STAG: Smart Tools and Applications in Graphics (2018)
M. Livesu, G. Pintore and A. Signoroni (Editors)

slice2mesh: meshing sliced data for the simulation of AM Processes

Marco Livesu Daniela Cabiddu Marco Attene

CNR IMATI, Genoa, Italy

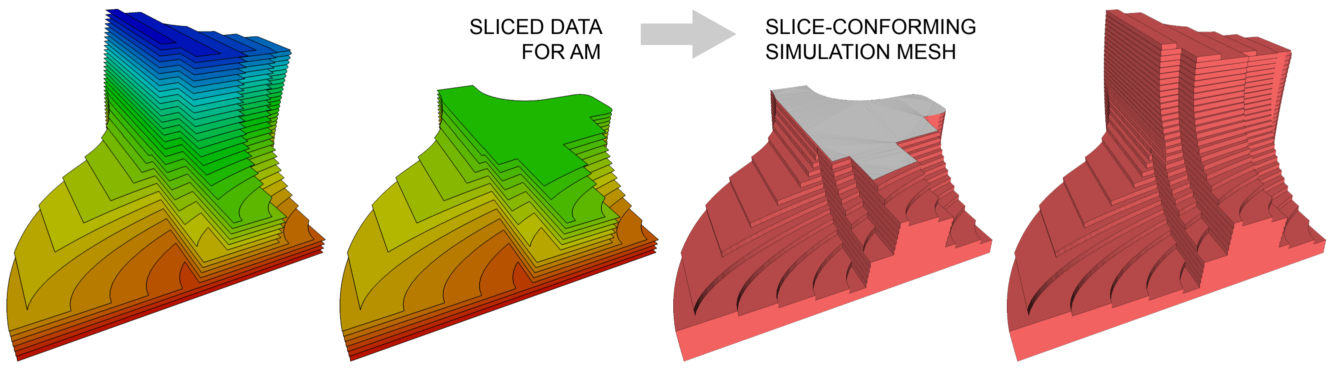

Figure 1: *We address mesh generation for the simulation of AM processes, proposing a tool that converts sliced data (left) directly into a volumetric mesh which endows in its connectivity the temporal evolution of the object to be printed (right).*

Abstract

Accurately simulating Additive Manufacturing (AM) processes is useful to predict printing failures and test 3D printing without wasting precious resources, both in terms of time ad material. In AM the object to be fabricated is first cut into a set of slices aligned with the build direction, and then printed, depositing or solidifying material one layer on top of the other. To guarantee accurate simulations, it is therefore necessary to encode the temporal evolution of the shape to be printed within the simulation domain. We introduce `slice2mesh`, to the best of our knowledge the first software capable of turning a sliced object directly into a volumetric mesh. Our tool inputs a set of slices and produces a tetrahedral mesh that endows each slice in its connectivity. An accurate representation of the simulation domain at any time during the print can therefore be easily obtained by filtering out the slices yet to be processed. `slice2mesh` also features a flexible mesh generation system for external supports, and allows the user to trade accuracy for simplicity by producing approximate simulation domains obtained by filtering the object in slice space.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Additive manufacturing (AM) enables the fabrication of a three-dimensional object by depositing successive layers of material one on top of the other. The process starts by cutting the object with a collection of planes orthogonal to the build direction, defining a set of 2D cross sections (or *slices*). The machine tool paths along which the printer will deposit or solidify material are then computed, translated into machine code, and transmitted to the 3D printer for the actual fabrication. For industrial printers the computation of the tool paths can be extremely complex and machine dependent. Therefore, these printers usually input a file containing the slices and calculate machine tool paths in a computer directly installed into the machine. A popular format to represent sliced data is the Common Layer Interface (CLI), which defines each slice as a set of piece-wise linear curves.

In AM the 3D printer is oblivious of the original shape. The fabrication is completely based on the geometry of the slices. In other words, the printer fabricates a proxy shape obtained by extruding the geometry of each slice along the build direction by an amount corresponding to the local layer thickness. The accuracy of the proxy depends on many factors, such as the build direction and the spacing between adjacent slices. The optimization of these parameters has been the subject of extensive research in recent years (see [LEM*17] and references therein).

Despite the huge amount of research in the field, preparing an object for fabrication with AM is still a trial-and-error operation in which the user experience plays a fundamental role [ALL*18]. Poor quality objects and even printing failures occur quite often for inexperienced users, increasing the production cost and limiting the scalability of 3D printing. Simulating fabrication processes provides an efficient way to study mitigation strategies to prevent failures [MVSC16, HL14] and possibly damages induced by a misuse of the printer. Furthermore, simulation allows to predict the ultimate product quality.

In this paper we focus on the generation of proper domains to simulate additive fabrication. From a meshing perspective the simulation poses three main challenges:

- *Domain*: the shape evolves in time, growing one layer at a time. To accurately simulate AM processes it is therefore necessary to generate a mesh that embeds in its connectivity the temporal evolution of the object. Doing so, a faithful representation of the domain at any time during the simulation can be obtained by simply filtering out all the mesh elements belonging to the slices yet to be processed. Notice that in principle one could also decide to re-mesh the domain at each time step, but this solution can be dramatically expensive from a computational standpoint;
- *Refinement*: especially in metal printing, the areas just hit by the laser reach very high temperatures, producing huge thermal gradients. In order to accurately catch the thermal and mechanical phenomena it is therefore important to be able to locally refine the mesh to improve the accuracy of the simulation;
- *Supports*: professional software such as Materialise Magics [Mat] define the support structures used to sustain the part during fabrication directly in the CLI file, in the form of piece-wise linear 1D curves. An accurate digital representation of the supports never exists into the machine, but rather supports are directly created by asking the printer to deposit material along these curves. Their ultimate size and shape will therefore depend on the machine precision (e.g., the diameter of the laser beam for SLS/SLM machines, or the thickness of the plastic filament for FDM printers). Support structures have a fundamental role in 3D printing, therefore a method to synthesize their shape and incorporate them into the simulation domain must be devised.

Current AM simulation approaches rely on standard meshing algorithms which do not specifically address these issues. Approximate domains such as rectilinear grids (i.e. voxels) are popular, but they need to be very dense to conform with the mesh slices and hardly support local refinement. Supports can be simulated assuming they are one voxel thick, but this assumption may be too rough and lead to inaccurate results. This is particularly true in metal printing, where the principal function of supports is to dissipate heat, and therefore small variations of their thickness may heavily affect the final result.

