of solid infill to create a closed surface for wires and components. **Layer 16**: Layer directly below wires and components. The internal perimeters (yellow) serve as a smooth "substrate" for the wires, aligned with the extrusion path of the conductive material. **Layer 17**: Channels and cavities are generated for the extrusion of wires and placing of components. **Layer 18**: Channels are covered by a region of solid infill. **Layer 22**: Placing of components. **Layer 24**: Components are also covered by a region of solid infill, providing reliable mechanical adherence.

The conductive paste extruder and vacuum gripper are mounted on individual, servo-controlled, micro-z-stages and can be positioned below or above the plastic extruder to avoid collisions. We use consumer grade PLA and ABS materials for the plastic part and a silver filled polymer ink (#6130F, Methode Electronics Inc) to print the wires. The conductive ink is thermally cured, partly by physical contact with hot plastic, with an additional curing phase in the heated print-chamber after the print is finished.

A second camera (bed camera) is attached to the printer's frame, next to the printbed, facing upwards. The cameras are used to align components during the pick and place process [20], calibrate the positions of tools, and to document the build process [32].

A component tray, consisting of several small boxes, is mounted to the printbed. SMD-components are prepared by an operator prior to the print-process. Each box holds exactly one component. The exact position of the component inside a box is determined by the head camera before each pick operation. This approach is very flexible for prototyping, where each printed object requires a different set of components.

OctoPrint [33] is used as a printserver to control the hardware. We implemented OctoPNP [34] as an OctoPrint plugin to integrate vision-based pick and place operations into the normal printing process. The shape and destination position of each component is encoded with a custom extension into the G-code format. OctoPNP extracts this additional information upon loading of a G-code file and assigns a component tray box for each component.

2.2. 3D-Electronics Design Software

The general concept of our software to integrate 3D-electronics design and routing was introduced in [35]. It is a modified version of the open source slicing software Slic3r. The processing of physical models into a sequence of machine movements (G-code) is the core functionality of a slicer. Our extension allows to import EAGLE schematics as eCAD models.

Figure 2 shows how the current state of a design is rendered as G-code preview by the user interface. The user can place components at different layers of the sliced object via drag and drop, similar to the layout step for a PCB board, by vertically "browsing" through the object. Components can be set to arbitrary positions and orientations, however, our pick and place system is currently constrained to rotations in the vertical z-axis, therefore support for wire generation after rotation in x

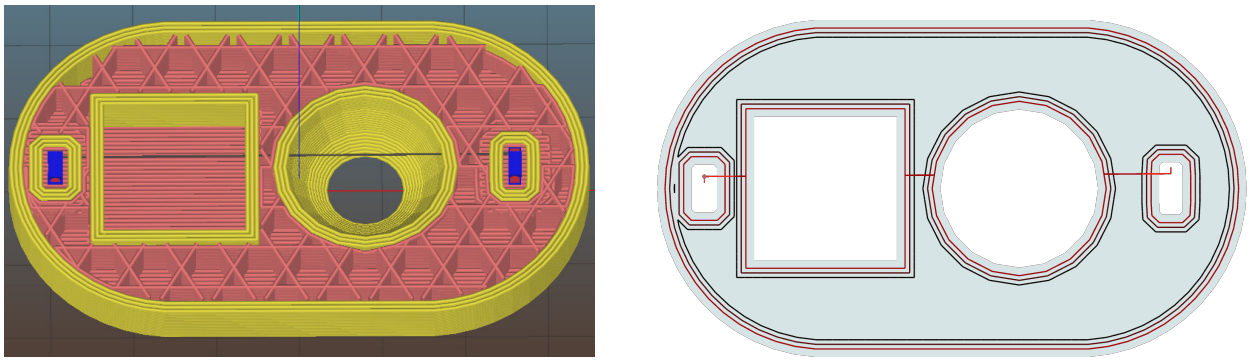

Figure 4: Simple case of a single connection within one layer. The graph representation (right) of the current layer is initialized by offsetting the contour polygons and the intersection of the rubberband with infill regions. The weight of the direct connection derived from the rubberband is equal to the most outer perimeter. The same weight value is later also used for grid edges.

or y is not fully implemented yet. Automatic positioning of components is considered an interesting topic for future research, but in many cases the exact 3D-position of most components is determined by the application (e.g. for sensors or LED interfaces). The netlist for each imported schematic is internally represented by a graph data-structure to model semantic relations. Unrouted connections between placed components are visualized as rubberbands. High-level routing is achieved by manually setting waypoints, serving as interpolation points for the low-level routing to generate extrusion paths.

Once the positions of components and additional wire waypoints are set, the software automatically generates cavities for components, channels for wires, and appropriate extrusion trajectories for conductive material. Cavities and channels are inserted into the solid object by computing the hull polygon from the component outline and extrusion width of the conductive material extruder for each affected layer, and subtracting this hull polygon from the layer contour. To reduce the risk of defects caused by the roughness of printed surfaces, an aligned *bed* is generated below every wire as illustrated in Figure 3, Layer 16. This is achieved by opening a very narrow gap in the contour of layer L_{-1}, causing the generation of a perimeter around the gap which is precisely aligned with the wire. Regions below and above the integrated electronics are marked as *internal solid* surfaces and printed with high material density to form a solid enclosure in objects with sparse infill.

3. Data Representation

The steps described in this section are a preliminary requirement for the actual routing algorithm, but it is safe to first read section 4 for a general understanding of the concept.

The application of routing algorithms requires a proper representation of the object shape and certain relevant process parameters. To achieve this, the original tessellated volumetric model is converted into a 3D graph representation $G = (V, E)$ with weighed edges: $w : E \rightarrow \mathbb{R}_0^+$. All coordinates, weights and distances are scaled by a factor of 10^6 and represented by integer values to avoid accuracy problems caused by floating point approximations. This is the default behavior of the underlying Slic3r implementation and was continued in our extension. On a 32 bit architecture, this covers a workspace of 4294 mm with 1 nm resolution.
In a first step, a common slicing algorithm is applied to split the volumetric object into a stack of 2D layer surfaces, each represented by a set of nested polygons. Section 3.2 describes how the resulting polygons are refined in a second step to remove redundancy and guarantee sufficient spatial resolution for the routing algorithm.

The printed contour of a given Layer L_i is generated by a set of n_p perimeter loops, forming the solid shell of the object. Within this shell, which is filled with a sparse infill structure, wires can be routed arbitrarily. Figure 4 illustrates how the graph is initialized based on the outline of each individual layer in a third step. Conductive traces should follow, but not interrupt perimeter extrusions if possible. Contour-aligned edges are therefore computed by offsetting the layer polygons inwards. Given the number n_p and width δ_p of perimeter extrusions, the offset values Δ are determined by:

$$\Delta_k = k \cdot \delta_p + \frac{\delta_c + \gamma}{2}, \quad k \in \{0 \,..\, n_p\} \tag{1}$$

where δ_c is the extrusion width of the conductive material and γ is the amount of extra space to form a channel around the wire.

