IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 7, 2021, accepted July 28, 2021, date of publication July 30, 2021, date of current version August 6, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3101365*

# Toward a Simple Design and Manufacturing Pipeline for Additive Manufacturing

**VAHID HASELTALAB[1], MELIK DOLEN[1], ULAS YAMAN[1], (Member, IEEE), AND CHRISTOPH HOFFMANN[2]**
[1]Department of Mechanical Engineering, Middle East Technical University, 06800 Ankara, Turkey
[2]Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA

Corresponding author: Ulas Yaman (uyaman@metu.edu.tr)

This work was supported in part by the Scientific and Technological Research Council of Turkey under Project 116E882 and Project 117M429.

**ABSTRACT** A novel design and manufacturing pipeline for Additive Manufacturing is presented. The architecture of the pipeline is motivated by the observation that the conventional pipeline is unnecessarily complex. Most of the time, only a small set of programming steps suffices for 3D design and manufacture. In particular, the proposed method requires no complex hardware or software, and it generates the geometric data on the fly. This is demonstrated using a simplified evaluation of general volumetric sweeps. The method side-steps many of the problems of the conventional Additive Manufacturing pipeline. Several scripts are provided that illustrate the capabilities and the advantages of the proposed approach. These scripts are processed on a single-board computer and the parts are manufactured on a Fused Filament Fabrication type of 3D printer.

**INDEX TERMS** Additive manufacturing, implicit modeling, slicing.

## I. INTRODUCTION

With the advent of **Additive Manufacturing** (AM), the expectation of achieving democratic manufacturing has come closer to reality. In the current state-of-the art, AM still relies on sophisticated software systems to create data/instructions needed to drive the fabrication hardware. From the standpoint of prospective manufacturers, the required know-how (including mastering the design/development tools) along with the corresponding limitations imposed by the software tools hamper the rapid product-development cycle. Consequently, we seek to simplify the AM fabrication pipeline by eliminating the need for complex propriety/non-propriety software tools.

The conventional design and fabrication pipeline of AM machinery is summarized in Figure 1. This pipeline commonly uses two separate computer systems. Product design and slicing are generally performed on a workstation or a PC with considerable hardware resources. Commercial **Computer Aided Design** (CAD) software tools are commonly utilized to develop 3D solid geometric models of products. Subsequently, these entities are converted to different geometric representations (such as STL, OBJ, AMF, 3MF, etc.) to be later transferred to proprietary **Computer Aided Manufacturing** (CAM) software. For **Powder Bed Fusion** (PBF) / **Fused Filament Fabrication** (FFF) / **Direct Energy Deposition** (DED) / **Stereolithography** (SLA) / **Digital Light Processing** (DLP) type of devices, the given geometry is transformed into the tool-trajectory data (i.e. position of the laser beam / extruder head / welding electrode etc. defined as a collection of rectilinear motion patches). At this stage, the resulting code (typically NC/G code of RS274D) needs to be further processed by the embedded computer system (a.k.a. 3D-Printer Processor) of the AM machinery. It is parsed and interpolated to produce the low-level instructions (i.e. discrete-time motion- and process control commands) needed to drive the axis controllers and other **Peripheral Devices** (PDs) of the fabrication equipment in real-time.

Although this conventional pipeline has been embraced by the AM community, it has many well-known drawbacks:

- Performance of the pipeline depends on the capabilities of CAD and CAM software tools utilized.
- The host PC should have considerable hardware resources including high-performance **Graphics Processing Unit** (GPU) along with multi-core

The associate editor coordinating the review of this manuscript and approving it for publication was Ming Luo.