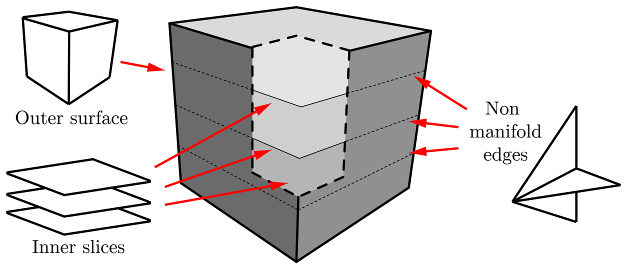

Figure 2: *Detailed structure of the Piece-wise Linear Complex (PLC) we create and eventually fill with tetrahedra. Incorporating both the boundary and the inner slices into the PLC produces non-manifold edges which are not supported by variational tetrahedralization approaches such as [ACSYD05].*

We present `slice2mesh`, to the best of our knowledge the first software capable of turning a sliced object directly into a volumetric mesh. Our meshing strategy starts with a CLI file and generates a tetrahedral mesh which embeds in its connectivity all the slices contained in such file. In other words, any tetrahedral element is fully contained in a single slice, with no element spanning across adjacent slices (Figure 1). This feature enables for an easy and efficient extraction of the simulation domain at any time during the fabrication process. Furthermore, being unstructured the mesh can be locally refined to improve the simulation accuracy.

We incorporated in our tool also practical features such as the meshing of support structures (Figure 12) and the generation of approximate simulation domains with lower elements count, obtained by filtering the object in slice space (Figures 8 and 10). This is very important to control the trade-off between simulation accuracy and running times. Indeed, high fidelity printers may use very thin slices with layer thicknesses in the order of microns, leading to meshes with millions of elements that can make the simulation prohibitive from a computational point of view. To this end, working in slice space makes our software independent from the complexity of the object to be printed, resulting in a robust simplification strategy.

In this article we discuss both technical solutions and algorithms at the basis of `slice2mesh`. A critical evaluation of the capabilities and limitations of our tool, as well as a view on its future improvements, are included at the end of the paper.

2. Related works

Our work relates to both volumetric mesh generation and FEM analysis of AM processes. We review here the most relevant works in the aforementioned research areas.

2.1. Volume mesh generation

Scientific literature offers a variety of methods for both hexahedral [XWHJ16, LVS*13, LLX*12] and mixed element [GJTP17, SRUL16] meshing. However, these methods do not permit to control the interior of the volume, therefore cannot be used to generate a mesh that conforms with both the boundary and the slices of a 3D printed object. Furthermore, they act like re-meshing tools, meaning that they are not able to turn a surface mesh directly into a volumetric one, but rather rely on a temporary volumetric discretization of the shape (typically a tetrahedral mesh).

For the tetrahedral meshing case, variational and Voronoi-based are the most popular approaches. Variational methods like [ACSYD05] require the input to be a 2-manifold. Being conforming with both the boundary and the slices imposes the presence of non-manifold edges (Figure 2), therefore such methods cannot be used in this context. Voronoi-based methods such as Tetgen [Si15] can mesh any Piece-wise Linear Complex (PLC), possibly containing non-manifold edges. In the last step of our approach we use Tetgen to turn our PLC into a full tetrahedral mesh.

Besides manifoldness, a surface mesh must satisfy other important requirements to be turned into a volumetric one: it must be watertight (i.e., it must fully enclose a solid), and it must not contain self-intersections. Although mesh repairing [ZGZJ16, ACK13, JKSH13] can alleviate these defects, most of the repairing tools available do not support non-manifoldness. Furthermore, mesh repairing often uses rational arithmetic and may be a huge bottleneck for the performances of the shape generation pipeline. The piece-wise linear complexes generated with our method are guaranteed to enclose a solid and do not contain self-intersections. As such, they do not need to be repaired and can be directly turned into tetrahedral meshes using Tetgen [Si15] or similar tools. Recent research in the field has shown that solid meshes can be also constructed starting from defective inputs that do not fulfill the all these requirements [HZG*18]. This tool was released while this article was in preparation, and we could not test it for our specific use case. We plan to evaluate it, and possibly include it in future releases of `slice2mesh` (Section 7).

2.2. Simulation of AM processes

The simulation of AM processes has fostered a lot of research in recent years, especially from the FEM community. The majority of the methods in literature focuses on the simulation of heat dissipation and residual stresses, two major issues for current 3D printing technologies. Here we summarize the current trends in terms of mesh generation for the simulation of AM processes; we point the reader to [SCS15] for a more comprehensive survey.

Uniform size meshing. Matsumoto and colleagues [MSOA02] proposed a layer by layer simulation performed on a regular 2D grid. In heat diffusion the major temperature gradients are those developed in the build direction, between the top layer and the substrate. As a result, 3D models should be preferred to 2D ones for obtaining accurate results [SCS15], even for the simulation of residual stresses [VBVB12]. Other methods employ discrete rods to simulate the plastic filament of FDM [STA17] or voxel-based approaches, performing the simulation on regular 3D lattices [DKS00, DS01, DS02, DS03, MB07, AAS14]. Although the horizontal component of the lattice can be forced to align with the slices (if a uniform strategy is used), too thin layers may produce excessively dense voxel grids, leading to computationally expensive simulations. Furthermore, voxelized objects do not faithfully reproduce the outer boundary of the prototype and may miss tiny or off-axis features, negatively affecting the result of the simulation.

Local refinement. In Selective Laser Melting (SLM), Selective Laser Sintering (SLS) and many others AM processes the biggest thermal gradients are localized nearby the melting pool. In order to better catch this behaviour and keep the simulation cost affordable recent methods exploit local mesh refinement. Many authors employ rather simple hexahedral meshes [YZC*16, HMR15, dSLCC14, CCL11, HHYE13], with finer elements nearby the melting pool (where the powder is molten by the heat source) and coarser elements elsewhere (where the thermal gradients are very low). Deposition layers are usually 1 element tall and 2 elements wide, thus generating elements being approximately equal to $\frac{1}{4}$ of the laser diameter, as suggested in [HMR15]. Both [RSM14] and [KBG*04] use a full structured adaptive hexahedral mesh with T-junctions. This mesh has the same limitations of voxels-based approaches. Furthermore, note that additional constraints on the solution values should be added on T-nodes so as to guarantee the continuity of the solution. Liu and colleagues [LZZ*12] proposed a micro scale tetrahedral mesh with consideration of powder arrangement. They experimented with a single layer sintering with the sample dimension of $6.13mm \times 6.08mm \times 1.54mm$, which already counts 60K tetrahedra. The authors do not discuss how to adapt their meshing to general 3D shapes and, even if so, meshing the entire domain according to such strategy would lead to extremely dense meshes, making the simulation prohibitive from a computational point of view. Zhang and colleagues [ZCL*10] use a mixed element mesh, with the top layer meshed as a regular array of hexahedra and the stack of layers below meshed with a progressively coarser tetrahedral mesh. Since the mesh is not conforming with the previous layers, a new mesh needs be generated for each slice.

Summarizing, none of the meshing methods presented in Section 2.1 supports the generation of volumetric meshes which are specific for the simulation of AM processes and encode in their interior the slice geometry. Additionally, all the settings described in Section 2.2 either rely on manually generated meshes or are too simple to scale on complex shapes. With `slice2mesh` we offer a meshing method that successfully addresses all these shortcomings.

3. Anatomy of a CLI file

The Common Layer Interface format [CLI] serves to encode sliced data for 3D printers and is supported both from commercial [Mat] and academic software. CLI files are accepted by a variety of desktop and industrial 3D printing machines (e.g. the EOS M270 laser sintering metal printer). We briefly explain here how data is organized and what data we extract for our purposes.