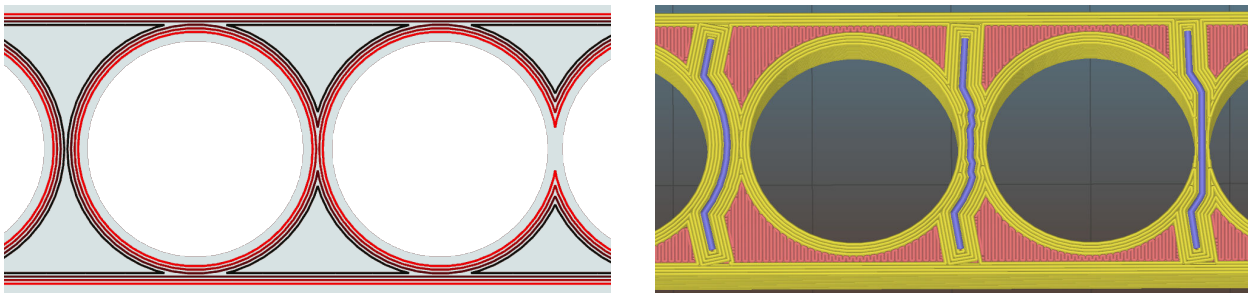

Figure 5: Effect of increasing edge weights towards the object surface. Wires will follow the contour in regions with sufficient space (left) and preserve as many extrusions as possible at narrow passages.

The edge weight w is set to a neutral value of 1.0 for the most inner edges and gradually increases towards the boundary:

$$w_k = 1.0 + (n_p - k) \cdot \alpha \tag{2}$$

where α can be configured in the user interface with the default value set to 0.1. Figure 5 illustrates how the progressive weighting scheme confines conductive traces to stay inside the object hull but still allows thinning of the plastic material in regions with insufficient space.

To assemble a full 3D-representation, contour edges are inserted to the graph for each layer, resulting in a stack of unconnected sub-graphs as initial data structure for the routing algorithm outlined in section 4.

In a final step, a straight connection is established for the wire which is about to be routed by inserting a set of edges along the rubberband representing this wire. The weight of such shortcut edges is set to $w = 1.0 + n_p \cdot \alpha$ which is equal to the weight of the outer perimeter edges. The direct connection can be omitted to enforce grid-aligned wire trajectories in infill regions.

3.1. Spatial Indices

The graph data-structure is complemented by several indices, organized in a layer-indexed list.

For efficient spatial relation queries, an R-tree [36] based index (*spatial-index*) of all vertices is maintained. The routing algorithm depicted in section 4 often tests whether a vertex already exists at or close to a given coordinate. Slight deviations introduced during the slicing and polygon offsetting steps frequently result in very similar, but not identical points. Therefore, two points with a distance below a certain threshold ε are considered to be coincident. Without a spatial index for simple lookup operations ("*is there a corresponding point on the next layer?*") all vertices of G must be checked for ε-coincidence individually by iterating the entire graph. A `nearest`-query is used to find the closest point which is then tested for ε-coincidence in a second step. Finding all points within a bounding box can be achieved by a `within`-query. This is later required to connect a new grid vertex with all nearby existing vertices.

The routing algorithm also frequently tests whether newly generated vertices are located inside the object's hull. To accelerate this, a supplementary stack of *deflated slices* is pre-computed. Each slice consists of a

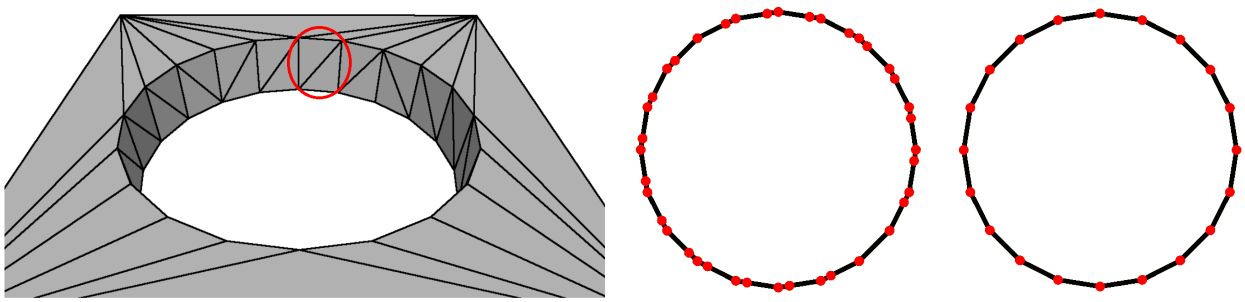

Figure 6: Removing redundant collinear points. **Left**: tessellated representation of a typical object with many rectangular surfaces, each composed of 2 or more triangles. **Center**: resulting polygon for one layer in the slicing process, containing a number of collinear points from the additional triangle edges (red marker in 2). **Right**: same polygon after collinear points were removed.

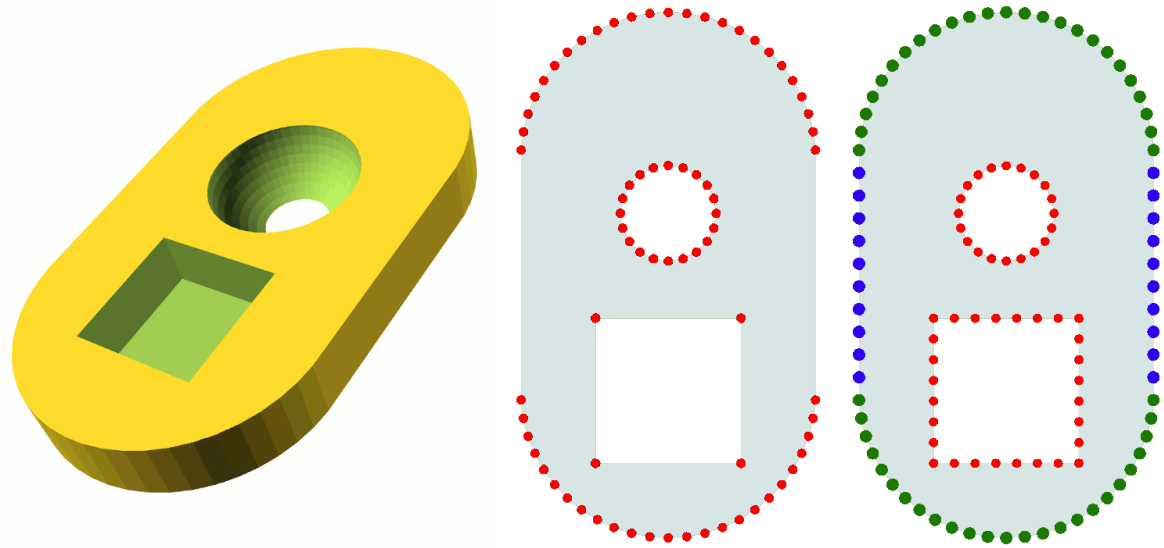

Figure 7: Alignment and upsampling of contour polygon points to guarantee maximum edge length constraint for inter-layer connections. The contour of a single layer (**center**) is compared pointwise against all points of the previous layer. **Right**: matching points with slight deviations are aligned (green), points matching the contour are inserted (blue). In a final step, additional points are inserted where the distance between two points is higher than `interlayer_overlap` (at the rectangular cutout).

polygon, containing only the infill area of one layer. To test a vertex, the corresponding deflated slice is selected by the z-coordinate and it is computed whether the $[x, y]$ coordinate is contained by this polygon.