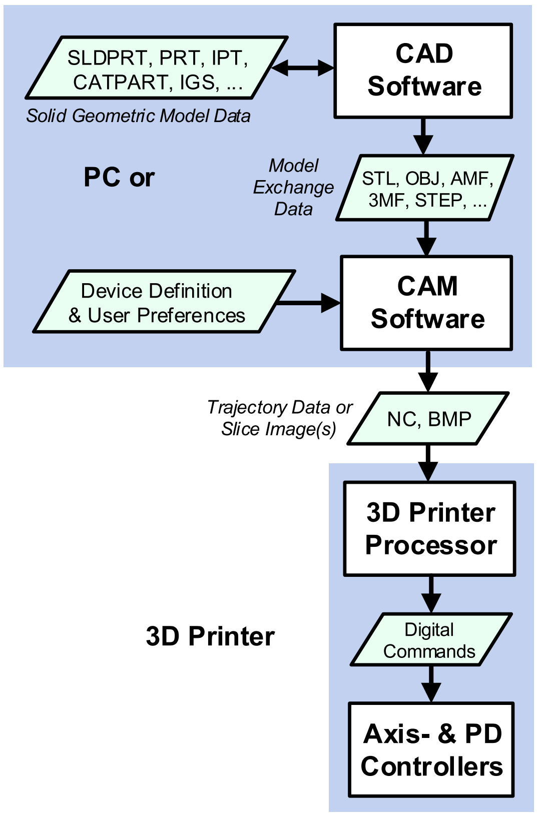

SLDPRT, PRT, IPT, CATPART, IGS, ...
Solid Geometric Model Data
CAD Software
PC or
Model Exchange Data
STL, OBJ, AMF, 3MF, STEP, ...
Device Definition & User Preferences
CAM Software
Trajectory Data or Slice Image(s)
NC, BMP
3D Printer Processor
3D Printer
Digital Commands
Axis- & PD Controllers

**FIGURE 1.** **Conventional design and fabrication pipeline of AM machinery.**

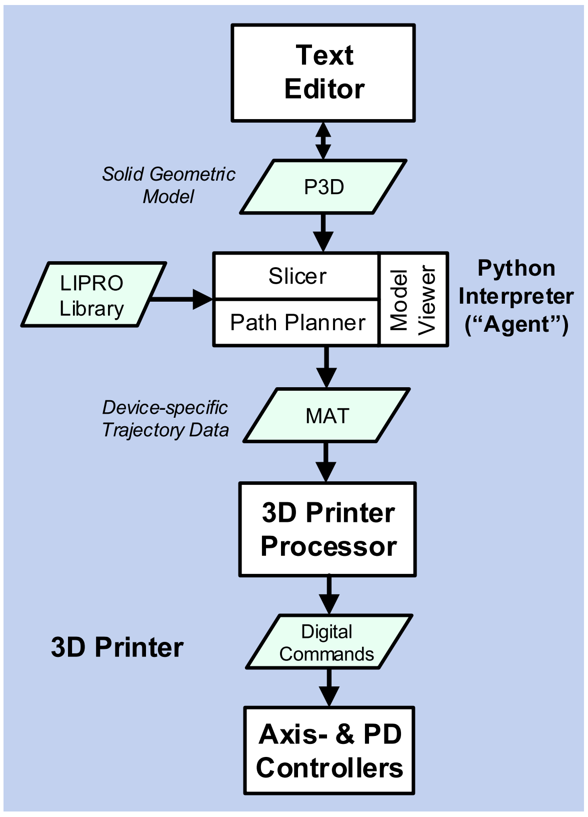


**FIGURE 2.** **Proposed design and fabrication pipeline of AM machinery.**

**Central Processing Unit** (CPU). This is of particular importance if the geometric models constitute detailed features such as intricate parts, surface textures/patterns, lattice structures, etc.

- Data conversion of the designed artifact to STL, OBJ, AMF, 3MF formats inflicts information loss (e.g. dimensional resolution, topological relationships among geometric features, material data, color, interior structure, etc.).
- The pipeline cannot be interrupted. If any revision on the geometric model is required, the whole process has to be started from the scratch.
- The overall pipeline is hard to maintain especially when upgrading to newer or different software tools (with additional process steps).
- The NC code generated by (either proprietary or open-source) CAM software is not universal. The code cannot be processed by any similar AM machine of the same technological class.

An alternative approach, which is presented in Figure 2, aims to overcome the above-mentioned disadvantages by revisiting the basics. In the proposed methodology, a single-board (i.e. small-form factor) computer to be integrated to the 3D-Printer Processor is utilized to perform the entire design- and fabrication stages of AM applications. No specific CAD and/or CAM software is required. Instead, the geometry of a fabricated object is implicitly modeled via a Python script/code called P3D. Slicing and tool-path generation are automatically performed with the assistance of a software agent that is designed for a specific AM technology such as FFF, PBF, DED, SLA/DLP, etc.