The header presents global information, such as the number of

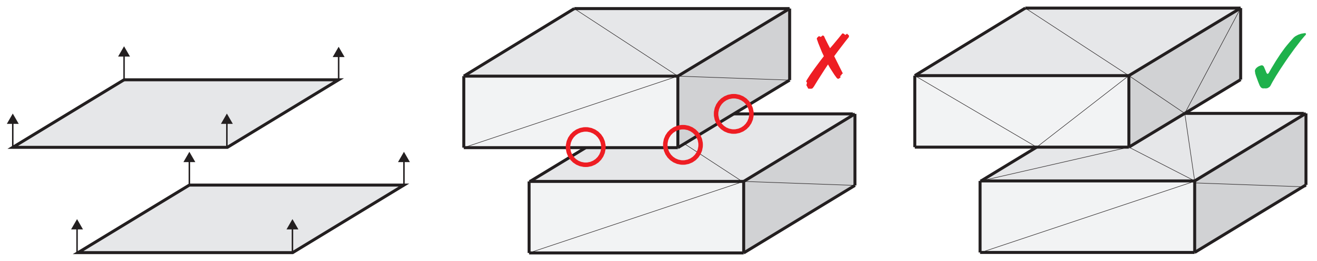

Figure 3: *Extruding each slice along the building direction (left) by an amount corresponding to the local layer thickness produces a digital copy of the object to be printed. If this process is performed locally, each extruded slice will be disconnected from its adjacent slices, thus generating a non-conforming mesh that cannot be used to produce a volumetric discretization of the domain (middle). Our method is guaranteed to produce a conforming Piece-wise Linear Complex (PLC) that can be turned into a valid simulation domain with any off-the-shelf volumetric mesher available (right).*

slices encoded in the file and the metric unit in which coordinates are expressed. Slices are then listed as a sequence of layers. Each layer begins with the keyword $$LAYER and is followed by a sequence of hatches ($$HATCHES) and polylines ($$POLYLINE). Hatches are sets of independent straight lines, each defined by one start and one end point. The purpose of hatches is to define both external structures and the machine tool paths along which the printer deposits or solidifies material. Polylines can be of three types: closed CCW, closed CW or open. Closed CCW polylines are used to represent the external boundaries of the slice. Closed CW polylines are used to represent the internal boundaries of the slice (i.e. holes). Open polylines are similar to hatches and can be used for the same purposes. The only difference is that hatches are interpreted as disconnect segments, whereas open polylines are line strips (the endpoint of the current edge is also the start point of the subsequent one). In our tool we rely on CinoLib [Liv17] for CLI processing, which for simplicity assumes that hatches are used only for machine toolpaths, and open polylines only for external supports. We therefore read from the file only polylines data, and this is the actual input `slice2mesh` uses for mesh generation.

4. Method

We input a CLI file containing a set of 2D slices and (optionally) a thickening radius for support structures; we output a simulation ready discrete domain in the form of a tetrahedral mesh. In case no thickening radius is provided, the support structures contained in the CLI file will not be included in the output mesh.

The algorithm works in two steps. The goal of the first step is to create a Piece-wise Linear Complex (PLC) that encodes both the outer and inner structure of the simulation domain. On the outside, the PLC will conform to the input slices, exposing the typical staircase effect produced by the slice extrusion (Figure 4). On the inside, the PLC will contain the geometry of the slices, so as to incorporate the temporal evolution of the simulation domain (Figure 2). In the second step, the PLC is filled with tetrahedral elements to produce the output mesh.

The first step is based on novel ideas described in Section 5. For the second step we rely on off-the-shelf third party software (i.e., Tetgen [Si15]) to turn our PLC into a tetrahedral mesh.

5. Generation of the PLC

The PLC generation is based on the idea of lifting vertices and edges of each slice one layer above, splitting edges to resolve intersections wherever necessary. Note that, even in absence of intersections, naively duplicating each slice and lifting it to the layer above would double the number of vertices and edges, resulting in a PLC where each thickened slice is disconnected from the others (Figure 3). As an example, the reader may consider a slicing composed of a stack of perfectly aligned quads. When lifted, each quad should be aware that it is a copy of an already existing quad, and no additional vertices should be added in the PLC.

We propose here a method which is able to address these issues and produce a correct PLC. Our approach is based on four steps. We first thicken the support structures, converting 1D lines to 2D polygons (Section 5.1). Then, we pre-compute all the lifting information and store it in a data structure (Section 5.2). We eventually proceed with the meshing of the PLC, which is composed of two types of elements: triangles aligned with the build direction (Section 5.3), and triangles that are orthogonal with respect to the build direction (Section 5.4).

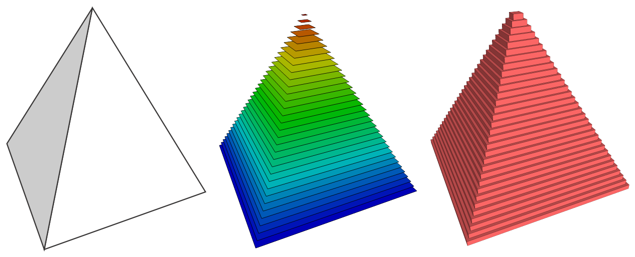

Figure 4: *Slicing a flat surface misaligned with the build direction creates the staircase approximation error typical of additive manufacturing. The staircase effect is lower for nearly vertical surfaces w.r.t. the build direction, and becomes much higher for nearly horizontal surfaces [LEM*17].*

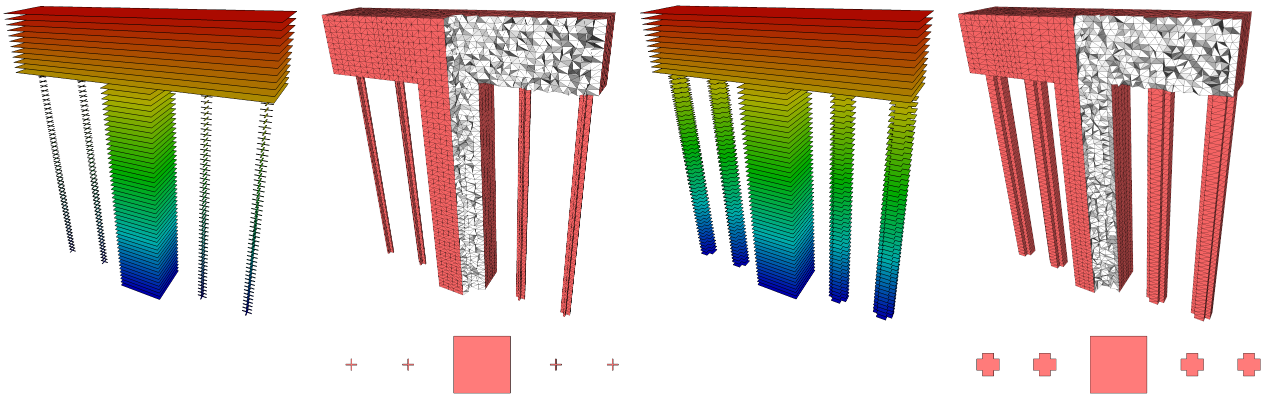

Figure 5: *Two different tetrahedral meshes of a T shape, obtained by using growing thicknesses for supports structures. Supports are often seen by the printer as 1D entities, and their fabricated size depends on hardware (e.g. laser beam or filament diameter).* `slice2mesh` *can be tuned to generate simulation domains specifically tailored for a particular hardware.*