3.2. Removing Redundancy

The triangle-based representation of tessellated surfaces often causes undesirable redundant information during the slicing step. Figure 6 (left) illustrates a typical example where planar surfaces are represented by a combination of two or more triangles. The contour polygons for a given layer are computed by intersecting a horizontal plane with all triangle edges. For the highlighted rectangular surface (red circle), this results in an additional vertex with a slightly different position at each layer (Fig.6 (center)). Such a redundant point would interrupt the search for matching inter-layer connections described in section 4.2, since no corresponding point exists at the adjacent layers.
Since all redundant points lie on a straight line between two actual boundary points of the considered surface, it is sufficient to remove collinear points. For each consecutive set of 3 points p_1, p_2, p_3, the distance between the mid point p_2 and the line $[p_1, p_3]$ is computed. If the distance is below a threshold ε, p_2 is skipped and p_1 is tested against the next pair of points. The result is depicted in Fig. 6 (right).

3.3. Upsampling

In addition to the problem of redundant collinear points, a low density of points on the perimeter polygons also poses an issue. The distance between each two adjacent points p_i and p_{i+1} must be smaller than the `interlayer_overlap` (ω). Otherwise the routing algorithm described in section 4 would not be able to connect new vertices in infill areas to the existing perimeter vertices. The search for an inter-layer connection would also fail, as the first back-traversal step would already exceed the maximum allowed overlapping length. To solve this problem, we insert additional (redundant) points into the contour polygons to achieve sufficient density. This step is executed right after the object is sliced into the layer representation prior to all offsetting operations, resulting in a graph where the length of each edge is guaranteed to be below ω.

If possible, additional points should be aligned with corresponding points on adjacent layers. For a given Layer L_i all polygon points are compared to the points in the previous layer L_{i-1}. In a first step, coincident points are aligned and points occurring in L_{i-1} but not in L_i are projected upwards (green and blue points in fig. 7). In a second step points are inserted at remaining long edges, e.g. at the rectangular cutout in fig. 7 which did not exist in the previous layer. Typically, adjacent contours are very similar and the point density is sufficient after the projection step, the final upsampling occurs only a few times for each object at strong contour changes.

4. Routing

The 3D position and rotation of electronic components within the object coordinate system is manually set by the user and implicitly determines the position of contact pads. The connection between two pads is derived from the netlist as direct connection (rubberband). A rubberband can be split into multiple segments by waypoints, to define a coarse wire trajectory. SMD-pads are treated as waypoints by the algorithm.

The routing problem is defined as finding the optimal, collision-free path, connecting two points A and B which is printable for a given object geometry and printer configuration. A and B are two adjacent waypoints, connected by a rubberband. The routing step is executed iteratively on all wire segments. Collisions with previously routed wires are implicitly avoided since the channels are treated as holes.

4.1. Planar Pathfinding

This section covers the subquestion of routing a wire within a single layer of the object as illustrated in figure 4. The problem is extended to the 3D case in the next section 4.2.

Once the initial graph representation of an object is fully assembled, a modified A* algorithm is applied to route printed wires. A* was preferred over several other possible routing approaches. Alternative algorithms could be purely grid-based with Steiner Rectilinear Minimal Trees (SRMT) or Integer Linear Programming (ILP) or probabilistic attempts e.g. based on Rapidly-exploring Random Trees (RRT). A* works well on graph representations which can encode both, the free-form topology of an object shape and a grid structure for infill regions.

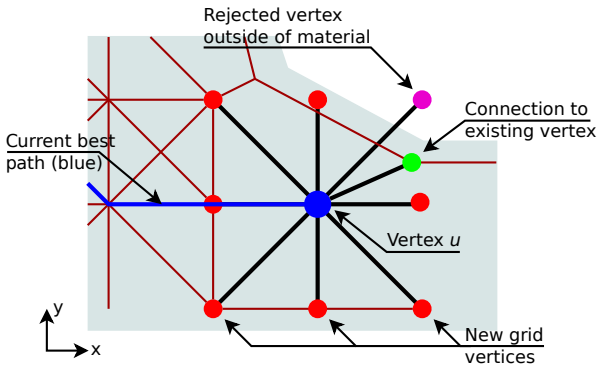

Figure 8: Dynamic generation of new grid vertices. Upon examination of the current vertex (blue), new vertices (red) are generated in 45°steps with a distance of r (`grid_step_distance`). Existing vertices within a radius of r are connected in a second step (green).

```
1  A* (G, s, h)
2      foreach vertex u in V do      // init vertices
3          d[u] := f[u] := ∞;
4          color[u] := WHITE;
5          p[u] := u;
6      end
7      color[s] := GRAY ;  // init start-vertex s
8      d[s] := 0;
9      f[s] :=h(s);
10     Insert(Q, s) ;      // start with vertex s
11     while Q ≠ ∅ do
12         u ← Extract-Min(Q);      // examine u
13         if u == goal then return u;
14         Expand-Grid(u);
15         if Test-Z-Connection(u) then
16             Insert(Q, r)
17         end
18         foreach vertex v in Adj[u] do
19             if w(u, v) + d[u] < d[v] then
20                 d[v] := w(u, v) + d[u];
21                 f[v] := d[v]+ h(v);
22                 p[v] := u;
23                 if color[v] ≠ GRAY then
24                     color[v] := GRAY;
25                     Insert(Q, v)
26                 end
27             end
28         end
29         color[u] := BLACK ;          // finish u
30     end
31 end
```

Algorithm 1: The modified A* algorithm used for searching an optimal wire route in G based on the formulation from the Boost Graph Library documentation [37] which was used for the implementation. In line 14 new grid vertices are inserted to the graph, based on the shape of the layer corresponding to the z-position of u (see section 4.1). The search for suitable inter-layer connections and re-insertion of the predecessor vertex r into the queue (lines 15-17) are explained in section 4.2.