This paper elaborates the above-mentioned **integrated** pipeline. The key element of this novel approach is the simplified model definition where the components of a 3D geometric model are constructed in a piecewise fashion by calling a simple **sweep** function. Here, the spatial / topological relationships among these components are described by the operators of **Constructive Solid Geometry** (CSG). Users can build complex geometric parts by embedding a number of **sweep** functions into the P3D Python script. Consequently, the input to the proposed AM pipeline is an **executable** P3D Python script that defines not only the geometry but also the relevant attributes of the solid part. Note that the shape to be fabricated does not involve proprietary data as would be necessary if the description had to rely on commercial CAD/CAM systems.

The paper presents a software agent (also implemented in Python programming language) to slice the model and generate directly tool trajectory data without user intervention. The approach has been implemented on a small-form factor computer system (Raspberry Pi Model 3).

The outline of the paper is as follows: Background information about the recent developments on implicit modeling, slicing, and AM pipeline is presented in Section 2.

The structure of the proposed approach is elaborated in Section 3. To assess the performance and benefits of the proposed scheme, several scripts are developed to model intricate parts. The details of these scripts along with the fabricated artefacts are given in Section 4 where comparisons with the conventional approaches are also done. Finally, summary and conclusion on key aspects of the work are presented in the last section.

## II. PRIOR WORK

The AM processes provide more flexibility than the traditional manufacturing methods but they still require careful planning [1]. There is a basic procedure for almost all AM processes from design to fabrication. Throughout the years, many research efforts have concentrated to this aspect but no efficient pipeline has been introduced to overcome all pronounced problems. Since our proposed method focuses on design and slicing perspective of the AM pipeline, recent literature on these topics is summarized in proceeding subsections.

### A. EXPLICIT MODELING AND SLICING

Explicit modeling and slicing methods can be considered as foremost approaches in computer graphics and AM. The main methods to represent artifacts include CSG, (triangle /polygon) meshes, **Boundary REPresentations** (BREP), voxels, and point clouds. A succinct review on these approaches can be found in [2]. Since these representation methods are not specifically developed within the context of AM, there exist many studies on customizing them for AM. In one of these studies, Chandru *et al.* [3] used voxels to perform design operations in AM. Similar to our study, they employed volumetric sweeps as the primitives for their geometric framework.

Slicing shapes that are defined explicitly is a fundamental research field, because they all affect the tool trajectories in AM machinery. Direct slicing is oftentimes coupled to CSG representations. Once the part is modeled by the corresponding CAD software, special scripts/programs are devised to carry out the slicing directly on the shape so as to increase the accuracy of path generation (see [4]). Likewise, there are numerous studies for slicing in conventional AM pipelines that employ mesh representations. For instance, Minetto *et al.* [5] proposed an optimal slicing algorithm to decrease the computation time, especially when the number of facets in the mesh is relatively large ($\gg 1000$).

A different approach based on BREP was adopted by Starly *et al.* [6]. They extracted the boundary information from STEP files and then sliced the models using ray-casting method. Slicing becomes straightforward when voxel-based modeling is employed. Alexa *et al.* [7] focused on the voxelization side to decrease the computation time while increasing the fabrication accuracy. Point-cloud representations are also utilized when the AM process needs to duplicate an artifact. Xu *et al.* [8] considered slicing the point-cloud based models. They constructed virtual edges of the given data and then the model was sliced directly without any conversion to different geometric representations.