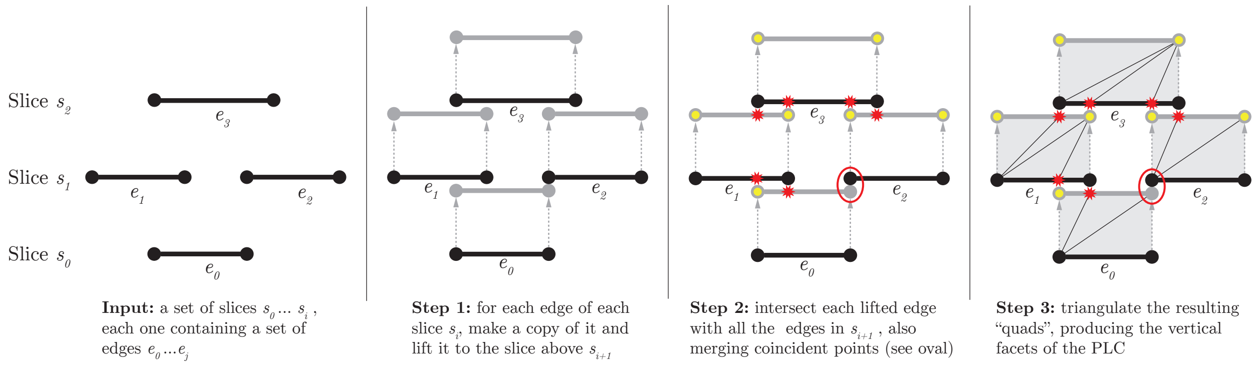
Slice s_2
Slice s_1
Slice s_0
e_3
e_1
e_2
e_0
Input: a set of slices $s_0 \ldots s_i$, each one containing a set of edges $e_0 \ldots e_j$
Step 1: for each edge of each slice s_i, make a copy of it and lift it to the slice above s_{i+1}
Step 2: intersect each lifted edge with all the edges in s_{i+1}, also merging coincident points (see oval)
Step 3: triangulate the resulting "quads", producing the vertical facets of the PLC

Figure 6: *A schematic 2D representation of our edge lifting strategy. Each edge is lifted from its current slice to the slice above, and tested for intersection and coincidence with all the edges in it. The resulting quad-like domains are eventually triangulated to produce the vertical facets of the PLC (right). Detected intersections are marked with a red star, newly added vertices with a yellow circle.*

5.1. Thickening of external supports

The open 1D curves representing external supports must be thickened before triangulation and require special treatment. We thicken them by applying the buffering algorithm implemented in the Boost Polygon Library (Figure 5). Note that many popular patterns for support structures are based on intersecting lines (e.g. lattices [LEM*17]). Furthermore, depending on the thickening radius, supports may also intersect other polygons on the same slice. A union of all the thickened lines and polygons must therefore be computed to resolve intersections (Figure 7). We do this using the Boost Polygon Library, which offers boolean facilities for curves and polygons. Note that, from this point on, it is impossible to distinguish between the sliced object and its support structures: all we have is a set of closed curves representing the outer and inner (holes) profile of 2D polygons. We eventually sanitize polygons, removing all the degenerate or quasi-degenerate edges that may have been created during boolean operations. We do this by using the Douglas-Peucker simplification algorithm [DP73], using 1% of the thickening radius as maximum deviation distance. In our experiments we observed that this latter simplification is fundamental to avoid failures and corner cases (e.g. computing intersections between degenerate edges).

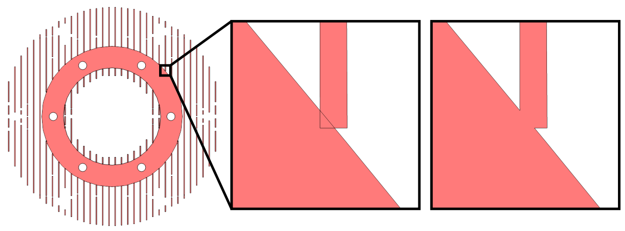

Figure 7: *Details on a slice of the Nugear (with linear supports). Thickening 1D support structures and converting them into polygons may generate intersections with other polygons living in the same slice. We avoid intersections by performing a boolean union of all the polygons in the slice.*

5.2. Pre-processing

To facilitate processing we pre-compute all the intersection data and store it in a data structure to have it ready when meshing the PLC. We test for intersections using Shewchuck's predicates [She97], thus ensuring the necessary numerical robustness.

We initialize V as an array containing all the slice vertices, and a E as an array containing all the edges. Edge endpoints in E are indexed with respect to the vertices in V. Ideally, at the end of the pre-processing we would like to have an updated version of V and E such that: V contains all and only the vertices of the PLC, and for each edge $e \in E$ we know

- what are the ids of its lifted endpoints (they may be vertices of the slice above, or newly generated vertices appended in V);
- how many edges from the slice above intersect the lifted copy of e (and where);
- how many edges lifted from the slice below intersect e (and where);

We obtain this information with a progressive approach. where we process one edge at a time. We lift each edge to the next slice, and we test it against all the edges in such slice. The complete procedure is given in Algorithm 1. A simplified 2D illustration can be found in Figure 6.

5.3. Horizontal meshing

With all the lifting data pre-computed, we can easily proceed with the generation of the PLC. Here we descibe how to generate its *horizontal* facets, that is, the ones that are aligned with the build direction. We will complete the PLC with its vertical facets in the subsequent section.

Horizontal meshing is local w.r.t. each slice and the slice immediately below (if any). For each slice s_i, we first go through each of its edges $e \subset E$, and split it at any of the intersection points found in pre-processing. This generates $n+1$ intersection-free sub-edges, where n is the number of intersection points detected in pre-processing. We augment the edge set of slice s_i by adding the lifted images of edges $e' \subset E$ in the slice below s_{i-1}, which we also split at their intersection points. Note that processing s_i and s_{i-1} separately may produce duplicated edges (e.g. if the slices perfectly overlap). We encode edges in a symbolical way (as pairs of vertex ids), thus avoiding the generation of duplicated entities.

We eventually mesh the slice s_i by generating a Constrained Delaunay Triangulation (CDT) of the so generated set of vertices and unique intersection-free edges, using the Triangle [She96] library. Triangle first constructs a CDT of the convex hull of the input set, and then removes the unnecessary triangles proceeding from the outside towards the interior until the input edges are revealed. Note that internal holes will be also filled with triangles. To clear holes we therefore filter the triangle list, discarding elements that do not project inside s_i or s_{i-1}. Triangles are either completely inside or outside the slice, therefore this check can be done considering triangle centroids and performing a point-in-polygon test. Repeating this procedure for each slice, we produce all the horizontal facets of the PLC.

5.4. Vertical meshing

Here we describe how to produce the PLC faces orthogonal to the build direction with a local, per edge, meshing strategy. Once again, we exploit the information pre-computed in Section 5.2.