We use a priority queue based implementation with dynamic grid generation in infill areas as outlined in Algorithm 1. In the pseudocode, $d(v)$ is the distance of vertex v from the start vertex s. $f(v)$ denotes the estimated cost of the best path from s over v to the goal vertex t, by combining the distance to v and the estimated distance to the goal: $f(v) = d(v) + h(v)$. A detailed discussion of the heuristic function h is given below in section 4.3. Q is a queue, holding all open vertices, ordered by their f-value. $p(v)$ points to the predecessor of v. The resulting

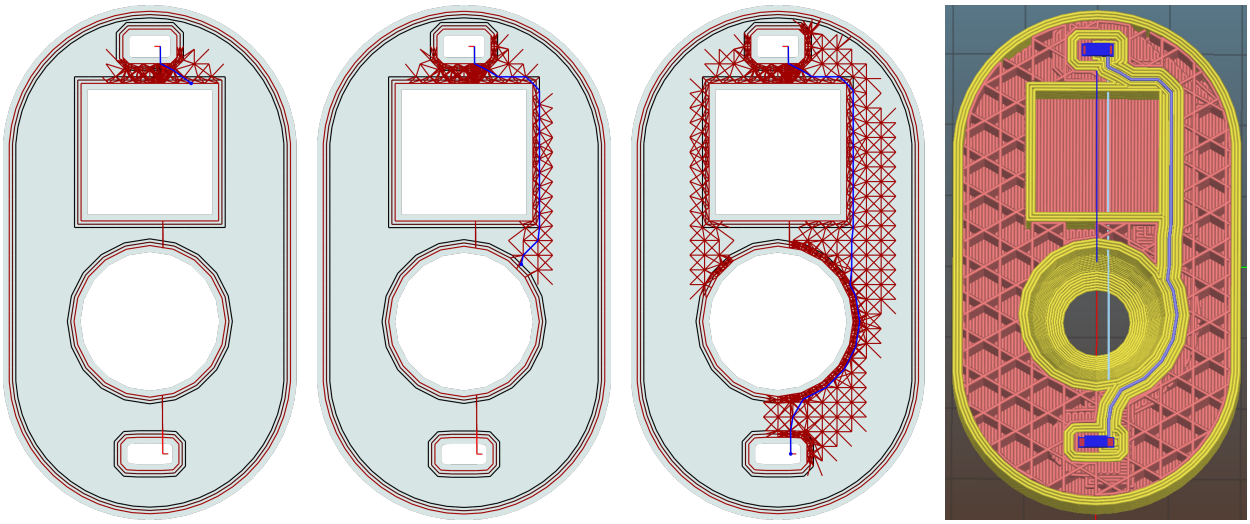

Figure 9: Dynamically growing grid in infill regions after 100 steps (a), 200 steps (b), successful termination (c) and the resulting G-code (d).

path is obtained by traversing the predecessor map from t to s after execution of the search algorithm.

Applied to the initial graph as shown in figure 4, the wire trajectory would have to strictly follow the layer contour. Individual features, i.e. the holes in figure 4, are only connected by the direct edges, derived from the rubberband. Infill areas are generally well suited to contain wires, but require discrete modeling to be used in the graph structure. To achieve this, an octagrid is dynamically generated during the search, with a spacing of `grid_step_distance` ν along the main axis.

Typical values for ν are in the range $[1..2]$ mm. Upon examination of the current vertex u, 8 new vertices v_k are generated in 45° steps around the current vertex during the examination step (line 14 in alg. 1) as illustrated in figure 8. New vertices are tested against the deflated slices dataset and rejected if they are not located within an infill area. In addition, all existing vertices within a distance of one grid step are efficiently selected via the spatial index introduced in section 3, and also connected to u to link up the infill grid with the pre-generated contour vertices. The weight of grid edges is again set to $w = 1.0 + n_p \cdot \alpha$, equal to the weight of the outer perimeter edges.

Figure 9 illustrates how the graph is successively growing during the search, resulting in a smooth wire trajectory.

4.2. Z-Connections

In a 3D circuit, wires should not be constrained to a single layer. Figure 10 illustrates different approaches to physically establish an electric contact, spanning multiple layers. For this study, only the *staircase* scheme was considered, to avoid collision issues, as our printer requires direct surface contact of the dispensing needle. In Figure 12 the example case from Figure 4 is modified, so the rubberband spans several layers.

In the graph representation, such a connection between two adjacent layers consists of a sequence of matching edges on both layers. The minimum length of the overlapping sequence is implemented as an adjustable parameter ω (`interlayer_overlap`). Pre-computing all grid vertices and testing all possible combinations of sequences is computationally not feasible. We therefore exploit the vertex examination step of the A* algorithm to efficiently find suitable

Figure 10: Possible approaches for direct-write wires to connect different layers. **Left**: short, overlapping wire segments forming a *staircase*. **Center**: a hole is printed over several layers and filled with conductive material, resembling a *via*. **Right**: printed *ramp* with open channel, the wire is applied in a nonplanar extrusion. Note that both latter solutions require application of material below the current print surface, bearing the risk of potential collisions between toolhead and printed part.

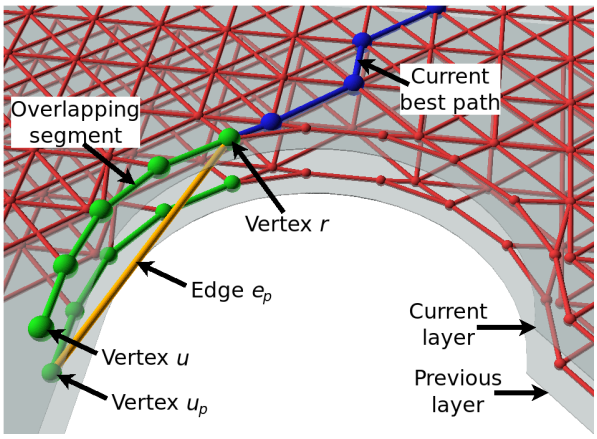

Figure 11: Dynamic generation of inter-layer connections. The current optimal route (blue) is traversed back, starting from vertex u. For all green points on this path, a matching vertex was found on the adjacent layer below. Once this overlapping segment is longer than `interlayer_overlap` ω, a new edge e_p is inserted, connecting both layers.

connections in the growing graph. In Algorithm 1, line 15, the current front vertex u is tested for possible inter-layer connections after new grid vertices are generated but before the relaxation of outgoing edges. The situation is illustrated in figure 11: starting from the front vertex u, the current best path (blue) is traversed back (green) until the sum of distances between all visited predecessor vertices is higher than ω. Each vertex along the path is tested for a corresponding vertex at the layer below. Since grid edges in infill regions of adjacent layers potentially do not exist yet, they are also inserted to the graph at the previous and next layer, according to the scheme depicted in Figure 8. If every point on the current best path has a matching point on an adjacent layer, an inter-layer edge e_p (orange) is added between the first matching point r on the path and the point corresponding to u in the previous layer u_p. Accordingly for the layer above. Traversing only the current best path limits the search for suitable inter-layer connections to sequences along the approximate direction towards the goal.

The weight $w(r, u_p)$ of e_p is set to the sum of the weights of all edges from u to r plus a constant factor Γ:

$$w(r, u_n) = \sum_{k=u}^{r} w(k, p(k)) + \Gamma \qquad (3)$$

where $p(k)$ denotes the predecessor of k. Γ is exposed to the user interface as a parameter to control how much layer changes should be avoided. The factor directly determines how far the algorithm explores horizontally around an obstacle before it considers searching at the next layers. Inter-layer connections tend to be less reliable than extrusions on one layer and require additional tool changes. Depending on the geometry of the individual object and the machine characteristics, layer changes can be either favored or avoided.

