Computers & Graphics (2019)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

slice2mesh: a Meshing Tool for the Simulation of Additive Manufacturing Processes

Marco Livesu, Daniela Cabiddu, Marco Attene

CNR IMATI, Genoa (Italy)

ARTICLE INFO

Article history:
Received March 1, 2019

Keywords: Computers and Graphics, Formatting, Guidelines

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 and 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.

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 [1] 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 [2]. 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 [3, 4] 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

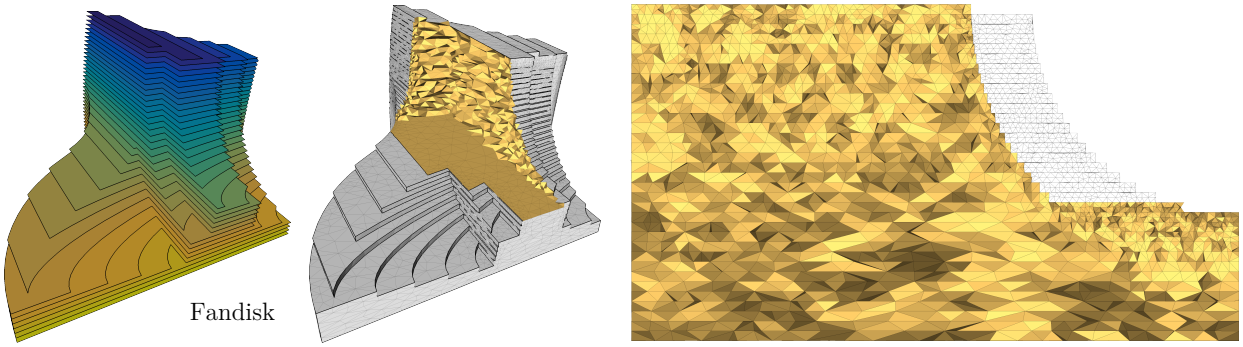

Fig. 1. `slice2mesh` **addresses mesh generation for the simulation of AM processes, converting sliced data (left) directly into a volumetric mesh (middle) which endows in its connectivity the temporal evolution of the object to be printed, as shown in the cut through orthographic view on the right, where the horizontal mesh edges encode the geometry of the input slices.**

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 [5] 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.

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 (Figure 2). 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 17) and the generation of approximate simulation domains with lower elements count, obtained by filtering the object in slice space (Figures 13 and 16). 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`. An early version of this paper was originally presented at the conference *Smart Tools and Apps for Graphics 2018* [6]. The original article was mainly focused on the generation of a Piece-wise Linear Complex that describes the structure of the tetrahedral mesh, providing little details on the actual mesh generation. In particular, the tool was limited in its applicability due to some over refinement that was triggered by tiny features present in the PLC. In this extended version we present a thorough study on the tetrahedralization step, analyzing in depth the performances of two popular mesh generation tools, Tetgen [7] and TetWild [8], and discussing their pros and cons when used within our frame-

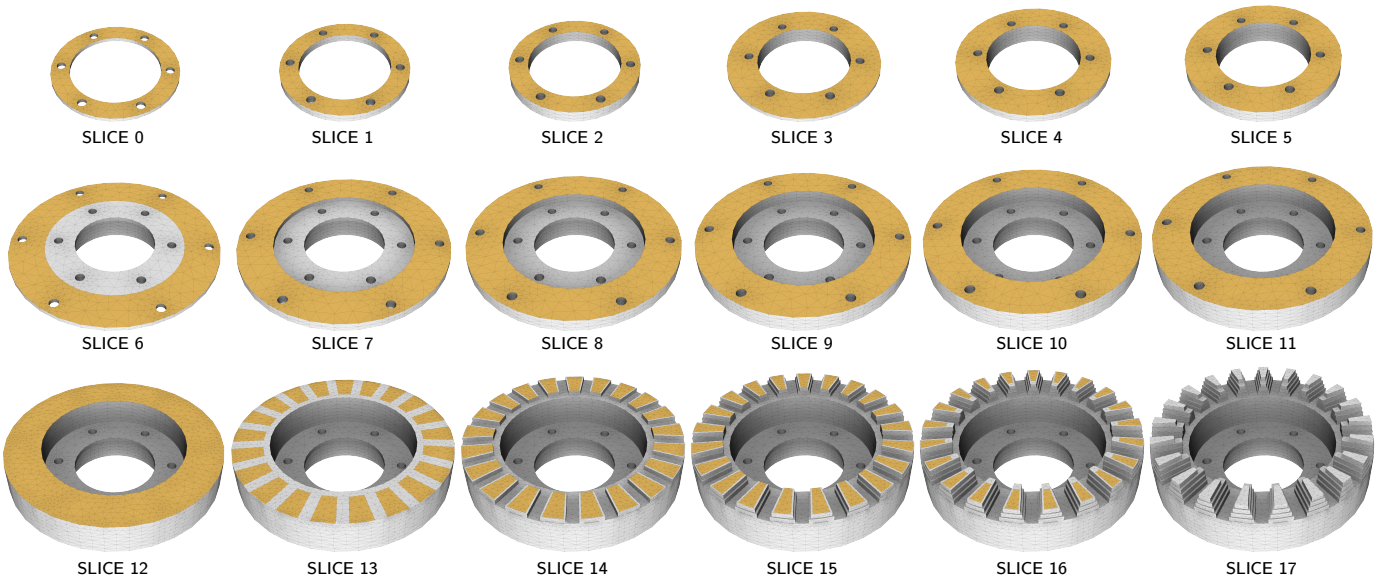

Fig. 2. Slice by slice growing sequence of the simulation domain obtained with `slice2mesh`. The outside of the mesh is white, the interior yellow.

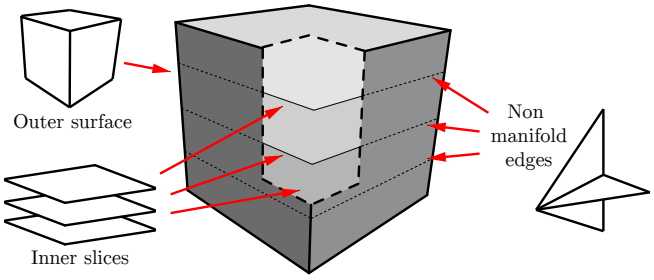

Fig. 3. Detailed structure of the Piece-wise Linear Complex (PLC) we create and eventually fill with tetrahedra. Incorporating both the boundary and the inner slices into the PLC produces non-manifold edges which are not supported by variational tetrahedralization approaches such as [9].

work. This allowed us to improve on our previous results, producing ready to use simulation meshes for all the models we tested. As for the previous version, `slice2mesh` is publicly available on GitHub, and can be accessed by cloning the repo `https://github.com/mlivesu/slice2mesh`.