### B. IMPLICIT MODELING AND SLICING

Although explicit modeling is utilized by most of the AM community, it brings some disadvantages to the design and fabrication pipeline. This is mainly due to the fact that in explicit modeling, parts are modeled as solids with homogeneous interior. When such approaches are adapted, the true potential of AM technologies cannot be unleashed. Thus, many studies investigate implicit modeling techniques in AM. In one of these works, Lefebvre [9] proposed an implicit modeling and slicing tool (IceSL) for FFF-based 3D printers. In this approach, Lua-based scripts were utilized to model geometric entities via CSG. The script, which incorporates both analytical primitives (i.e. cones, spheres, cubes, etc.) and the STL models, was processed with the full assistance of a GPU. The graphics rendering, CSG operations on the geometric entities (defined in the script), and the subsequent slicing operations were all performed at the maximum resolution of the 3D printer by making good use of the A-Buffer technique. Consequently, fabricated artefacts were expected to be high-fidelity replicas of the original models. Notice that the model visualization and slicing in this work took place at very high speeds owing to the fact that all operations were carried out (in parallel) on the GPU.

Instead of CSG, Zhao *et al.* [10] used **Layer-Depth Normal Images** (LDNI) to define models implicitly. Using Boolean operations, they improved the efficiency of truss generation. The study of Zeng *et al.* [11] can be considered an extension of the work of Zhao *et al.* [10]. They utilized Boolean expressions of LDNI solids to enhance the efficiency of slicing on complicated CSG objects. In another study, Steuben *et al.* [12] provided an implicit slicing method for evaluating the tool-paths for each layer. They further examined the effect of infill patterns on the performance of different models in terms of their stress- and strain distributions. They eventually compared them to the outcomes of conventional (explicit modelling) methods. The final test in their study showed a great improvement on the structural integrity of their specimens. A similar study was conducted by Xia *et al.* [13] to define the interior of the parts implicitly by utilizing the orientations of the maximum principal stress components. Li *et al.* [14] followed a similar approach. They used several implicit functions to model the parts and fabricated them directly on 3D printers. They highlighted the resulting advantages of implicit modeling and slicing in AM.

Considering the above-mentioned research efforts in the conventional design and fabrication pipeline of AM technologies, there remain open relevant research areas. In particular, mitigating or eliminating the disadvantages enumerated before awaits further work. This paper aims at doing that in the framework of implicit models and low processing power.

## III. PROPOSED METHOD

The proposed pipeline combines three major tasks:

(i) The artifact to be fabricated is modeled implicitly in Python using CSG augmented by our library functions that are specifically developed for efficient geometric modeling.

(ii) The cross-sections of the solid model, at arbitrary elevations, can be obtained directly using the P3D script defining the model.

(iii) A universal AM agent generates automatically the interpolation data needed to fabricate each- and every layer to be printed.

The following sub-sections elaborate these methods.

### A. CONSTRUCTION OF SOLID GEOMETRIC MODELS

Swept volumes are ubiquitous tools for generating numerous features. Thus, there is a large body of research that examines this concept [15]. Moreover, almost every CAD software offers a method/utility/function/operator to generate such solid geometric models. The method we offer here relies on a single (but general-purpose) **sweep** function that creates the boundaries/surfaces of solid geometric models on a piecewise fashion. User defines a planar cross-section profile that moves along a given guide curve, and constructs the boundaries (envelope) of the swept volume. A generic operation performed by this function is shown in Figure 3, and can be expressed as follows:

$$\boldsymbol{s} := sweep(\boldsymbol{P_s}, \boldsymbol{P_e}, \boldsymbol{G}, \Omega, \boldsymbol{\sigma}) \tag{1}$$

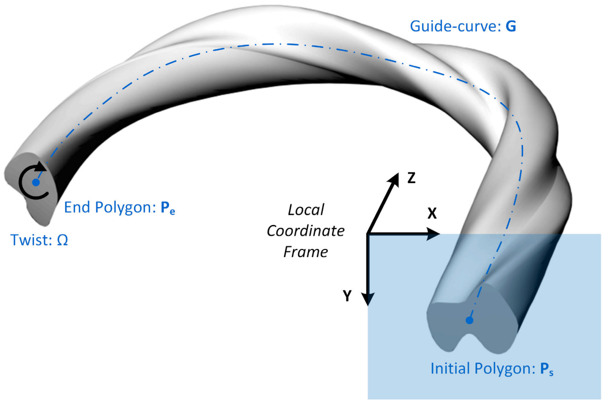


**FIGURE 3. The operation of sweep function and its critical parameters.**

where

$\boldsymbol{P_s} = \{\boldsymbol{p_s}[i] \in \mathbb{R}^{2\times1} : i \in \mathbb{N}_{\leq N_s}\}$ is the start profile,