The new graph edge e_p potentially shortcuts a longer sequence of vertices, particularly in cases where the wire follows the contour of an object, as it is the case in figure 11. This edge is used by the routing algorithm, but not for toolpath generation. To preserve the original trajectory, all overlapping sections are stored in an external key-value data structure `overlap-map`. Each interlayer-edge is mapped by its vertex-ID to a sequence of 2D-points, representing the overlapping portion of two extrusions at the layer boundary, excluding the z-information (the blue segment of the path). Upon generation of the G-code for the conductive wire extrusion, this sequence is inserted twice, with different z-values for both affected layers, such that the end of an extrusion is repeated on the next layer to close the electric connection. The resulting overlap is highlighted in the overlay in figure 12.

The newly generated edge $e_p(r, u_p)$ is not directly connected to front vertex u and will therefore not be evaluated during the current A* examination step. However, vertex r is already on the closed list (marked as *black*) and will also not be evaluated again, leaving e_p as a "dead branch" in the original formulation of A*. To enforce a re-evaluation, r is inserted into the queue Q again with $f(r) = 0$, pushing r to the front. In the next iteration, r is examined again with u_p being in the adjacency list, correctly appending them to the fringe if their relaxation is successful.

4.3. Heuristic

The A* algorithm uses a distance function to accelerate the search by providing additional information about the general direction to the goal. As the remaining distance is not known in advance, a heuristic $h(u)$ is required to estimate the distance from u to the goal vertex t. A good heuristic utilizes previously available knowledge about the topology of the search domain and can significantly influence the performance. A heuristic is said to be *admissible* if it never overestimates the actual minimal cost between the current vertex u and the goal t. The cost measure is not equal to the length of the path, as the edges have different weights. Slightly longer trajectories along a perimeter are therefore favored over a straight

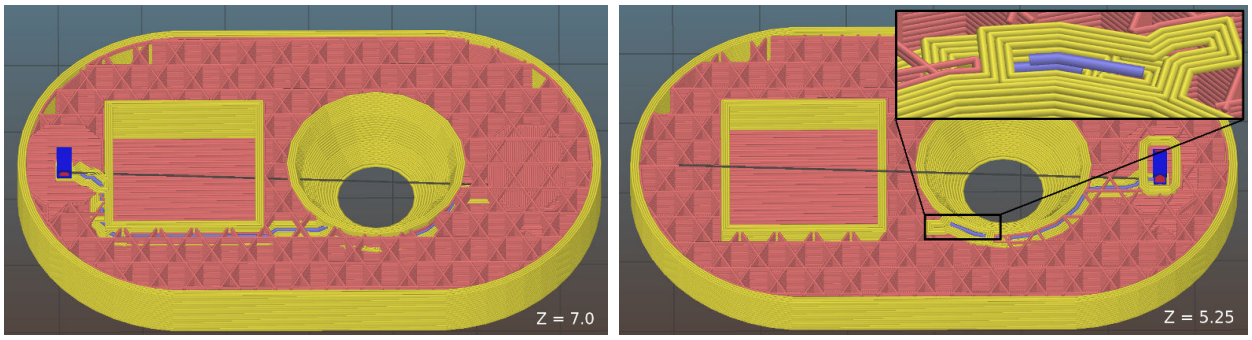

Figure 12: Rendering of a tool path where the wire automatically follows the contour of both an increasingly inclined surface (bowl) and a long straight vertical surface (cube). Several layer changes occur along both structures. Note that the wire is intentionally covered with sparse infill only for transparency visualization. The overlay shows a close-up of one layer-change, viewed from the opposite side.

connection. The cost of the resulting path returned by A* is minimal, if the heuristic also is monotonic. However, even with a monotonic, admissible heuristic, all equally suitable paths with the same distance to the current fringe must be evaluated. In a 3D-object, if the straight path is obstructed by the object's shape, the number of evaluated vertices increases cubically with the difference between path length estimated by the heuristic and actual path length. The search can be accelerated by relaxing the admissibility criterion and slightly overestimating the actual distance. By adding a factor ε ($\varepsilon > 1$) to the heuristic, the search is performed faster, producing a non-optimal result. The result is still guaranteed to have a cost of at most ε times the optimal path [38].

The heuristic in our algorithm is based on the Euclidian distance:

$$h(u) = \sqrt{{d_x}^2 + {d_y}^2 + {d_z}^2} \cdot \varepsilon \tag{4}$$

If start and goal vertex are located on different layers, this heuristic significantly underestimates the remaining z-distance due to the additional constant cost factor Γ which prevents the generation of unnecessary inter-layer connections. For a more accurate estimation of the z-distance, the number of layer changes must be counted (the layer thickness is not fixed and can vary throughout the object). With $\|d_z\|$ denoting the number of layer hops, the actual distance is estimated as:

$$h(u) = \left(\sqrt{{d_x}^2 + {d_y}^2 + {d_z}^2} + \|d_z\| \cdot \Gamma \right) \cdot \varepsilon \tag{5}$$

It is generally preferable to equally distribute layer transitions along the trace and avoid local "stacks" of short extrusions. To support a uniform distribution, the heuristic is further extended with an adaptive factor:

$$z_\alpha = \left| \left(\frac{d_{xy}}{D_{xy}} - \frac{d_z}{D_z} \right) \cdot \nu \right| \tag{6}$$

D_{xy} is the horizontal Euclidean distance between start and goal vertex. d_{xy}/D_{xy} therefore describes the remaining portion of the horizontal distance in the range [0..1], accordingly for the vertical ratio. The difference of both ratios is exactly zero along the direct line between both vertices. In other words: the difference becomes a positive value if the z-distance is too small or too large in relation to the remaining xy-distance. The result is a dimensionless number and must be scaled to match the dimension of the grid. It is therefore multiplied with the `grid_step_distance` ν, so the additional estimated value does not exceed one grid step. The resulting heuristic is implemented for the A* algorithm as:

$$h(u) = \left(\sqrt{{d_x}^2 + {d_y}^2 + {d_z}^2} + \|d_z\| \cdot \Gamma + z_\alpha \right) \cdot \varepsilon \tag{7}$$

5. Results and Discussion

Several objects were sliced and printed to test the ability of the algorithms introduced in this paper to cope with different fundamental situations.