Given an edge e and its lifted copy e', we define a quad having as bottom vertices the extrema of e and as top vertices their lifted image (the extrema of e'), The lower base of this quad is split into as many sub-segments as the number of split points computed for e.

```
Data: Slice vertices $V$ and edges $S$
Result: PLC vertices $V$, and edge split info
for each edge $e(v_a, v_b)$ lifted from slice $s_i$ do
    for each edge $e'(v_c, v_d)$ from slice $s_{i+1}$ do
        $p = (v_a, v_b) \cap (v_c, v_d)$;

        // try to lift $v_a$;
        if $p = v_a = v_c$ then
            set $v_c$ as lifted image of $v_a$
        else
            if $p = v_a = v_d$ then
                set $v_d$ as lifted image of $v_a$
            else
                if $p = v_a$ then
                    add vertex $p$ to $V$ and set it as lifted image of $v_a$
                end
            end
        end

        // try to lift $v_b$;
        if $p = v_b = v_c$ then
            set $v_c$ as lifted image of $v_b$
        else
            if $p = v_b = v_d$ then
                set $v_d$ as lifted image of $v_b$
            else
                if $p = v_b$ then
                    add vertex $p$ to $V$ and set it as lifted image of $v_b$
                end
            end
        end

        // update split info for $e(v_a, v_b)$;
        if $p = v_c$ and $p \neq v_a$ and $p \neq v_b$ then
            add $v_c$ as split point of lifted edge $e(v_a, v_b)$
        else
            if $p = v_d$ and $p \neq v_a$ and $p \neq v_b$ then
                add $v_d$ as split point of lifted edge $e(v_a, v_b)$
            else
                if $p \neq v_a \neq v_b \neq v_c \neq v_d$ then
                    add vertex $p$ to $V$ and set it as split point of lifted edge $e(v_a, v_b)$
                end
            end
        end

        // update split info for $e'(v_a, v_b)$;
        if $p = v_a$ and $p \neq v_c$ and $p \neq v_d$ then
            add $v_a$ as split point of edge $e'(v_c, v_d)$
        else
            if $p = v_b$ and $p \neq v_d$ and $p \neq v_d$ then
                add $v_b$ as split point of edge $e'(v_c, v_d)$
            else
                if $p \neq v_a \neq v_b \neq v_c \neq v_d$ then
                    add vertex $p$ to $V$ and set it as split point of edge $e'(v_c, v_d)$
                end
            end
        end
    end
end

// lift unlifted vertices;
for each vertex with no lifted image yet do
    lift it to the slice above and append the new vertex to $V$;
end
```

Algorithm 1: Our edge pre-processing strategy. Note that in the actual implementation we substituted the inner loop with a query on a spatial data structure (a quad-tree) to avoid useless intersection tests, thus reducing complexity from $O(n^2)$ to $O(n \log n)$.

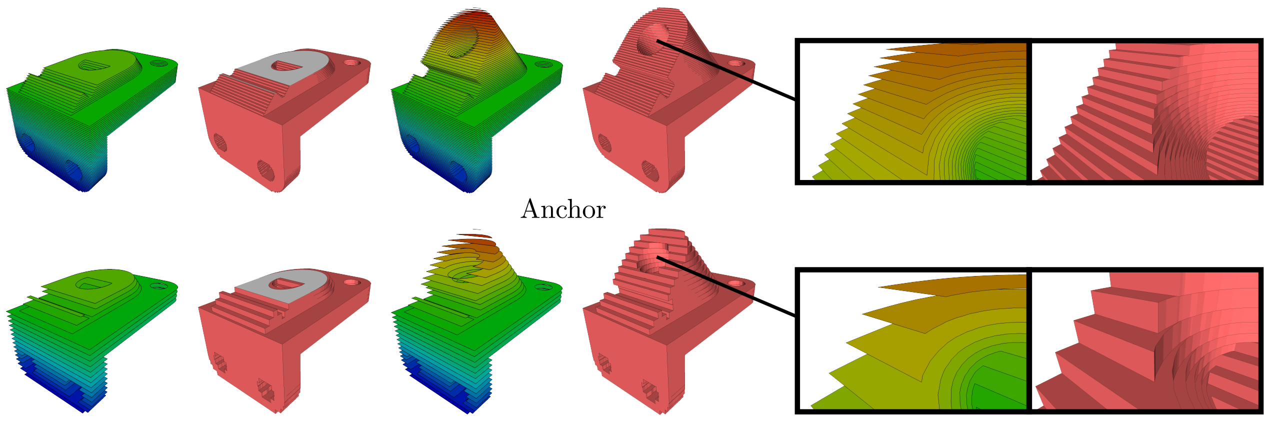

Figure 8: *Two different slicings for the Anchor model, with their associated volumetric meshes. Right: a finer slicing better approximates off-axis features.*

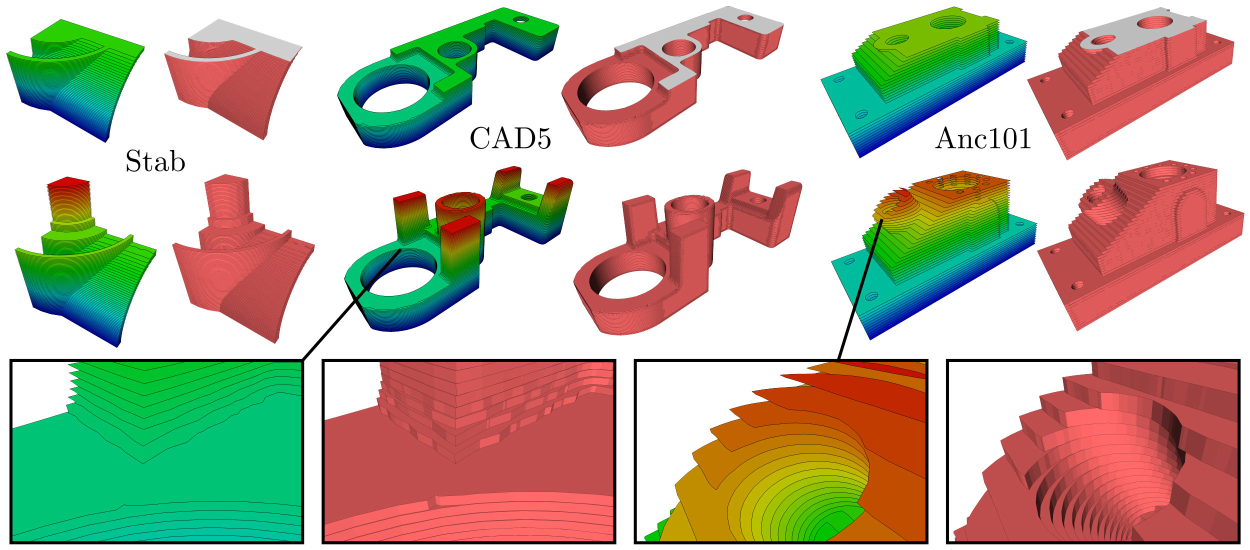

Figure 9: *A gallery of sliced CAD models meshed with* `slice2mesh`.