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 [10, 11, 12] and mixed element [13, 14] 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 [9] 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 3), therefore such methods cannot be used in this context. Voronoi-based methods such as Tetgen [7] 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 [15, 16, 17] 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 [7] or similar tools. Recent research in the field (e.g., TetWild [8]) has shown that solid meshes can be also constructed starting from defective inputs that do not fulfill the all these requirements. In this revised version we test the capabilities of TetWild to overcome over-refinement issues triggered by tiny details that may be present in our PLCs.

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 [18] for a more comprehensive survey.

Uniform size meshing. Matsumoto and colleagues [19] 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 [18], even for the simulation of residual stresses [20]. Other methods employ discrete rods to simulate the plastic filament of FDM [21] or voxel-based approaches, performing the simulation on regular 3D lattices [22, 23, 24, 25, 26, 27]. 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 [28, 29, 30, 31, 32], 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 [29]. Both [33] and [34] 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 [35] 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 [36] 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 [37] serves to encode sliced data for 3D printers and is supported both from commercial [5] 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 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 [38] 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 3). 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 [7]) to turn our PLC into a tetrahedral mesh.

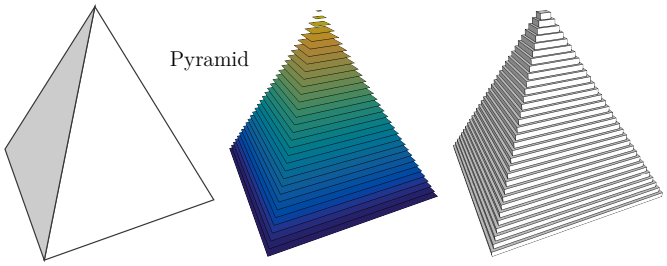

Fig. 4. Slicing a flat surface misaligned with the build direction creates the staircase approximation error typical of additive manufacturing. The staircase effect is lower for nearly vertical surfaces w.r.t. the build direction, and becomes much higher for nearly horizontal surfaces [1].

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 5). 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).

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 6). Note that many popular patterns for support structures are based on intersecting lines (e.g. lattices [1]). 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 [39], 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).

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 [40], 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 8.

5.3. Horizontal meshing

With all the lifting data pre-computed, we can easily proceed with the generation of the PLC. Here we describe 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 [41] 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

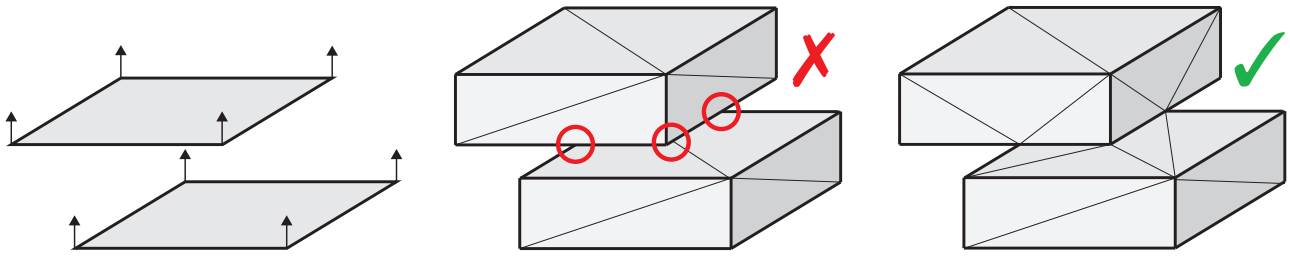

Fig. 5. Extruding each slice along the building direction (left) by an amount corresponding to the local layer thickness produces a digital copy of the object to be printed. If this process is performed locally, each extruded slice will be disconnected from its adjacent slices, thus generating a non-conforming 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).

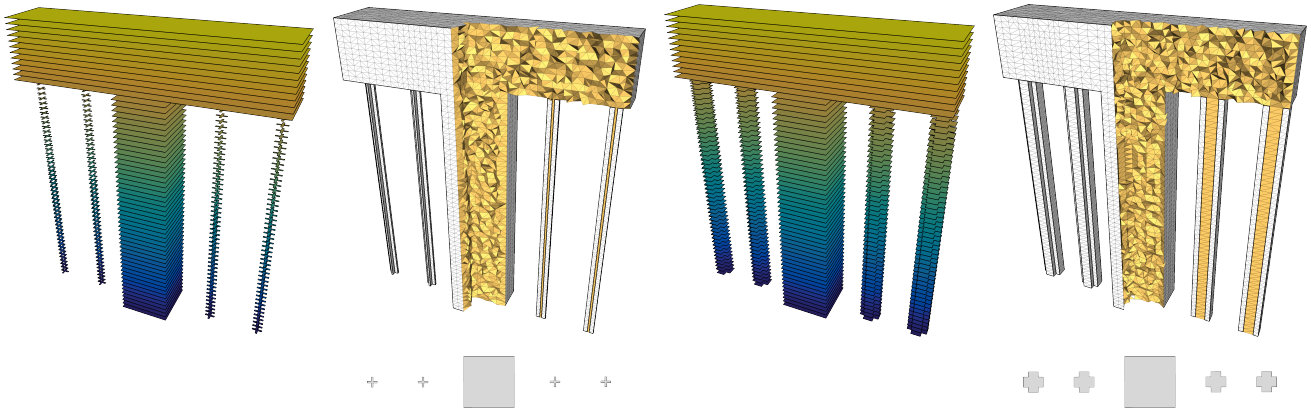

Fig. 6. Two different tetrahedral meshes of a T shape, obtained by using growing thicknesses for supports structures. Supports are often seen by the printer as 1D entities, and their fabricated size depends on hardware (e.g. laser beam or filament diameter). `slice2mesh` can be tuned to generate simulation domains specifically tailored for a particular hardware.

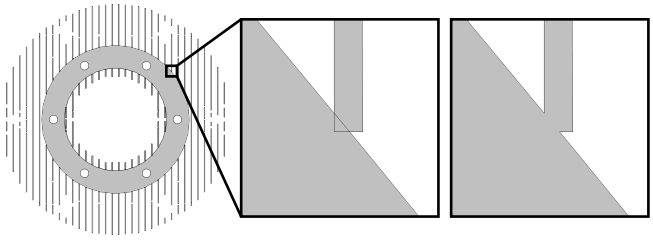

Fig. 7. Details on a slice of the Nugear (with linear supports). Thickening 1D support structures and converting them into polygons may generate intersections with other polygons living in the same slice. We avoid intersections by performing a boolean union of all the polygons in the slice.

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. 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 8. 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.

We point out that not all edges need to be split. For example, consider an object with an internal spherical cavity, and consider two consecutive slices that cut the cavity in its lower hemisphere. In this case, the lower polygon representing the cavity would be entirely contained in the upper polygon, its edges will be simply replicated at the slice above without modifications, and these replicated edges would bound the horizontal triangulation of the upper slice.