Figure 13 (left) shows an example where the wires are forced to detour over several layers to stay inside the object while at the same time avoiding a collision. Wire collisions are inherently avoided by routing wires sequentially. For each individual net, the routing graph is updated and only regions not yet occupied by previously routed wires are considered. As the result strongly depends on the order in which wires are processed, a heuristic is used to optimize the order of the rubberbands prior to the routing step. The general approach is simple and common in literature:

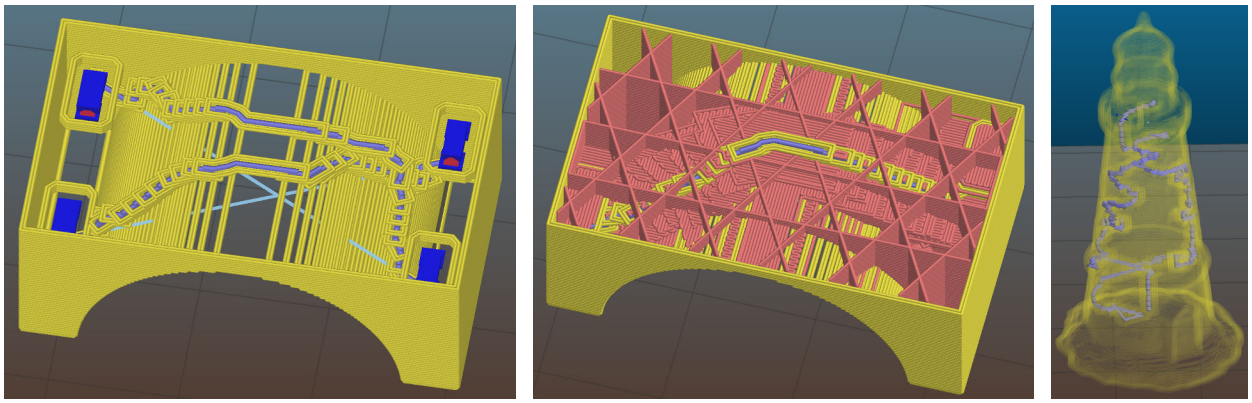

Figure 13: Wire routing over a high vertical distance. **Left**: wires stay inside the object and avoid collisions. Infill material is not rendered. **Center**: sparse infill enabled to demonstrate how the routing preserves the configured shell thickness with a solid bed in all situations. **Right**: Vertical connections pushed to their limits, connecting an LED at the top of the lighthouse.

1. For each net, count the number of collisions with other nets, assuming that all connections are realized as straight lines. Sort the nets ascending by number of collisions.

2. In a second step, order nets with an equal number of collisions by their length, starting with the shortest.

Computing the collision of wires represented by a one-dimensional line is insufficient, as the extrusion and channel around the wires have a certain extent. This is solved by first inflating all other wires to the channel diameter and computing intersections with the inflated polygons. Existing methods from VLSI and PCB routing could be applied to improve the results, e.g. setting predominant directions for some layers or Rip-Up and Reroute.

Figure 13 (right) illustrates the limitations of z-connections in layered manufacturing. While the routing algorithm finds a solution, the required staircase pattern induces a convoluted trajectory. Excessive stacking of short conductive segments causes frequent tool changes and increases the risk of defects. Additional slicing directions or reliable generation of vias spanning higher distances are an important future research topic. Note that both objects shown in figure 13 are non-functional renderings to illustrate the result of our algorithm in specific situations and were not actually printed.

Direct embedding of electronics is particularly useful in situations where a small number of wires need to be integrated into a complex geometry or where components require a particular positioning. For a high number of components and wire density, PCBs are better suited. The "instrumented object" presented in figure 14 is a good illustration for this finding. It is used to record dexterous manipulation movements executed with the human hand in a tracking setup. The position of the fingers, the object itself, and contact forces between finger and object are recorded and generalized into parametric motion primitives to control robotic multi-finger grippers. Optical proximity sensors embedded into the walls of the inner part measure the deformation of the outer hull, which acts as a deformable cantilever. If the mechanical properties of the structure are known, the force inducing the deformation can be computed. The data are collected by a microcontroller and sent to a host computer via Bluetooth connection. The highly integrated circuit of the microcontroller and an additional Inertial Measurement Unit (IMU) for position tracking are provided on a PCB and connected to the printed circuit via the base-plate.

The full 3D integration of a circuit is demonstrated in figure 15. Three LEDs are integrated into the boat as navigation lights, connected to a microcontroller. A few waypoints were manually set at the bow of the boat to achieve a small distance to the components, and two waypoints at the stern for the power connection. The routing algorithm then generated the trajectories. It successfully kept the wires within the material at the walls and avoided collisions with other wires and components.

The graph based routing algorithm currently adds significant additional processing time to the slicing process. A runtime profiling of the C++ code suggests that one single geometric subroutine (`Point::projection_onto`) within the polygon handling library accounts for approximately 90% of the

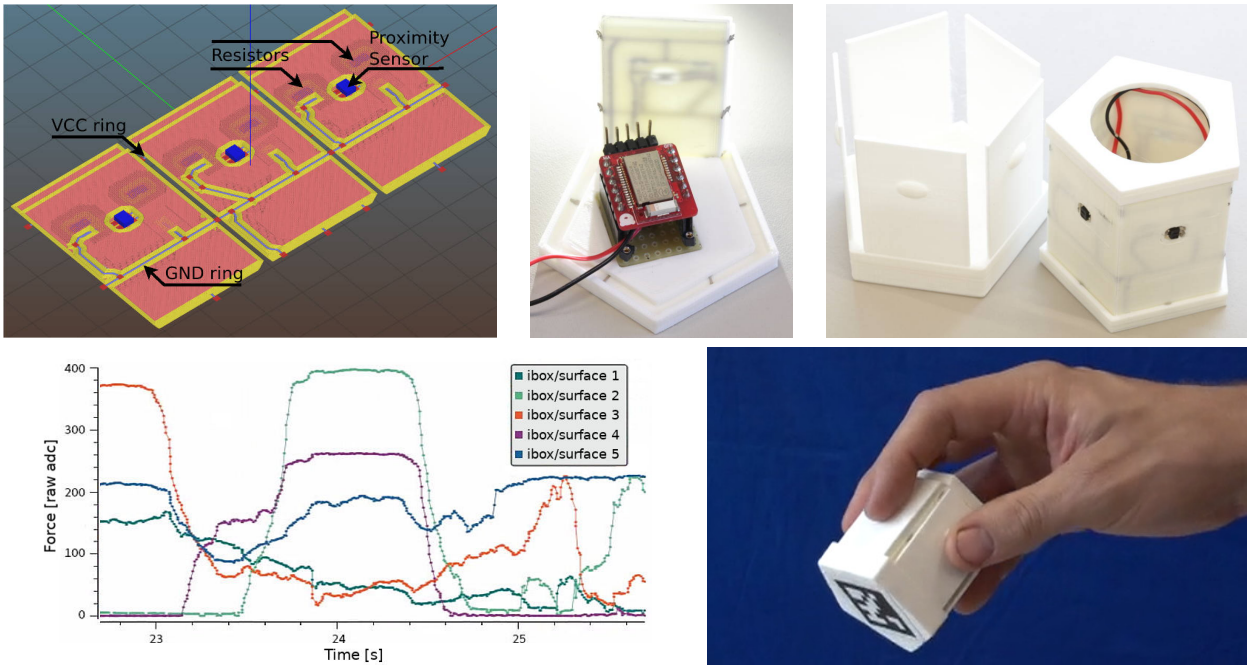

Figure 14: **Top**: force sensitive "instrumented object" with fully 3D-printed integrated electronics. Optical proximity sensors are embedded into the surfaces to measure deformations of the outer hull. The sides of the inner part are printed as a foldable object which is bent into the final form during assembly. **Bottom**: force data recorded during a manipulation task. The position of the object is tracked with a camera, using the visual marker attached to the bottom.