The same goes for the upper base, which is split into as many sub-segments as the number of split points computed for e'. Having this information, we can trivially triangulate the resulting convex polygon starting from the bottom-left corner and connecting it with all the split points of e' plus the top right corner, and starting from the top right corner and connecting it with all the split points of e. The result of this procedure is illustrated in the right part of Figure 6. Repeating it for all the edges of all slices but the top one we produce all the missing faces of the PLC, which is now ready for tetrahedralization.

6. Results

We implemented `slice2mesh` using CinoLib [Liv17] for CLI and geometry processing, the Boost Polygon Library for curve thickening and 2D booleans, Triangle [She96] for planar triangulations, and Tetgen [Si15] for the final tetrahedralization. In Figures 1, 8 and 9 we show a variety of results. In Table 1 we report statistics and timings. We release our tool to the public domain, making available at the following link `https://github.com/mlivesu/slice2mesh`.

Conforming vs non-conforming meshes. `slice2mesh` generates meshes that encode the temporal evolution of the domain and are therefore more accurate than a mesh produced with general purpose meshing tools. For general meshes the temporal evolution of the domain can be artificially simulated by filtering out all the tetrahedra with centroid above the quote of the current slice (Figure 11). Note however that the surface exposed by this naive mesh filtering will be much higher, possibly resulting in unfaithful estimation of

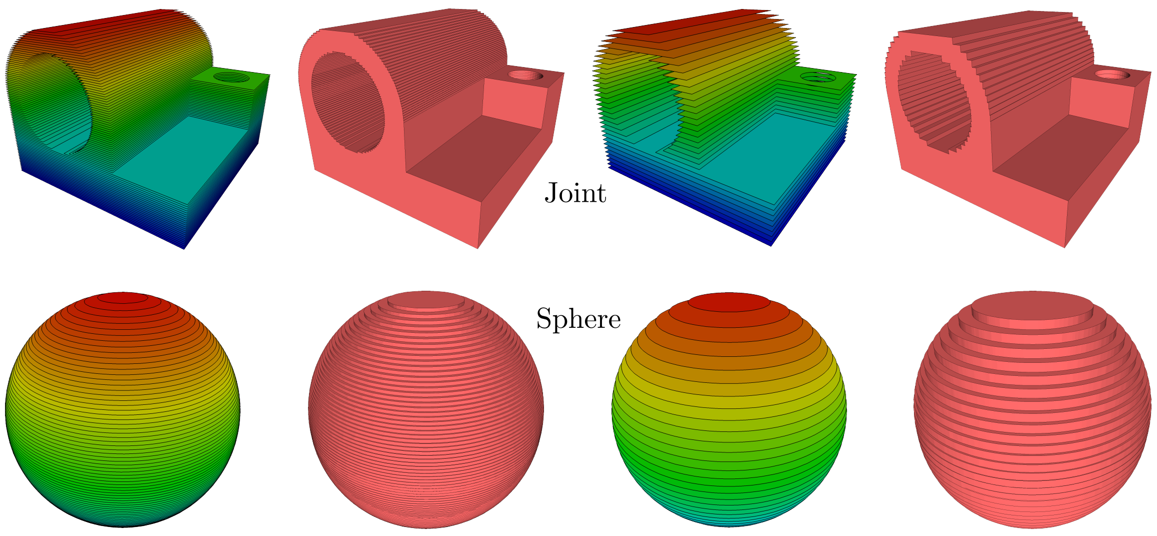

Figure 10: *Precisely simulating the activity of industrial printers may be too ambitious, due to the extremely thin slices they use.* `slice2mesh` *can generate approximate representations of the simulation domain by filtering the shape in slice space (i.e. sub-sampling the input slices). This allows to perform approximate simulations, in which bundles o multiple adjacent slices are activated at at each time step.*

physical phenomena such as heat dissipation, where the augmented surface area artificially boosts heat exchange.

Conforming voxels vs tets. With a proper choice of mesh density, voxel grids can be forced to be slice conformal and encode the temporal evolution of the printed object. Our method results in a more accurate simulation domain, which is 100% compliant with the proxy shape the printer is asked to fabricate. The same does not hold for voxels, where tiny or off-axis features cannot be represented and may require excessive refinement for a faithful simulation. Moreover, unstructured grids trivially support local refinement, whereas for voxel grids refinement is global (thus more complex and costy).

Supports. `slice2mesh` naturally incorporates external supports into the simulation domain (Figures 5 and 12). The algorithm mimics the action of the printer, thickening 1D lines with a value that adapts to the specifics of the printer. The user can prescribe as thickening radius the laser beam (for laser printing) or the section of the plastic filament (for FDM), thus generating an extremely accurate discrete representation of the real fabricated object. To the best of our knowledge, no commercial or academic tool has a similar feature. Furthermore, thickening happens in slice space, making the algorithm oblivious of the complexity of the supports, leading to a robust and scalable tool.

Simplification. Simulating industrial printers can be expensive due to the extremely small layer thicknesses these machines use (order of microns). A way to make computations affordable without using clusters or extremely powerful machines is to activate bundles of n adjacent slices all together. `slice2mesh` supports this simplification by allowing the user to sub-sample the input slices, considering only a sub-set of them for the generation of the PLC. As for supports, everything happens in slice space, therefore computations are oblivious of the complexity of the object to be printed, making simplification scalable and robust. Numbers regarding slice filtering are given in Table 1 (see models with two line entries). Visual representations of the so generated domains are given in Figures 8 and 10.

PLC. Besides simulation, the PLC has its own value and could be used for other AM-related purposes. Having a digital representation

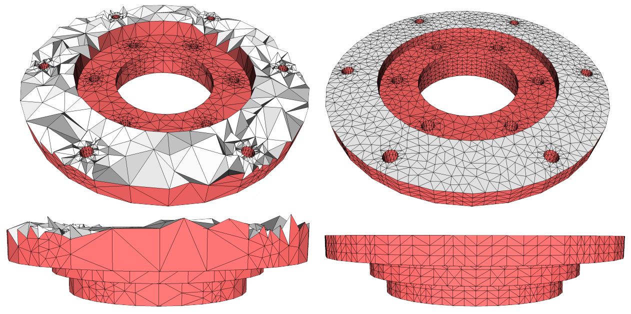

Figure 11: *Left: a general tetrahedral mesh, where all the elements having their centroid above the current slice have been filtered out. Right: our slice-conformal tetrahedral mesh. Simulations of physical phenomena such as heat dissipation may be unreliable due to the area of the exposed surface, which is artificially increased in the left example (1308.54mm^2 vs 778.23mm^2).*