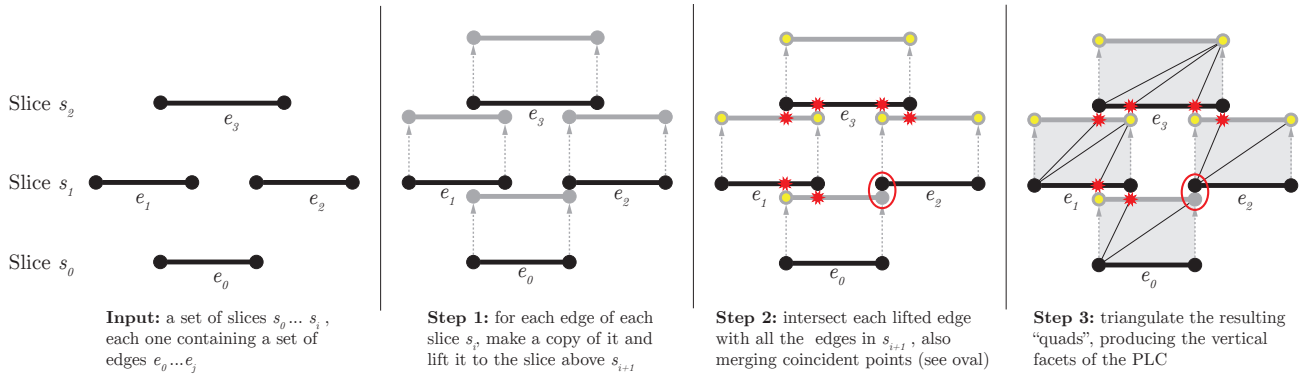

Fig. 8. A schematic 2D representation of our edge lifting strategy. Each edge is lifted from its current slice to the slice above, and tested for intersection and coincidence with all the edges in it. The resulting quad-like domains are eventually triangulated to produce the vertical facets of the PLC (right). Detected intersections are marked with a red star, newly added vertices with a yellow circle.

6. Tetrahedralization

To turn the PLC into a tetrahedral mesh we rely on external software. In our tests we found out that this operation can be extremely complex and result in a number of issues, such as crashes of the meshing tool, runs that last for days, or extremely over refined meshes with unbearable number of elements for any simulator and therefore no practical usefulness. In this section we discuss meshing issues, starting from the challenges that the tetrahedralization of our PLCs pose (Section 6.1), and then analyzing the performances of two popular mesh generation tools, which will be discussed in Section 6.2 and 6.3, respectively. In Table 1 we report numerical statistics about the results produced with both tools.

6.1. Meshing issues

Imagine a perfect cube standing on its base. If printed in this position, all of its slices will be identical. Now imagine to apply some torsion to the cube, rotating its upper face by a small angle. The slices are no longer identical and, if two consecutive slices are projected on a common plane which is parallel to both, eight tiny triangular pockets appear (Figure 9, left). Depending on the amount of torsion and the density of the slicing, the pockets can be arbitrarily small, producing features that are difficult to incorporate in the tetrahedral mesh. Similarly, adjacent slices with nearly tangent contours may generate critical features, as depicted in the right part of Figure 9. Besides the difficulty in embedding these critical features in the tetrahedral mesh, one should recall that a good simulation mesh contains only *well shaped* elements. In practice, this means that additional Steiner points are added in the domain to provide enough degrees of freedom for high quality meshing, and a huge amount of them will be localized nearby these tiny features, to satisfy per-element quality constraints and provide the proper grading with the coarser parts of the PLC (Figure 11, left). The geometries we consider contain lots of tiny features and require a huge number of Steiner points to stay within acceptable quality bounds. To this end, they push state of the art meshing tools to the limits of what they can do, possibly witnessing the need for more research in the field.

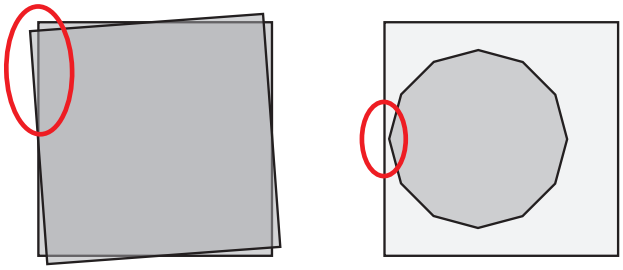

Fig. 9. Left: two almost identical slices project one on top of the other, generating tiny pockets. Right: the contours of two adjacent slices are almost tangent, generating a tiny tunnel (right). These configurations are critical for tetrahedral mesh generation, which may either fail or produce overly complex meshes in order to incorporate such features.

6.2. TetGen

TetGen [7] is a program to compute tetrahedralizations of any 3D domain, and is extremely popular in academia. It works in two steps: it first generates a constrained Delaunay tetrahedralization of the input points and faces, basically meshing the whole convex hull; then, iteratively removes boundary tetrahedra until all the external faces of the PLC are revealed, leaving the inner constraints embedded in the connectivity of the mesh. The skin of the output tetrahedral mesh is therefore a faithful replica of the input; no deviation from it is allowed. In reality, TetGen uses finite precision floating point numbers to express coordinates, and pre processes the input PLC to avoid operations involving quantities that are close to machine precision. Specifically, faces of the PLC that are nearly coplanar are snapped to the same plane, possibly colliding with other geometric elements. It should be noticed that in most of the cases this operation is harmless; however, in our specific setting extremely tiny features abound, and we empirically observed that we could not generate valid tetrahedral meshes without tweaking the threshold for the coplanarity test, which we lowered from the default value of 1×10^{-8} to 1×10^{-13}, using the flag `-T1e-13`. With this tiny modification we were able to solve the first issue stated in Section 6.1, producing coarse tetrahedral meshes with minimum amount of Steiner points for nearly

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 $O(n \log n)$.

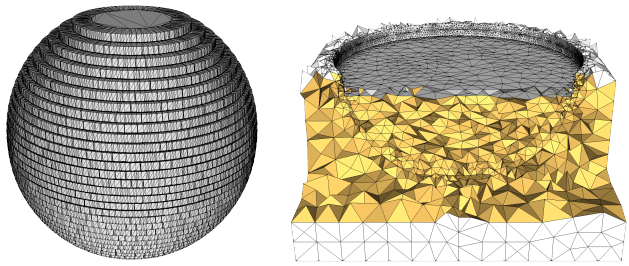

Fig. 10. Left: the PLC of a sphere, produced with `slice2mesh`. The mesh contains elements also in the interior, describing the geometry of each slice. Right: a tetrahedral mesh obtained with TetWild. The interior/exterior filtering based on generalized winding numbers is fooled by our inner structures, and the software is not able to reproduce the sphere.

all the PLCs we have created. The meshes illustrated in the previous version of this article [6] were of this kind. However, we could experience a few cases that made TetGen crash in any case, even if we could verify that the input was perfectly valid.