$\boldsymbol{P_e} = \{\boldsymbol{p_e}[i] \in \mathbb{R}^{2\times1} : i \in \mathbb{N}_{\leq N_e}\}$ is the end profile,

$\boldsymbol{G} = \{\boldsymbol{g}[\boldsymbol{i}] \in \mathbb{R}^{3\times1} : i \in \mathbb{N}_{\leq N_g}\}$ is the $C^0$-continuous guide curve,

$\Omega \in \mathbb{R}$ is the twist rate along the swept path (0 is the default),

$\boldsymbol{\sigma} = \{\sigma[i] \in \mathbb{R}^+ : i \in \mathbb{N}_{\leq N_\sigma}\}$ is the shape scale factor, (default $\emptyset$),

$N_s, N_e, N_g, N_\sigma \in \mathbb{N}$ denote the sizes of the sets $\boldsymbol{P_s}$, $P_e$, etc.

With respect to the output argument of Equation 1, $\boldsymbol{s}$ defines the outer surface of the swept volume as a triangle mesh (i.e. STL): $\boldsymbol{s} = \{\boldsymbol{V}[i] \in \mathbb{R}^{3\times3} : i \in \mathbb{N}_{\leq N_v}\}$ is the set of Cartesian coordinates of the vertices of the triangular facets. The resulting Python function can handle the following cases:

- Shape interpolation between $\boldsymbol{P_s}$ and $\boldsymbol{P_e}$ along the sweep course;
- Rotation (or roll) and isometric scaling of the interpolated shape along the travel path;
- Automatic shape blending at sharp turns of the guide curve since there are no restrictions imposed on $\boldsymbol{G}$ than $C^1$ continuity;
- Self-intersecting surfaces and ill-defined guide-curves.

Figure 4 presents the rendered images of some models created by a single call of the sweep function. Notice that these test cases happen to be quite challenging for most CAD software packages. It is clear that, with suitable input arguments, all of the common geometric operations (such as extrude, revolve, loft, etc.) found in commercial

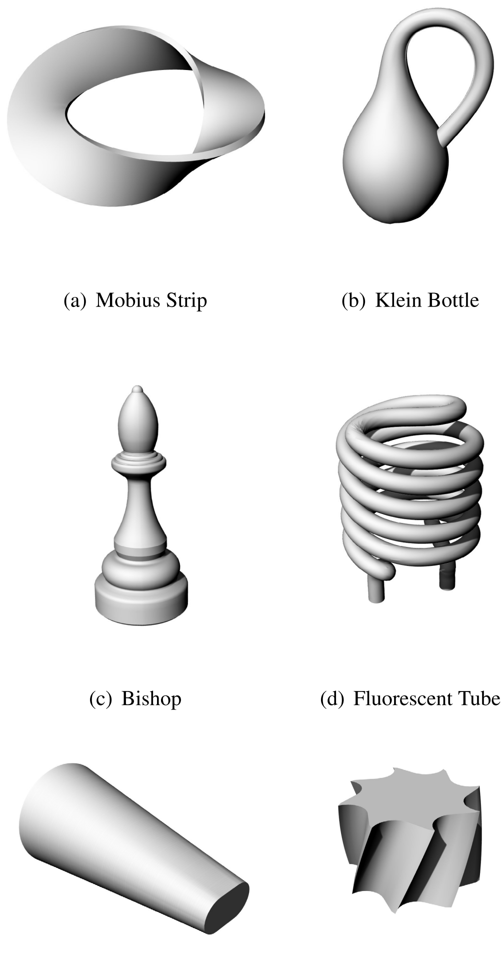


**FIGURE 4. Some solid geometric models generated by sweep function.**

CAD software can be implemented using our versatile function.