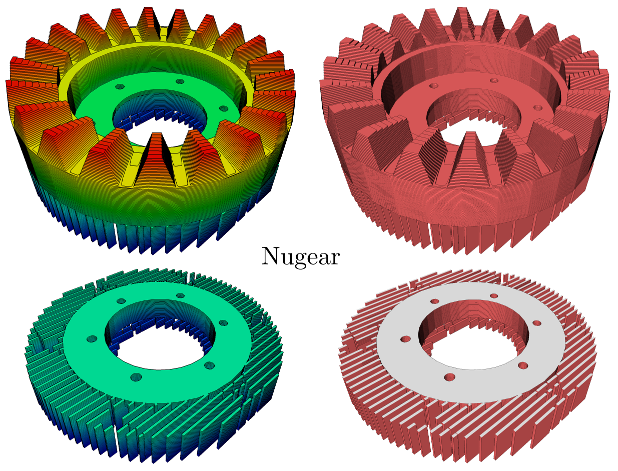

Figure 12: *An example of volumetric mesh which includes both the object and its thickened support structures. Matching the thickening factor with the specifications of the printer (e.g. filament thickness or laser beam) the user can generate simulation domains which are tailored for a specific hardware.*

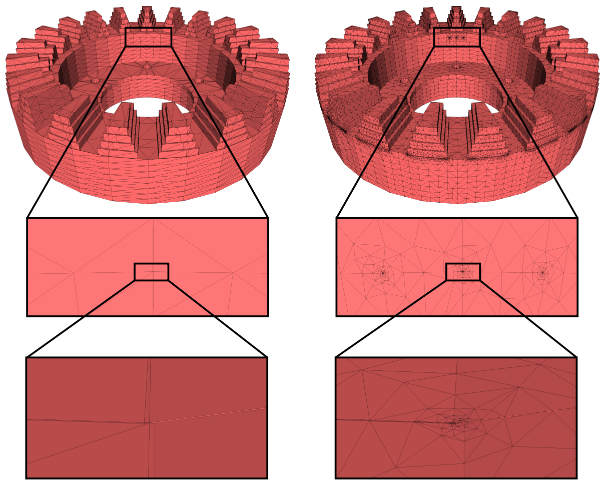

Figure 13: *Left: a slice conformal tetrahedral mesh with minimal Steiner points. Right: a refinement of the mesh aside, obtained with the* `-q` *flag of Tetgen. When adjacent slices are very similar but not identical,* `slice2mesh` *tends to introduce extremely tiny features which make the meshing challenging and trigger over-refinement if quality bounds are prescribed on tetrahedra.*

of the printed object can be useful for rendering and education, but also for estimating quantities such as the staircase error and the volumetric loss, which are extensively used in the process planning pipeline, for example to optimize the build orientation or surface finish [ALL*18, LEM*17]. With optional flags, `slice2mesh` can be asked to output the PLC. The user can choose whether to include the inner faces, or to export only the external boundary.

Model	Slices	t_{pre}	t_{horiz}	t_{vert}	t_{tet}	Verts	Tets
Anc101	76	4.07	2.93	0.02	45.98	141K	478K
	38	1.93	1.44	0.01	18.08	68K	238K
Anchor	86	0.39	0.24	<0.01	7.93	33K	110K
	29	0.12	0.07	<0.01	2.53	11K	38K
CAD5	77	2.75	1.81	0.02	30.10	148K	492K
Fandisk	35	0.26	0.18	<0.01	4.59	20K	67K
Joint	83	0.08	0.05	<0.01	12.63	53K	171K
	31	0.02	0.02	<0.01	2.10	11K	35K
Nugear	88	2.69	1.05	0.01	53.1	200K	620K
	19	0.05	0.02	<0.01	0.94	5K	15K
Pyramid	33	0.02	0.02	<0.01	0.84	6K	17K
Sphere	83	0.55	0.39	<0.01	8.17	44K	139K
	33	0.21	0.15	<0.01	30.6	18K	56K
Stab	95	1.25	0.81	0.01	17.26	74K	239K
T	60	0.02	0.01	<0.01	0.47	3K	6K

Table 1: *Statistics of our method. We implemented* `slice2mesh` *as a single threaded C++ application and run it on a Mac Book Pro equipped with a 2.9GHz Intel i5 and 16GB or RAM. For each model we report: number of slices; execution times (in seconds) for edge pre-processing (t_{pre}), horizontal meshing (t_{horiz}), vertical meshing (t_{vert}) and tetmesh generation t_{tet}; and output statistics (number of vertices/elements of the tetmesh). Running times are dominated by tetrahedral mesh generation, for which we rely on Tetgen [Si15]. Note that we report the time necessary to produce a tetrahedral mesh with as few Steiner points as possible (i.e., no quality bounds were set when calling Tetgen).*

7. Limitations

In its current version `slice2mesh` suffers from a number of limitations, mostly regarding the generation of the tetrahedral mesh. Currently the PLC may contain incredibly tiny features that are difficult to handle for Tetgen, possibly resulting in failures. Even if failure does not occur, these features may trigger over refinement, resulting in tetrahedral meshes that are overly dense in some isolated spots (Figure 13). This behaviour seems to be triggered by adjacent slices which almost perfectly project one on top of the other and, when lifted, generate tiny details that are beyond the capabilities of the finite arithmetic used in Tetgen. To generate our results we always called Tetgen with the `-T1e-13` flag, which sets a very low tolerance for the coplanarity test. Without this flag, most of the times Tetgen is not able to produce a mesh. Nonetheless, when a quality bound is provided (i.e. with the `-q` flag), Tetgen often enters in a refinement loop which may keep the machine busy even for hours. We are currently considering different strategies to simplify the PLC in post-processing, removing tiny details before calling Tetgen. The recently released TetWild [HZG*18] (which uses rational arithmetic) may also be a suitable alternative, although we haven't tested it enough in depth yet.

8. Conclusions and future works

We presented `slice2mesh`, a tool that offers dedicated meshing facilities that are specifically tailored for the simulation of additive manufacturing processes. In the paper we show that the simulation of AM processes poses a number of challenges for mesh generation tools, and that general purpose meshing techniques fall short, offering either rough approximations of the domain (e.g. voxels), or meshes that fail to encode the temporal evolution of the printed object.

We believe the solutions we proposed are just a scratch on the surface of the problem. At the moment, the limitations on tetrahedral mesh generation reported in Section 7 severely limit the usability of `slice2mesh`, making it more an academic toy than a real meshing tool. To make it a concrete option for practitioners and professionals, all such limitations should be addressed and robustly handled.

For future works, the next natural step is to validate our meshes in real simulations. We aim to conduct rigorous experiments to measure the performances of our slice conformal meshes when compared with other *"fabrication-unaware"* meshing techniques. Other interesting venues for future research are: the inclusion of machine tool paths into the tessellation (which would allow not only to activate one slice at a time, but also to literally follow the actual path the machine uses), and the generation of hexahedral (or prism) meshes, which would help us to reduce the number of degrees of freedom and produce coarser meshes for faster simulations.

Acknowledgements

This project has received funding from the European Union Horizon 2020 research and innovation program, under grant agreement No.680448 (CAxMan).