For the second issue, that is the generation of refined tetrahedral meshes which meet some prescribed per element quality bound, we could not find a satisfactory solution. TetGen allows to prescribe quality criteria with the `-q` flag, but it was able to converge only for a few simple objects, failing for the rest of the dataset. As an example, the PLC of the Fandisk ran for six days on a cluster having 32 Intel Xeon 2.00GHz CPUs and 125 GB of RAM without converging. It is not yet clear whether TetGen was actually adding Steiner points for six days, or it was stuck in an infinite loop because it could not find a proper position (expressed in finite precision coordinates) to insert a new one. Besides the difficulties of using finite precision, indeed, the refinement algorithm used in TetGen is not guaranteed to converge if the input PLC has dihedral angles smaller than 69.3 degrees, unless additional points are carefully placed around these angles. Unfortunately, the number of these additional points can grow arbitrarily [42].

6.3. TetWild

TetWild is a robust tetrahedralization software recently presented in [8]. Differently from TetGen, it makes no assumption on the input, and is capable of turning into a tetrahedral mesh objects that do not unambiguously enclose a solid (i.e., they self-intersect or are not watertight), or that contain topological issues (e.g. non-manifold vertices/edges). In practical terms, TetWild is allowed to deviate from the input PLC, meaning that its faces and vertices are not necessarily embedded in the connectivity of the tetrahedral mesh. Similarly to TetGen, TetWild first generates a mesh of a scaffold containing the input geometry, and then removes the external tets to reveal the shape. In this case, this operation is performed using generalized winding numbers [17], which are used to robustly separate the interior of the object from the outside. As pointed out in Figure 3, our PLCs describe both the outside and the inner structure of the mesh, and involve a number of non-manifold edges. Generalized winding numbers do not perform well in these conditions, and are not able to correctly label inside and outside (Figure 10). We solved this issue by disabling the tet filtering inside TetWild,

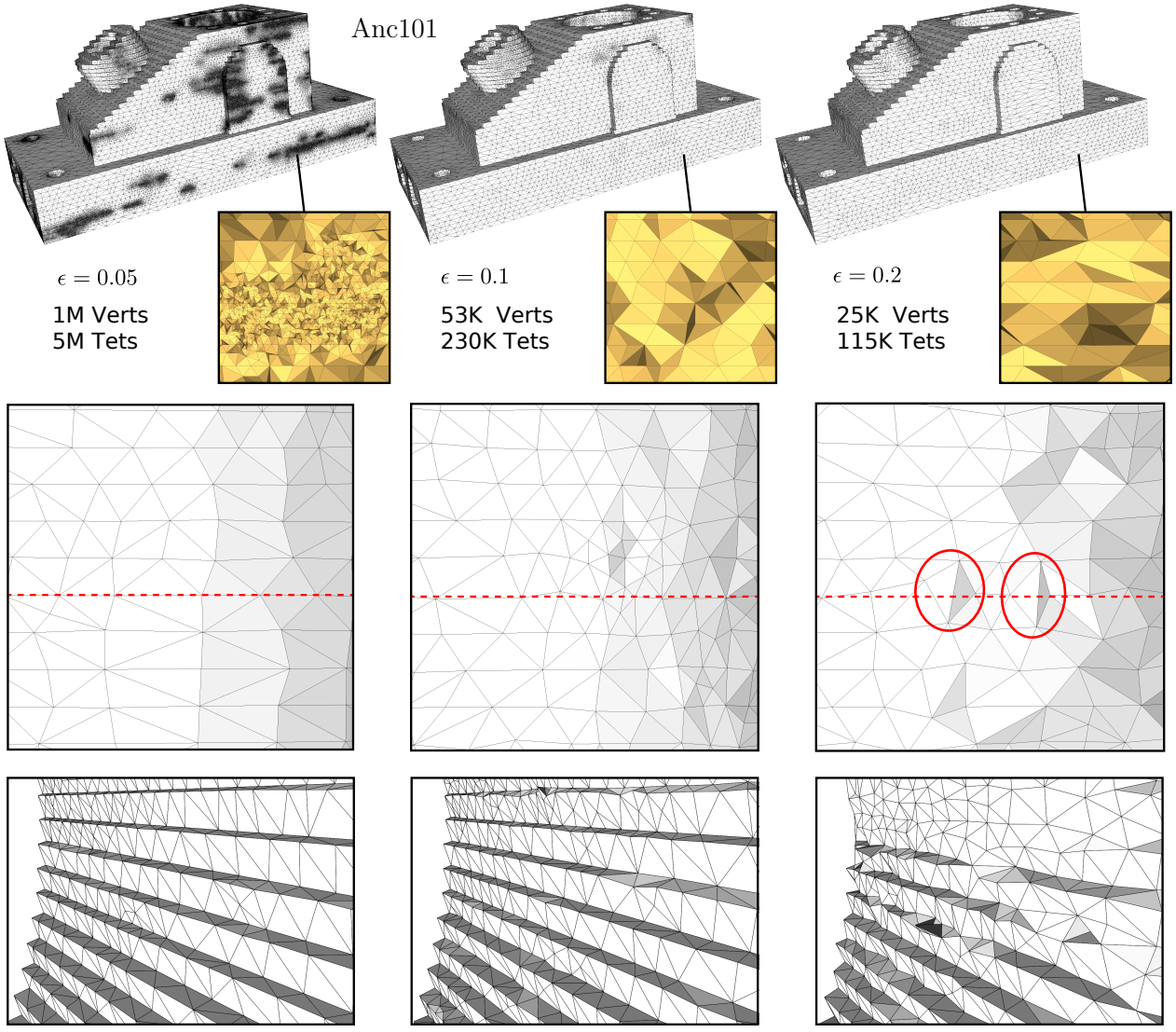

Fig. 11. Three meshes produced with TetWild with growing numbers of parameter ϵ, which controls the deviation from the input PLC. Very tight constraints (left column) produce overly dense clusters of elements nearby tiny features, that the grading pushes also in the interior (yellow inset). Conversely, too loose constraints (right column) produce much coarser meshes, but the edges of the mesh start not to follow the geometry of the slices (red dashed line), and the reproduction of the staircase effect is unsatisfactory (bottom right corner). Values in between (mid column) provide a good balance between element count and geometric fidelity.

and then run it separately using a version of the PLC which does not contain the internal structure of the mesh, and therefore correctly encodes the skin of the mesh we aim to produce.