### 1) DEFINING SOLID GEOMETRIC MODELS

Using the familiar regularized CSG operations, complex solid geometric models can be constructed from the segments of the form $\mathbf{S} = \{\mathbf{s}[1], \mathbf{s}[2], \ldots, \mathbf{s}[N]\}$. Here, each segment can be generated by the sweep function described above:

$$\boldsymbol{s}[n] := \mathit{sweep}(\boldsymbol{P_s}[n], \boldsymbol{P_e}[n], \boldsymbol{G}[n], \Omega[n], \boldsymbol{\sigma}[n]) \quad (2)$$

$$\boldsymbol{s}[n] := \mathit{transform}(\boldsymbol{s}[n], \boldsymbol{A}[n], \boldsymbol{B}[n]) \quad (3)$$

Note that the sweep function generates a triangle mesh in a local coordinate frame. That is, the function computes the vertex coordinates of all the triangular facets in a local reference frame. Therefore, a homogeneous coordinate transformation on $\boldsymbol{s}[n]$ is needed so as to represent the mesh in a common global reference frame. Hence, the transform function has been created for this purpose. In Equation 3, the orientation of the local coordinate frame is defined with respect to the global frame. Here, $\boldsymbol{A}[n] \in \mathbb{R}^{3\times1}$ is a vector containing Euler angles. Similarly, $\boldsymbol{B}[n] \in \mathbb{R}^{3\times1}$ is the offset vector defining the location of the origin for the local coordinate system in the global reference frame.

Notice that the Boolean operations must accompany the transformed models (i.e. $\boldsymbol{s}[n]$). For this purpose, we define a set of operations on the geometric models:

$$\boldsymbol{\Psi} = \{\psi[n] \in \{1, -1\} : n \in \mathbb{N}_{\leq N}\} \quad (4)$$

where $\psi[n] = 1$ refers to the union operation, and $-1$ to the difference operation. Hence, the overall operation on the component pieces is simply expressed as

$$\boldsymbol{S} = \bigcup_{i\in\boldsymbol{I}^+} \boldsymbol{s}[i] - \bigcup_{i\in\boldsymbol{I}^-} \boldsymbol{s}[i] \quad (5)$$

where $\boldsymbol{I}^+ = \{i \in \mathbb{N}_{\leq N} : \psi[i] = 1\}$ and $\boldsymbol{I}^- = \{i \in \mathbb{N}_{\leq N} : \psi[i] = -1\}$. In the near future, we plan to augment the operation set $\boldsymbol{\Psi}$ to include lattice structures inside the model, surface attributes, and their corresponding properties / parameters.

Our Python script P3D will use the developed library to define complex solid geometric models. When this code is **executed** on the Python interpreter, two lists with reserved names (PL and OL) are created as the outcome. The list PL corresponds to the superset **S** and contains the triangle mesh data of the components. The list OL describes the accompanying Boolean operations. The firmware makes good use of the data contained in these two lists to produce the commands to fabricate the final object from its elements. Figure 5 illustrates the flowchart of the geometric model definition procedure.

Note that there is no need to assemble the component models into a single final mesh. Depending on the AM technology, the cross-sections of the final model S can be directly generated from the component models $\boldsymbol{s}[n]$.

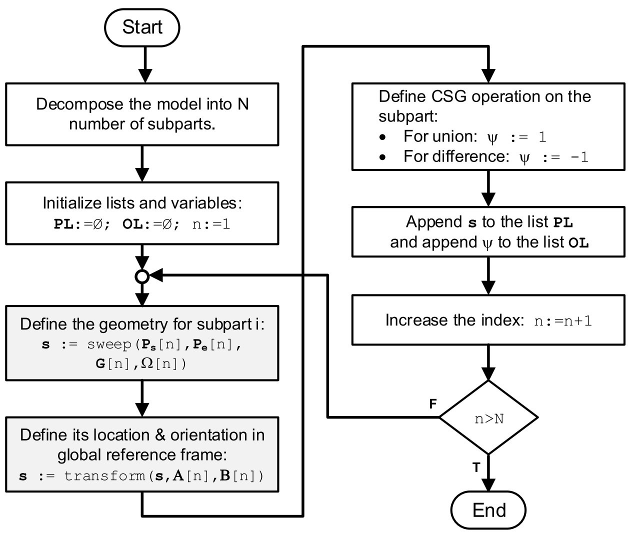

Start
Decompose the model into N number of subparts.
Initialize lists and variables: PL:=∅; OL:=∅; n:=1
Define the geometry for subpart i: s := sweep(Ps[n],Pe[n], G[n],Ω[n])
Define its location & orientation in global reference frame: s := transform(s,A[n],B[n])
Define CSG operation on the subpart:
For union: ψ := 1
For difference: ψ := -1
Append s to the list PL and append ψ to the list OL
Increase the index: n:=n+1
F
n>N
T
End