References

[AAS14] ANTONY K., ARIVAZHAGAN N., SENTHILKUMARAN K.: Numerical and experimental investigations on laser melting of stainless steel 316l metal powders. *Journal of Manufacturing Processes 16*, 3 (2014), 345–355. 3

[ACK13] ATTENE M., CAMPEN M., KOBBELT L.: Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR) 45*, 2 (2013), 15. 3

[ACSYD05] ALLIEZ P., COHEN-STEINER D., YVINEC M., DESBRUN M.: Variational tetrahedral meshing. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 617–625. 2, 3

[ALL*18] ATTENE M., LIVESU M., LEFEBVRE S., FUNKHOUSER T., RUSINKIEWICZ S., ELLERO S., MARTÍNEZ J., BERMANO A. H.: Design, representations, and processing for additive manufacturing. *Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging 10*, 2 (2018), 1–146. 2, 9

[CCL11] CONTUZZI N., CAMPANELLI S., LUDOVICO A.: 3 d finite element analysis in the selective laser melting process. *International Journal of Simulation Modelling 10*, 3 (2011), 113–121. 3

[CLI] Common layer interface. http://www.hmilch.net/downloads/cli_format.html. 3

[DKS00] DAI K., KLEMENS P., SHAW L.: Numerical simulation of bi-materials laser densification. In *the proceedings of the 11th Annual SFF Symposium, edited by DL Bourell, et al., The University of Texas* (2000), pp. 386–392. 3

[DP73] DOUGLAS D. H., PEUCKER T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (1973), 112–122. 5

[DS01] DAI K., SHAW L.: Thermal and stress modeling of multi-material laser processing. *Acta Materialia 49*, 20 (2001), 4171–4181. 3

[DS02] DAI K., SHAW L.: Distortion minimization of laser-processed components through control of laser scanning patterns. *Rapid Prototyping Journal 8*, 5 (2002), 270–276. 3

[DS03] DAI K., SHAW L.: Finite-element analysis of effects of the laser-processed bimaterial component size on stresses and distortion. *Metallurgical and Materials Transactions A 34*, 5 (2003), 1133–1145. 3

[dSLCC14] DE SARACIBAR C. A., LUNDBÄCK A., CHIUMENTI M., CERVERA M.: Shaped metal deposition processes. In *Encyclopedia of Thermal Stresses*. Springer, 2014, pp. 4346–4355. 3

[GJTP17] GAO X., JAKOB W., TARINI M., PANOZZO D.: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 114. 3

[HHYE13] HUSSEIN A., HAO L., YAN C., EVERSON R.: Finite element simulation of the temperature and stress fields in single layers built without-support in selective laser melting. *Materials & Design 52* (2013), 638–647. 3

[HL14] HILLER J., LIPSON H.: Dynamic simulation of soft multimaterial 3d-printed objects. *Soft robotics 1*, 1 (2014), 88–101. 2

[HMR15] HEIGEL J., MICHALERIS P., REUTZEL E.: Thermo-mechanical model development and validation of directed energy deposition additive manufacturing of ti–6al–4v. *Additive Manufacturing 5* (2015), 9–19. 3

[HZG*18] HU Y., ZHOU Q., GAO X., JACOBSON A., ZORIN D., PANOZZO D.: Tetwild - tetrahedral meshing in the wild. *ACM Transactions on Graphics (SIGGRAPH 2018) (to appear)* (2018). 3, 9

[JKSH13] JACOBSON A., KAVAN L., SORKINE-HORNUNG O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 33. 3

[KBG*04] KOLOSSOV S., BOILLAT E., GLARDON R., FISCHER P., LOCHER M.: 3d fe simulation for temperature evolution in the selective laser sintering process. *International Journal of Machine Tools and Manufacture 44*, 2 (2004), 117–123. 3

[LEM*17] LIVESU M., ELLERO S., MARTÍNEZ J., SYLVAIN L., ATTENE M.: From 3d models to 3d prints: an overview of the processing pipeline. *Computer Graphics Forum 36*, 2 (2017), 537–564. 2, 4, 5, 9

[Liv17] LIVESU M.: cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes, 2017. https://github.com/mlivesu/cinolib/. 4, 7

[LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 177. 3

[LVS*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 171. 3

[LZZ*12] LIU F., ZHANG Q., ZHOU W., ZHAO J., CHEN J.: Micro scale 3d fem simulation on thermal evolution within the porous structure in selective laser sintering. *Journal of Materials Processing Technology 212*, 10 (2012), 2058–2065. 3

[Mat] MATERIALISE: Magics. https://www.materialise.com/en/software/magics. 2, 3

[MB07] MA L., BIN H.: Temperature and stress analysis and simulation in fractal scanning-based laser sintering. *The International Journal of Advanced Manufacturing Technology 34*, 9-10 (2007), 898–903. 3

[MSOA02] MATSUMOTO M., SHIOMI M., OSAKADA K., ABE F.: Finite element analysis of single layer forming on metallic powder bed in rapid prototyping by selective laser processing. *International Journal of Machine Tools and Manufacture 42*, 1 (2002), 61–67. 3

[MVSC16] MONTEVECCHI F., VENTURINI G., SCIPPA A., CAMPATELLI G.: Finite element modelling of wire-arc-additive-manufacturing process. *Procedia CIRP 55* (2016), 109–114. 2

[RSM14] RIEDLBAUER D., STEINMANN P., MERGHEIM J.: Thermomechanical finite element simulations of selective electron beam melting processes: performance considerations. *Computational Mechanics 54*, 1 (2014), 109–122. 3

[SCS15] SCHOINOCHORITIS B., CHANTZIS D., SALONITIS K.: Simulation of metallic powder bed additive manufacturing processes with the finite element method: a critical review. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* (2015), 0954405414567522. 3

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222. 6, 7

[She97] SHEWCHUK J. R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry 18*, 3 (1997), 305–363. 6

[Si15] SI H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS) 41*, 2 (2015), 11. 3, 4, 7, 9

[SRUL16] SOKOLOV D., RAY N., UNTEREINER L., LÉVY B.: Hexahedral-dominant meshing. *ACM Transactions on Graphics (TOG) 35*, 5 (2016), 157. 3

[STA17] SHETH S., TAYLOR R. M., ADLURU H.: Numerical investigation of stiffness properties of fdm parts as a function of raster orientation. In *Solid Freeform Fabrication 2017: Proceedings of the 28th Annual International Solid Freeform Fabrication Symposium* (2017). 3

[VBVB12] VAN BELLE L., VANSTEENKISTE G., BOYER J. C.: Comparisons of numerical modelling of the selective laser melting. In *Key Engineering Materials* (2012), vol. 504, Trans Tech Publ, pp. 1067–1072. 3

[XWHJ16] XIANZHONG F., WEIWEI X., HUJUN B., JIN H.: All-hex meshing using closed form-induced polycubes. 3

[YZC*16] YANG Q., ZHANG P., CHENG L., MIN Z., CHYU M., TO A. C.: Finite element modeling and validation of thermomechanical behavior of ti-6al-4v in directed energy deposition additive manufacturing. *Additive Manufacturing* (2016). 3

[ZCL*10] ZHANG D., CAI Q., LIU J., ZHANG L., LI R.: Select laser melting of w–ni–fe powders: simulation and experimental study. *The International Journal of Advanced Manufacturing Technology 51*, 5-8 (2010), 649–658. 3

[ZGZJ16] ZHOU Q., GRINSPUN E., ZORIN D., JACOBSON A.: Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG) 35*, 4 (2016). 3