Our experimentation confirms that TetWild's approach is extremely robust, and allowed us to produce all the refined meshes shown in this article. However, it also reveals an intrinsic meshing problem for which a satisfactory solution is still missing. In fact, its ability to process pathological meshes containing extremely tiny features by allowing a (bounded) deviation from the input geometry comes at a cost: the resulting mesh is no longer conformal with the input PLC. In Figure 11 we show different meshes obtained by enabling growing deviations from the input geometry. This is controlled by parameter ϵ, which sets a local deviation tolerance as a fraction of the diagonal of the bounding box of the model. As one can notice, such deviations can produce tets that span across adjacent slices (red ovals), and deviate from the input geometry in bizarre ways, even if they stay within the prescribed tolerance (bottom right corner). Tetrahedra may cross adjacent slices either in the interior or on the boundary. The latter case is typically worse because, besides spoiling slice conformity, we horizontally deviate from the only accurate information we have, that is, the input slice geometry. Fortunately, our experiments show that such a deviation is always smaller than ϵ in practice, whereas the average horizontal deviation is orders of magnitude smaller (see Figure 14). Surface conformity is maintained during the first phase only, when TetWild uses exact arithmetic to internally represent the initial unrefined mesh. If such a conformal mesh is needed, one may think of just stopping TetWild before the refinement stage, but this requires to approximate exact coordinates to make the mesh suitable for downstream applications. Unfortunately, on domains with very small features, such

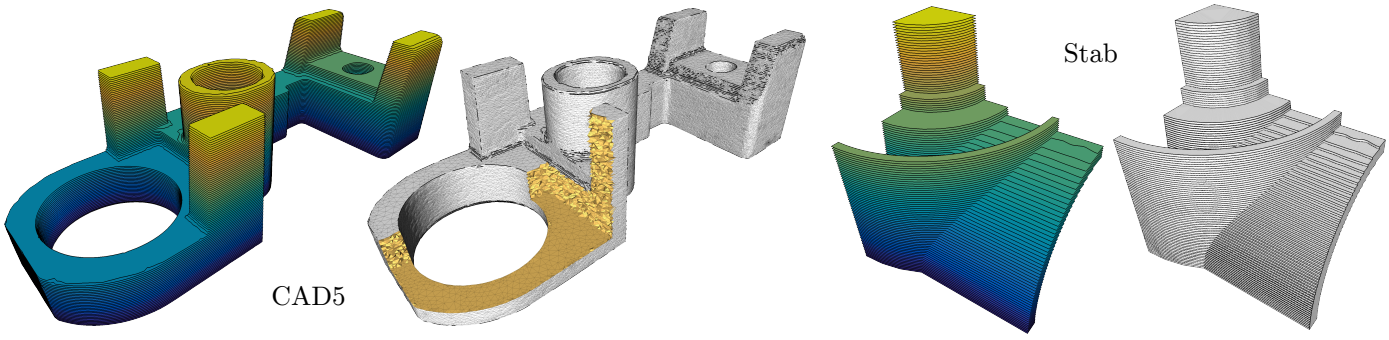

Fig. 12. A gallery of sliced CAD models meshed with `slice2mesh`.

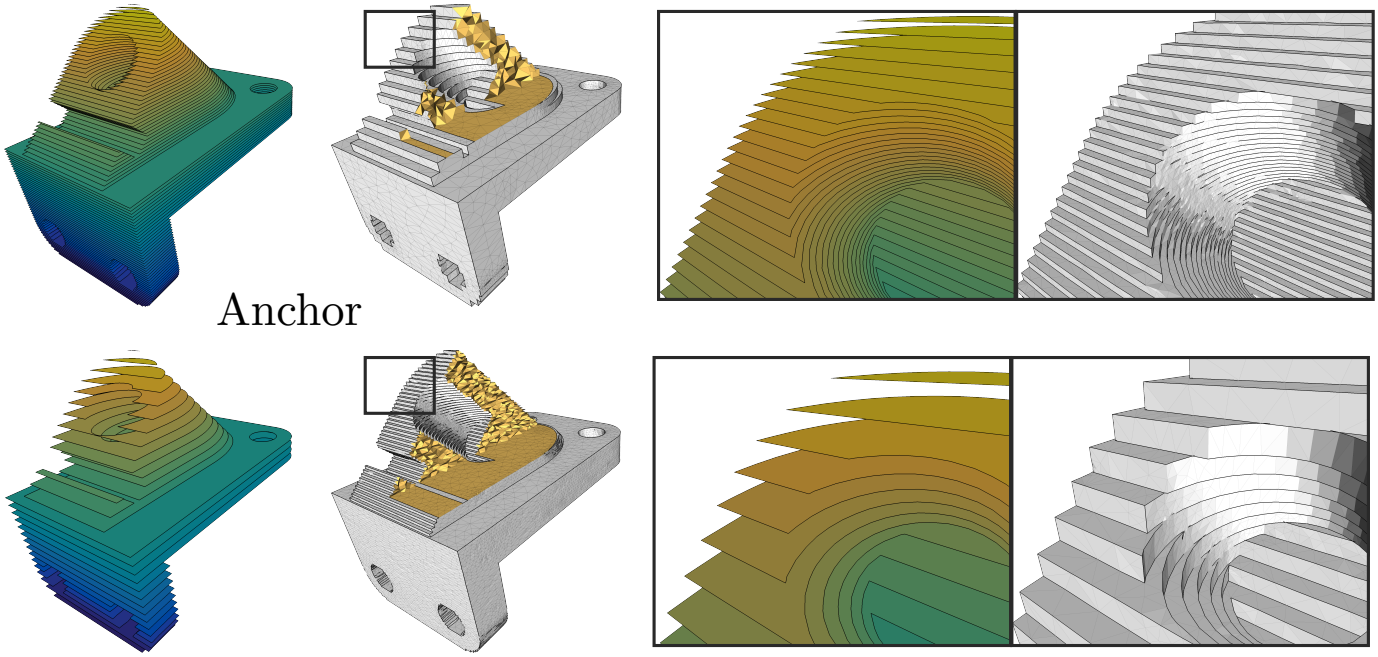

Fig. 13. Two different slicings for the Anchor model, with their associated volumetric meshes. Right: a finer slicing better approximates off-axis features.

a rounding may easily lead to flipped tetrahedra. To the best of our knowledge, robust snap rounding in 3D is still a challenging problem. Some theoretical solutions have been recently proposed [43], but their complexity is too high to be implemented and used in practice.