**FIGURE 5. Flowchart of parametric solid geometric model (P3D) definition.**

It is critical to recognize that a sizeable number of CAD modelling/slicing software uses scripts. That includes OpenSCAD, Open CASCADE, FreeCAD, OpenJsCAD, ImplicitCAD, RapCAD, BlockSCAD3D, and lately IceSL [9]. Some of them such as OpenPySCAD, PythonOCC have Python language support. The most common features in those languages are as follows:

- They are mostly designed as alternatives to the (GUI-based) conventional CAD/CAM software packages;
- They utilize extensively the OpenGL API;
- Their APIs comprise a vast number of methods including 3D graphics rendering engines/functions;
- They are mostly designed to run on workstations and PCs with considerable hardware resources. Furthermore, some of these tools like IceSL require even special hardware, such as high-end graphics cards/GPUs to render the designs.

The method proposed here circumvents the considerable complexity challenges arising from the issues explained above:

- The functions used in P3D script do not require a foreign API such as OpenGL; they are purely Python implementations.
- Since only two functions are sufficient to construct complicated geometric models, the learning curve is smooth.
- Users can create/edit/modify P3D scripts as needed. This is especially useful when the design has symmetries or contains replicated features. Moreover, it is easy to modify the topology, such as creating parametric designs.
- The proposed conceptualization of the geometric design is easy to implement on computers with modest capabilities and resources, such as Raspberry Pi, BeagleBone, Odroid, etc.

In the following section, we shall discuss how to generate cross-sections at arbitrary elevation from the individual component models.

### *B. OBTAINING CROSS-SECTIONS FROM SOLID GEOMETRIC MODELS*

In AM process planning, the most important stage is to obtain the cross-sections of the geometric model. The model is defined in terms of the two sets $\boldsymbol{S}$ and $\boldsymbol{\Psi}$, i.e., the lists PL and OL. The function **dissect** generates the cross-section of a triangle mesh at any given elevation. It is used as follows:

$$\boldsymbol{C} := dissect(\boldsymbol{s}, \boldsymbol{z}) \tag{6}$$

where $\boldsymbol{s}$ is a set associated with a (single) triangle mesh, and $\boldsymbol{z}$ is the elevation, in the direction of the z-axis. Formally, the plane z = h cuts the model, with h the second parameter in the function call. The constant h is in the global coordinate system. The output set (**C**) is a collection of polygonal chains in the dissection plane representing various cross-sections.

The function *modify* executes the Boolean operation $\psi$, add/union or subtract/difference, on the cross-section $\boldsymbol{C}$ and the polygon $\boldsymbol{P}$:

$$\boldsymbol{C} := \text{modify}(\boldsymbol{C}, \boldsymbol{P}, \psi) \tag{7}$$

The output $\boldsymbol{C}$ is the modified cross-section. The final cross-section is computed using a number of such modifications. The basic algorithm used in the function can be summarized as follows:

- Given a set $\boldsymbol{C}$ of polygons $\boldsymbol{C} = \{\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_m}\}$, find the subset of polygons, with index in set $\boldsymbol{I}$, that have a nonempty intersection with a given polygon $\boldsymbol{P}$:

$$\boldsymbol{I}(\boldsymbol{P}) = \left\{ i \in N_{\leq m} : \boldsymbol{c_i} \cap \boldsymbol{P} \neq \phi \right\} \tag{8}$$

- If $\psi = 1$ (i.e. $\cup$), then unite **P** with the intersecting polygons in **C**:

$$\boldsymbol{C}' = \left\{ \boldsymbol{c_k} \in \boldsymbol{C} : \forall k \in N_{\leq m} \wedge k \notin I \right\} + \boldsymbol{P} \cup \bigcup_{k \in \boldsymbol{I}} \boldsymbol{c_k} \tag{9}$$

- If $\psi$ = -1 (i.e. –), then take the difference of **P** with the intersecting polygons in **C**:

$$\boldsymbol{C}' = \left\{ \boldsymbol{c_k} \in \boldsymbol{C} : \forall k \in N_{\leq m} \wedge k \notin I \right\} + \left\{ \boldsymbol{c_k} - \boldsymbol{P} : \forall k \in I \right\} \tag{10}$$

- Let **C**:= **C**′ and return **C**.