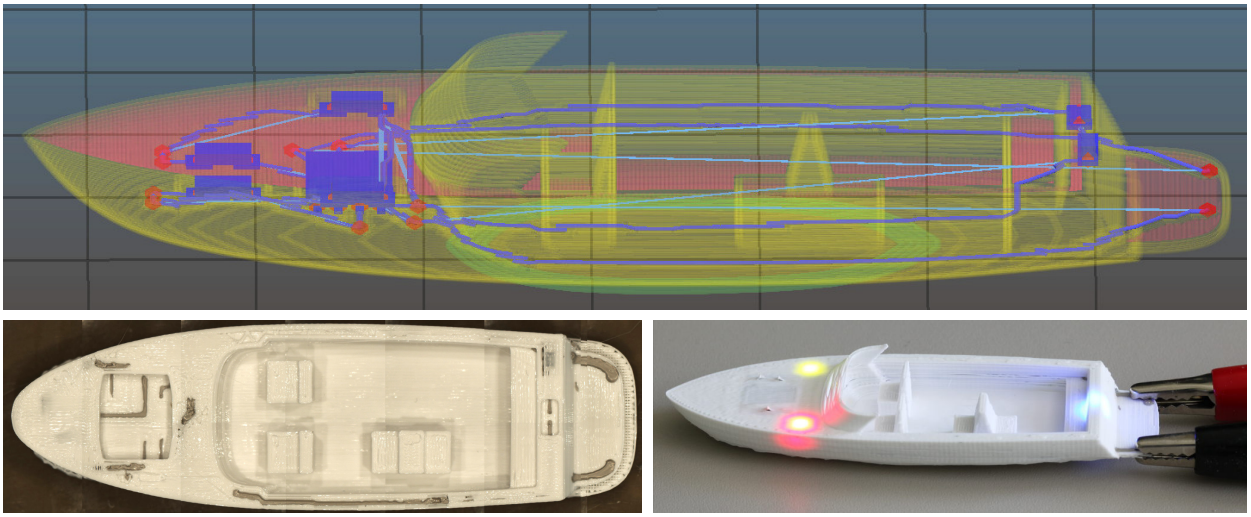

Figure 15: 3D circuit generated with the autorouting algorithm. **Top**: transparent rendering of the G-code with conductive material highlighted in purple. **Left**: Stitched high-resolution image of an intermediate layer, recorded during the printing process. The image shows plastic extrusion, a long wire extrusion following the perimeter and contour of the portside boat hull, and short wire extrusions for the LEDs and microcontroller in the bow. **Right**: Printed boat with blinking LEDs. The pattern is generated by a pre-programmed Attiny85-20 microcontroller. STL-model by Maxlarsen under CC BY 3.0 license.

processing time. This subroutine could potentially be accelerated by optimization or caching. The execution time also depends on the A* graph search parameters and object shape. To estimate the effect we processed our demonstration object shown in figure 4 with a combination of different parameters:

Table 1: Runtime analysis of the routing algorithm for different parameters.

	Grid resolution	Heuristic (ε)	Inter-layer cost (Γ)	Runtime	Path length
Baseline	1.0 mm	1.0	1.5	3.26 s	77.1 mm
Resolution	**0.5 mm**	1.0	1.5	12.92 s	70.3 mm
	1.5 mm			2.35 s	77.2 mm
	2.0 mm			1.28 s	77.5 mm
ε	1.0 mm	**1.1**	1.5	2.87 s	77.1 mm
		1.3		1.27 s	78.2 mm
Γ	1.0 mm	1.0	**0.0**	7.44 s	74.2 mm
			3.0	2.83 s	77.1 mm
			5.0	2.50 s	76.8 mm
Combined	**2.0 mm**	**1.3**	**5.0**	0.60 s	78.7 mm

For each row, the altered parameter is highlighted in bold text. All other parameters are fixed. Reducing the grid resolution decreases the runtime as expected. Increasing the heuristic factor ε yields lower runtimes at the cost of less optimal trajectories. The layer-changing cost Γ also significantly affects the runtime. A low value prefers horizontal exploration, including several adjacent layers while a high value minimizes the number of layer changes for the potential cost of longer in-layer detours. The special case of $\Gamma = 0$ eliminates layer changing costs at all, often resulting in an "undulating" trajectory, jumping up and down between the layers. The differences in the resulting path length reflect situations where the wire "dives" below the cutout (~70 mm) or circumvents the obstacle on a higher layer (~77 mm).

As a reference: slicing only the plastic object without any electronics requires 0.085 s. Compared to the print-time of more than one hour, the computing time to prepare the G-code is negligible in all cases.

6. Conclusion

In this paper we introduced an approach to arrange, route and print electronic circuits in 3D objects. The pre-defined schematic is imported into a slicing software where the position of components and wire waypoints are set in a preview rendering of the G-code. The solid object is then translated into a graph representation, suitable for the application of a routing algorithm, which generates printable extrusion paths for conductive material, well embedded into the structure of the printed object. We successfully demonstrated the approach with several applications and printed objects.

The slicing and routing software described in this paper is available under an AGPLv3 license on Github:

https://github.com/platsch/Slic3r/tree/electronics

Be aware that this is a highly experimental implementation and currently not suitable for production use.

Future research clearly should attempt to overcome the limitations caused by the layered structure. For the 3-axis gantry systems used in this paper this could be achieved by developing solutions for vertical vias as depicted in figure 10. A promising approach is the application of design and routing concepts for 4-/5-axis manufacturing machines. Those configurations are commonly used for machining and allow deposition of material from every angle, facilitating conformal printing of wires along the surface or partitioning of the entire object into multiple subvolumes with different slicing- and wire-routing-directions. The same applies to component handling where rotations are currently only possible around the z-axis. While the basic hardware already exists, collision avoidance and G-code generation for additive processing still pose major challenges.

References

[1] E. MacDonald, R. Wicker, Multiprocess 3D printing for increasing component functionality, Science 353 (2016) aaf2093–aaf2093. doi:10.1126/science.aaf2093.

[2] A. H. Espera, J. R. C. Dizon, Q. Chen, R. C. Advincula, 3D-printing and advanced manufacturing for electronics, Progress in Additive Manufacturing (2019) 245–267. doi:10.1007/s40964-019-00077-7.

[3] E. Sells, A. Bowyer, Rapid prototyped electronic circuits, Tech. rep., University of Bath, Department of Mechanical Engineering (2004).

[4] J. Palmer, P. Yang, D. Davis, B. Chavez, P. Gallegos, R. Wicker, F. Medina, Rapid prototyping of high density circuitry, in: Proceedings of the Rapid Prototyping & Manufacturing Conference, 2004.

[5] E. De Nava, M. Navarrete, A. Lopes, M. Alawneh, M. Contreras, D. Muse, S. Castillo, E. Macdonald, R. Wicker, Three-Dimensional Off-Axis Component Placement and Routing for Electronics Integration using Solid Freeform Fabrication, Proceedings of the 19th International Solid Freeform Fabrication Symposium (2008) 362–369.

[6] K. B. Perez, C. B. Williams, Combining Additive Manufacturing and Direct Write for Integrated Electronics – A Review, Proceedings of the International Solid Freeform Fabrication Symposium (2013) 962–979.