7. Results

We implemented `slice2mesh` using CinoLib [38] for CLI and geometry processing, the Boost Polygon Library for curve thickening and 2D booleans, Triangle [41] for planar triangulations, and either Tetgen [7] or TetWild [8] for the final tetrahedralization. For the meshes obtained with TetWild, the final inside/outside tet filtering was computed in post-processing on a modified version of the PLC which does not contain the internal slices, using the implementation of the generalized winding numbers available in libIGL [45]. In Figures 1, 13 and 12 we show a variety of results. In Table 1 and Figures 15 and 14 we report statistics and timings of our experiments. For each set of slices, we considered two alternative meshing settings for Tetgen (with/without the `-q` flag for quality refinement), and three settings for Tetwild ($\epsilon = 0.05, 0.1, 0.2$). As can be noticed from the table, only a few models were tested on TetGen with the mesh refinement enabled, as the program either crashed or ran for multiple days without producing a mesh. 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 18). Note however that the surface exposed by this naive mesh filtering will be much higher, possibly resulting in unfaithful estimation of physical phenomena such as heat dissipation, where the augmented surface area artificially

Model	Slices	Tetgen		TetWild					
		`-T1e-13`	`-T1e-13q`	ϵ = 0.05		ϵ = 0.1		ϵ = 0.2	
		#V/#T	#V/#T	#V/#T	q	#V/#T	q	#V/#T	q
Anc101	38	16K/58K	2.3M/8.7M	1M/5M	1.53/10.62	53K/230K	1.46/12.63	25K/115K	1.30/345
Anchor	86	33K/110K	crashed	330K/1.6M	1.68/7.58	45K/209K	1.64/8.33	38K/178K	1.35/9.53
	29	3.3K/11K	4.3M/16.2M	28K/121K	1.41/15.74	12K/50K	1.4/10.12	7K/27K	1.43/6.16
CAD5	77	148K/492K	crashed	735K/3.7M	1.35/12.42	148K/719K	1.66/8.91	129K/631K	1.71/15.37
Fandisk	35	6K/8.5K	stopped after 6 days	418K/2.1M	1.31/13.02	39K/167K	1.51/8.2	23K/105K	1.51/12.04
Joint	83	11K/34K	not tested	206K/950K	1.77/13.8	110K/535K	1.56/17.09	93K/460K	1.35/34.05
	31	2K/7K	not tested	62K/274K	1.4/12.16	28K/123K	1.37/11.8	17K/75K	1.57/16.12
Nugear	88	200K/620K	not tested	not tested		not tested		not tested	
	19	5K/14K	20K/110K	22K/92K	1.23/10.98	19K/78K	1.55/7.98	16K/66K	1.28/9.87
Pyramid	33	2K/6K	not tested	6K/25K	1.36/15.44	7K/29K	1.43/18.23	6K/25K	1.66/17.87
Sphere	83	17K/55K	not tested	230K/1.1M	1.43/8.91	90K/418K	1.5/9.99	64K/310K	1.47/12.72
	33	4K/12K	not tested	40K/174K	1.87/17.11	24K/103K	1.67/23.12	16K/73K	1.34/16.66
Stab	95	74K/239K	not tested	not tested		not tested		not tested	
T	60	not tested	2K/6K	10K/30K	1.67/9.91	12K/38K	1.68/14.03	10K/30K	1.52/15.11

Table 1. Statistics for tetrahedral mesh generation. We tested TetGen [7] with and without the `-q` flag for quality mesh generation, and also TetWild [8], with growing numbers for parameter ϵ, which controls deviation from the input PLC. We report number of vertices (#V), tets (#T), and average/worst element quality (q, we used the radius ratio metric, as defined in [44]). Mesh quality for tetrahedral meshes produced with TetGen was always rather poor ($> 10^{10}$): indeed, the input PLC typically contains arbitrarily bad shaped triangles and, since TetGen exactly conforms to such a PLC, badly shaped tetrahedra cannot be avoided.

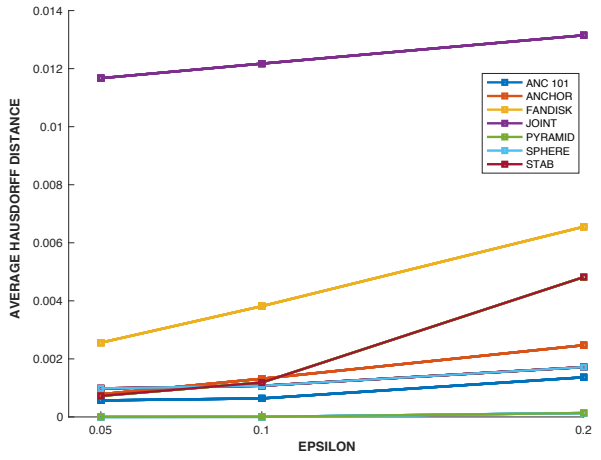

Fig. 14. Approximation error introduced by TetWild with growing levels of ϵ. Error is obtained by averaging the per polygon Hausdorff distance between the input slices and a slicing of the tetrahedral meshes. Note that inter slice error grows linearly with ϵ and is always smaller than ϵ itself.

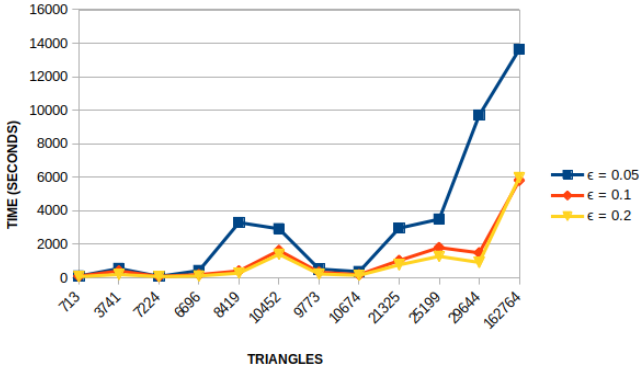

Fig. 15. Running times of `slice2mesh`. Tetrahedral meshes were generated using TetWild, considering different values for the ϵ parameter, which sets the maximum distance from the nominal geometry. The horizontal axis reports the complexity of the input, expressed in number of triangles in the PLC. Tetrahedralization software was run on a cluster equipped with 32 Intel Xeon 2.00GHz CPUs and 125 GB of RAM.

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 6 and 17). 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

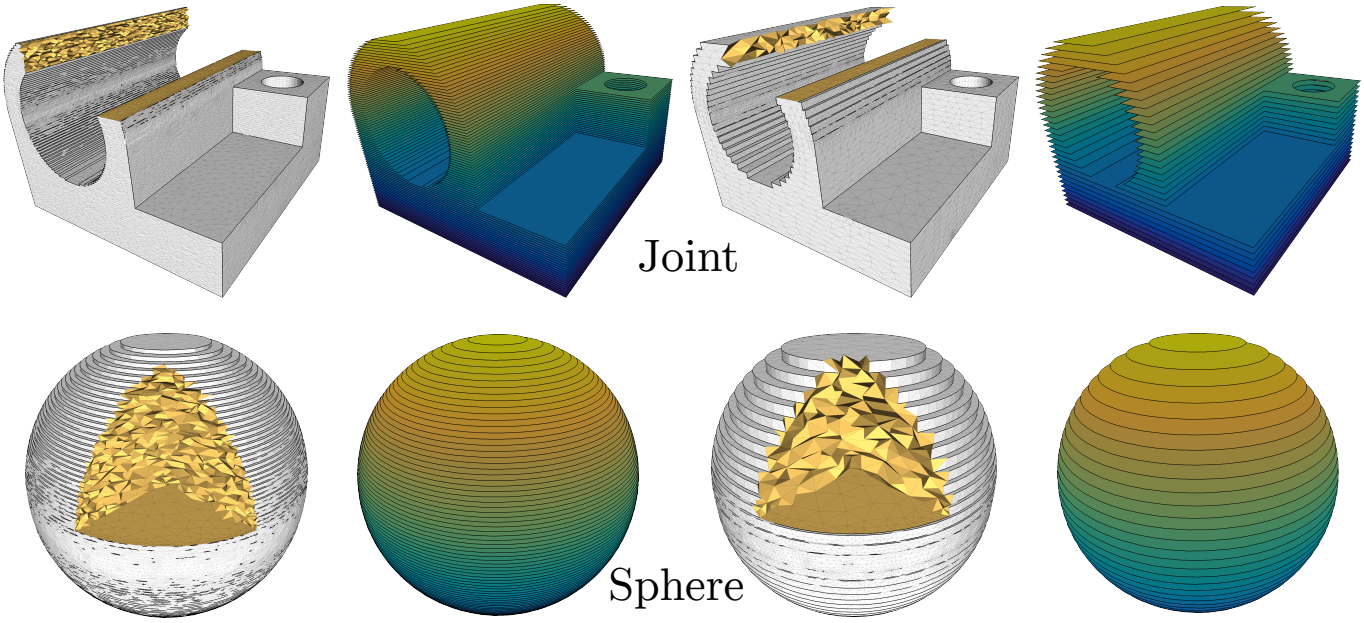

Fig. 16. Precisely simulating the activity of industrial printers may be too ambitious, due to the extremely thin slices they use. `slice2mesh` can generate approximate representations of the simulation domain by filtering the shape in slice space (i.e. sub-sampling the input slices). This allows to perform approximate simulations, in which bundles o multiple adjacent slices are activated at at each time step.

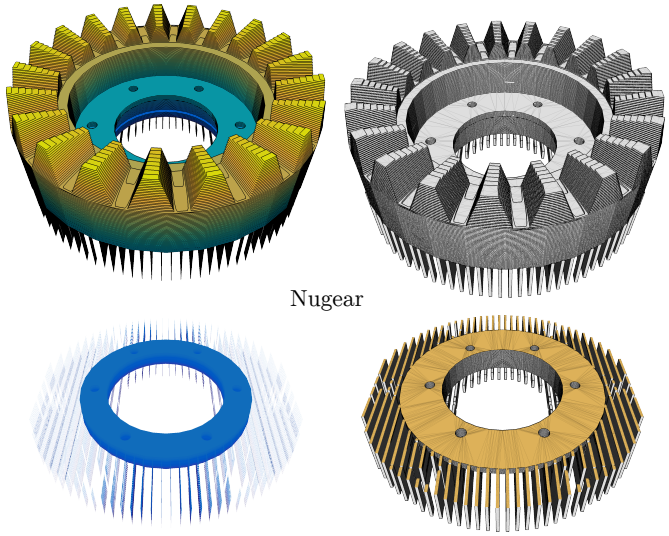

Fig. 17. An example of volumetric mesh which includes both the object and its thickened support structures. Matching the thickening factor with the specifications of the printer (e.g. filament thickness or laser beam) the user can generate simulation domains which are tailored for a specific hardware.

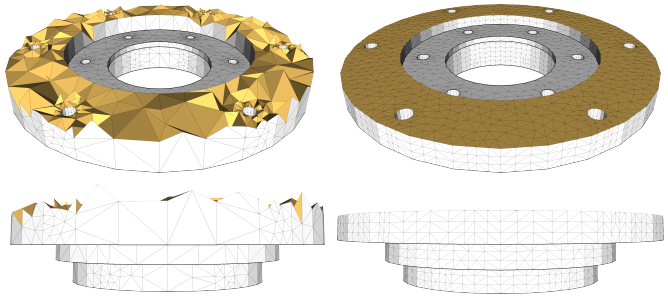

Fig. 18. Left: a general tetrahedral mesh, where all the elements having their centroid above the current slice have been filtered out. Right: our slice-conformal tetrahedral mesh. Simulations of physical phenomena such as heat dissipation may be unreliable due to the area of the exposed surface, which is artificially increased in the left example (1308.54mm^2 vs 778.23mm^2).

(see models with two line entries). Visual representations of the so generated domains are given in Figures 13 and 16.

PLC. Besides simulation, the PLC has its own value and could be used for other AM-related purposes. Having a digital representation 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 [2, 1]. 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.

We point out that `slice2mesh` is not a surface reconstruction algorithm, but rather a tool to produce a simulation domain that is a digital replica of what a 3D printer would produce out of a given set of slices. Neither `slice2mesh` nor the 3D printer are aware of the original 3D model. If this model is poorly sliced, possible defects would be incorporated both in the mesh produced by `slice2mesh` and in the physical object produced by the printer.

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 demonstrated the capabilities of our tool on a number of results, also showing some useful features such as a flexible meshing system for support structures and a robust mesh simplification tool that operates directly in slice space.

In this extended version we also discuss meshing issues that arose during our experimentation. At the moment mesh generation heavily relies on the robustness of TetWild [8], which was used to produce the vast majority of the refined meshes shown in this paper. As discussed in Section 6.3, robustness comes at the cost of non conformity between the input PLC and the resulting volumetric mesh. We argue that better trade-offs between surface conformity and mesh quality can be found, and our current research is focused on this aspects.

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. Though our preliminary tests are promising, they also revealed that some fabrication scenarios employ an extremely small layer thickness which leads to a huge number of tetrahedra. Thus, while our method can be readily used for blown-powder technologies, where the typical layer thickness is about 200-300 microns, it is not appropriate for powder-bed laser sintering, where the thickness can drop down to less than 30 microns. Besides the layer sub-sampling discussed in Sect. 7, we are currently investigating approaches to (1) parallelize the FEA itself, and (2) create of a multi-resolution version of our dense mesh that adapts as the simulation proceeds (i.e. full resolution only near the *active* layers).

Other interesting venues for future research involve the generation of hexahedral or mixed element meshes to further reduce element count (or obtain more accurate simulations for same mesh complexity), and the ability to incorporate in the mesh machine toolpaths. Machine toolpaths are usually encoded in the hatches of the CLI files, and `slice2mesh` currently discards them. This would allow to faithfully simulate the printing process, activating not only one slice at a time, but literally following element by element the actual deposition process. Meshes of this kind may have interesting applications in aerospace (e.g. to study meta-materials). To this end, the biggest challenge we foresee is the ability to bound element count, which may explode and result in too complex meshes with no practical usefulness.

Acknowledgements

We are grateful to Daniele Panozzo and Yixin-Hu for insightful discussions on TetWild, and for promptly editing their code to allow us to optionally disable the inside outside filtering from command line. This project has received funding from the European Union Horizon 2020 research and innovation program, under grant agreement No.680448 (CAxMan).