[7] D. Espalin, D. W. Muse, E. MacDonald, R. B. Wicker, 3D printing multifunctionality: Structures with electronics, International Journal of Advanced Manufacturing Technology 72 (2014) 963–978. doi:10.1007/s00170-014-5717-7.

[8] J. T. Muth, D. M. Vogt, R. L. Truby, Y. Mengüç, D. B. Kolesky, R. J. Wood, J. a. Lewis, Embedded 3D printing of strain sensors within highly stretchable elastomers, Advanced Materials (2014) 6307–6312. doi:10.1002/adma.201400334.

[9] T. Wasley, J. Li, D. Ta, J. Shephard, J. Stringer, P. Smith, E. Esenturk, C. Connaughton, R. Kay, Additive Manufacturing of High Resolution Embedded Electronic Systems, Proceedings of the 26th Annual International Solid Freeform Fabrication Symposium (2016) 1838–1855.

[10] J. Bayless, M. Chen, B. Dai, Wire embedding 3D printer (2010). URL https://www.reprap.org/mediawiki/images/2/25/SpoolHead_FinalReport.pdf

[11] C. Kim, a. Cuaron, M. Perez, D. Espalin, E. MacDonald, R. Wicker, Cooperative Fabrication Methodology for Embedding Wire on Curved Surfaces, Proceedings of the International Solid Freeform Fabrication Symposium (2014) 185–196.

[12] M. Hedges, A. B. Marin, 3D Aerosol Jet Printing - Adding Electronics Functionality to RP/RM, Direct Digital Manufacturing Conference.

[13] R. R. Salary, J. P. Lombardi, M. S. Tootooni, R. Donovan, P. K. Rao, P. Borgesen, M. D. Poliks, Computational fluid dynamics modeling and online monitoring of aerosol jet printing process, Journal of Manufacturing Science and Engineering 139 (2) (2017) 021015. doi:10.1115/1.4034591.

[14] J. Ledesma-Fernandez, C. Tuck, R. Hague, High Viscosity Jetting of Conductive and Dielectric Pastes for Printed Electronics, Proceedings of the International Solid Freeform Fabrication Symposium (2015) 40–55.

[15] J. Stringer, T. M. Althagathi, C. C. Tse, V. D. Ta, J. D. Shephard, E. Esenturk, C. Connaughton, T. J. Wasley, J. Li, R. W. Kay, P. J. Smith, Integration of additive manufacturing and inkjet printed electronics: a potential route to parts with embedded multifunctionality, Manufacturing Review 3. doi:10.1051/mfreview/2016011.

[16] Voxel8 Inc., Voxel8 company website, accessed: March 2020. URL https://www.voxel8.com

[17] Nscrypt Inc., nscrypt company website, accessed: March 2020. URL https://www.nscrypt.com/printed-electronics/

[18] Neotech AMT GmbH, Neotech AMT company website, accessed: March 2020. URL http://www.neotech-amt.com

[19] J. Li, Y. Wang, G. Xiang, H. Liu, J. He, Hybrid Additive Manufacturing Method for Selective Plating of Freeform Circuitry on 3D Printed Plastic Structure, Advanced Materials Technologies 4 (2) (2019) 1–10. doi:10.1002/admt.201800529.

[20] F. Wasserfall, Embedding of SMD populated circuits into FDM printed objects, Proceedings of the 26th International Solid Freeform Fabrication Symposium (2015) 180–189.

[21] E. MacDonald, R. Salas, D. Espalin, M. Perez, E. Aguilera, D. Muse, R. B. Wicker, 3D printing for the rapid prototyping of structural electronics, IEEE Access (2014) 234–242. doi:10.1109/ACCESS.2014.2311810.

[22] Autodesk Project Wire, accessed: March 2020. URL https://academy.autodesk.com/inspiration/3d-printed-electronics-open-new-doors/

[23] C. Bailey, E. Aguilera, D. Espalin, J. Motta, A. Fernandez, M. A. Perez, C. Dibiasio, D. Pryputniewicz, E. MacDonald, R. B. Wicker, Augmenting computer-aided design software with multi-functional capabilities to automate multi-process additive manufacturing, IEEE Access 6 (2017) 1985–1994. doi:10.1109/ACCESS.2017.2781249.

[24] Dassault Systèmes, SolidWorks, accessed: March 2020. URL https://www.solidworks.com

[25] Ultimaker BV, Cura Slicing Software, accessed: March 2020. URL https://ultimaker.com/software/ultimaker-cura/

[26] Autodesk Inc, EAGLE product website, accessed: March 2020. URL https://www.autodesk.com/products/eagle/

[27] G. T. Carranza, U. Robles, C. L. Valle, J. J. Gutierrez, R. C. Rumpf, Design and Hybrid Additive Manufacturing of 3-D/Volumetric Electrical Circuits, IEEE Transactions on Components, Packaging and Manufacturing Technology 9 (2019) 1176–1183. doi:10.1109/TCPMT.2019.2892389.

[28] M. Ankenbrand, Y. Eiche, J. Franke, Programming and Evaluation of a Multi-Axis/Multi-Process Manufacturing System for Mechatronic Integrated Devices, 2019 International Conference on Electronics Packaging (ICEP) (2019) 273–278.

[29] F. Wasserfall, Topology-Aware Routing of Electric Wires in FDM-Printed Objects, Proceedings of the 29th International Solid Freeform Fabrication Symposium (2018) 1649–1659.

[30] F. Wasserfall, Integration of Conductive Materials and SMD-Components into the FDM Printing Process for Direct Embedding of Electronic Circuits, Ph.D. thesis, University of Hamburg (2020). URL http://ediss.sub.uni-hamburg.de/volltexte/2020/10259/

[31] Kühling&Kühling GmbH, HT500 product website, accessed: March 2020. URL https://kuehlingkuehling.de/ht500-3/

[32] F. Wasserfall, N. Hendrich, D. Ahlers, Optical In-Situ Verification of 3D-Printed Electronic Circuits, Proceedings of the 15th IEEE Conference on Automation Science and Engineering (CASE) (2019) 1302–1307doi:10.1109/COASE.2019.8842835.

[33] G. Häußge, Octoprint project website, accessed: March 2020. URL https://octoprint.org

[34] F. Wasserfall, Octopnp github repository, accessed: February 2020. URL https://github.com/platsch/OctoPNP/

[35] F. Wasserfall, D. Ahlers, N. Hendrich, J. Zhang, 3D-Printable Electronics - Integration of SMD Placement and Wiring into the Slicing Process for FDM Fabrication, Proceedings of the 27th International Solid Freeform Fabrication Symposium (2016) 1826–1837.

[36] A. Guttman, R-trees a dynamic index structure for spatial searching, Proceedings of the ACM SIGMOD international conference on Management of data (1984) 47–57. doi:10.1145/602259.602266.

[37] Boost Graph Library documentation, accessed: March 2020. URL https://www.boost.org/doc/libs/1_72_0/libs/graph/doc/astar_search.html

[38] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.