References

[1] 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 2017;36(2):537–564. doi:10.1111/cgf.13147.

[2] Attene, M, Livesu, M, Lefebvre, S, Funkhouser, T, Rusinkiewicz, S, Ellero, S, et al. Design, representations, and processing for additive manufacturing. Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging 2018;10(2):1–146.

[3] Montevecchi, F, Venturini, G, Scippa, A, Campatelli, G. Finite element modelling of wire-arc-additive-manufacturing process. Procedia CIRP 2016;55:109–114.

[4] Hiller, J, Lipson, H. Dynamic simulation of soft multimaterial 3d-printed objects. Soft robotics 2014;1(1):88–101.

[5] Materialise, . Magics. `https://www.materialise.com/en/software/magics`; ????

[6] Livesu, M, Cabiddu, D, Attene, M. Slice2Mesh: meshing sliced data for the simulation of AM Processes. In: Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference. The Eurographics Association; 2018,.

[7] Si, H. Tetgen, a delaunay-based quality tetrahedral mesh generator. ACM Transactions on Mathematical Software (TOMS) 2015;41(2):11.

[8] Hu, Y, Zhou, Q, Gao, X, Jacobson, A, Zorin, D, Panozzo, D. Tetwild - tetrahedral meshing in the wild. ACM Transactions on Graphics (SIGGRAPH 2018) 2018;(to appear).

[9] Alliez, P, Cohen-Steiner, D, Yvinec, M, Desbrun, M. Variational tetrahedral meshing. ACM Transactions on Graphics (TOG) 2005;24(3):617–625.

[10] Fang, X, Xu, W, Bao, H, Huang, J. All-hex meshing using closed-form induced polycube. ACM Transactions on Graphics (TOG) 2016;35(4):124.

[11] Livesu, M, Vining, N, Sheffer, A, Gregson, J, Scateni, R. Polycut: monotone graph-cuts for polycube base-complex construction. ACM Transactions on Graphics (TOG) 2013;32(6):171.

[12] Li, Y, Liu, Y, Xu, W, Wang, W, Guo, B. All-hex meshing using singularity-restricted field. ACM Transactions on Graphics (TOG) 2012;31(6):177.

[13] Gao, X, Jakob, W, Tarini, M, Panozzo, D. Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. ACM Transactions on Graphics (TOG) 2017;36(4):114.

[14] Sokolov, D, Ray, N, Untereiner, L, Lévy, B. Hexahedral-dominant meshing. ACM Transactions on Graphics (TOG) 2016;35(5):157.

[15] Zhou, Q, Grinspun, E, Zorin, D, Jacobson, A. Mesh arrangements for solid geometry. ACM Transactions on Graphics (TOG) 2016;35(4).

[16] Attene, M, Campen, M, Kobbelt, L. Polygon mesh repairing: An application perspective. ACM Computing Surveys (CSUR) 2013;45(2):15.

[17] Jacobson, A, Kavan, L, Sorkine-Hornung, O. Robust inside-outside segmentation using generalized winding numbers. ACM Transactions on Graphics (TOG) 2013;32(4):33.

[18] 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.

[19] 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 2002;42(1):61–67.

[20] Van Belle, L, Vansteenkiste, G, Boyer, JC. Comparisons of numerical modelling of the selective laser melting. In: Key Engineering Materials; vol. 504. Trans Tech Publ; 2012, p. 1067–1072.

[21] Sheth, S, Taylor, RM, 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,.

[22] 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, p. 386–392.
[23] Dai, K, Shaw, L. Thermal and stress modeling of multi-material laser processing. Acta Materialia 2001;49(20):4171–4181.
[24] Dai, K, Shaw, L. Distortion minimization of laser-processed components through control of laser scanning patterns. Rapid Prototyping Journal 2002;8(5):270–276.
[25] 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 2003;34(5):1133–1145.
[26] Ma, L, Bin, H. Temperature and stress analysis and simulation in fractal scanning-based laser sintering. The International Journal of Advanced Manufacturing Technology 2007;34(9-10):898–903.
[27] Antony, K, Arivazhagan, N, Senthilkumaran, K. Numerical and experimental investigations on laser melting of stainless steel 316l metal powders. Journal of Manufacturing Processes 2014;16(3):345–355.
[28] Yang, Q, Zhang, P, Cheng, L, Min, Z, Chyu, M, To, AC. Finite element modeling and validation of thermomechanical behavior of ti-6al-4v in directed energy deposition additive manufacturing. Additive Manufacturing 2016;.
[29] Heigel, J, Michaleris, P, Reutzel, E. Thermo-mechanical model development and validation of directed energy deposition additive manufacturing of ti–6al–4v. Additive Manufacturing 2015;5:9–19.
[30] de Saracibar, CA, Lundbäck, A, Chiumenti, M, Cervera, M. Shaped metal deposition processes. In: Encyclopedia of Thermal Stresses. Springer; 2014, p. 4346–4355.
[31] Contuzzi, N, Campanelli, S, Ludovico, A. 3 d finite element analysis in the selective laser melting process. International Journal of Simulation Modelling 2011;10(3):113–121.
[32] 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 2013;52:638–647.
[33] Riedlbauer, D, Steinmann, P, Mergheim, J. Thermomechanical finite element simulations of selective electron beam melting processes: performance considerations. Computational Mechanics 2014;54(1):109–122.
[34] 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 2004;44(2):117–123.
[35] 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 2012;212(10):2058–2065.
[36] 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 2010;51(5-8):649–658.
[37] Common layer interface. `http://www.hmilch.net/downloads/cli_format.html`; 1993.
[38] Livesu, M. cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. 2017. Https://github.com/mlivesu/cinolib/.
[39] Douglas, DH, Peucker, TK. 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 1973;10(2):112–122.
[40] Shewchuk, JR. Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete & Computational Geometry 1997;18(3):305–363.
[41] Shewchuk, JR. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: Applied computational geometry towards geometric engineering. Springer; 1996, p. 203–222.
[42] Si, H. Three dimensional boundary conforming delaunay mesh generation. Ph.D. thesis; TU Berlin; Germany; 2008.
[43] Devillers, O, Lazard, S, Lenhart, WJ. 3D Snap Rounding. In: Speckmann, B, Tóth, CD, editors. 34th International Symposium on Computational Geometry (SoCG 2018); vol. 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-066-8; 2018, p. 30:1–30:14. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/8743`. doi:`10.4230/LIPIcs.SoCG.2018.30`.
[44] Stimpson, C, Ernst, C, Knupp, P, Pébay, P, Thompson, D. The verdict library reference manual. Sandia National Laboratories Technical Report 2007;9.
[45] Jacobson, A, Panozzo, D, et al. libigl: A simple C++ geometry processing library. 2018. Http://libigl.github.io/libigl/.