Consequently, the overall cross-section of a solid model, implicitly expressed as Equation (5), can be constructed by the following function:

$$\boldsymbol{C} := xsect(\boldsymbol{S}, \boldsymbol{\psi}, z) \tag{11}$$

The pseudo-code is listed in Algorithm 1.

**Algorithm 1** Pseudocode of the Function *xsect*

```
1:  function C:= xsect(S,Ψ,z)
2:      for i:=1 to |S| do
3:          C:= dissect(S[i],z);
4:          if i = 1 then
5:              Q:= C;
6:          else
7:              for j:= 1 to |C| do
8:                  Q:= modify(Q,C[j],ψ[i]);
9:              end for
10:         end if
11:     end for
12:     C:= Q;
13: end function
```

### *C. PATH PLANNING*

The presented approach advocates the use of a path planning agent to fabricate automatically the part defined with our P3D Python script. The agent is to be part of the machine's firmware and is expected to eliminate the need for propriety- and non-propriety CAM software tools. The eliminated software tools free the user from both having to learn them and understanding the machine-specific data/codes (like NC/G Codes) that define the tool trajectory. The agent is expected to generate automatically the motion commands for the motor drivers along with the inputs to the auxiliary units (or peripheral devices). Hence, the input to the proposed AM pipeline is a single script defining the solid geometry.

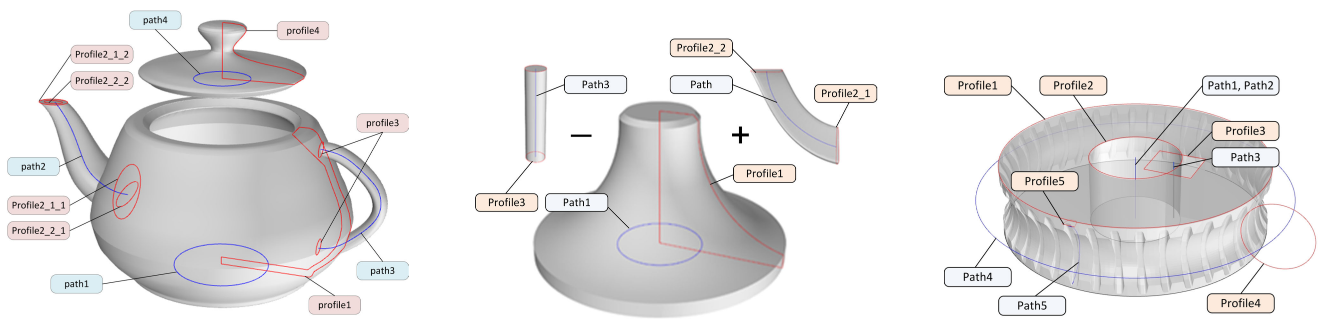
For this purpose, two path planning agents for the FFF and DLP technologies are devised using the Python scripting language. These open-source programs benefit from the library functions that are specifically developed for the AM process planning and modelling. Additionally, these programs are expected to serve as prototypes for firmware developers so that they can adapt them to any specific machine architecture, configuration, and hardware.

We shall sketch an abridged (offline) version of the algorithm implementing the path planning agent for the FFF technology:

1) Given the process parameters, such as $\Delta z$; nozzle diameter: $2\rho$), run the given P3D script to get $\boldsymbol{S}$ and $\boldsymbol{\Psi}$;
2) Initialize all variables/lists and set elevation $z := 0$;
3) Find the cross-section of the model at elevation $z$: $\boldsymbol{C} := xsect(\boldsymbol{S}, \boldsymbol{\psi}, z)$
4) Group polygons in $\boldsymbol{C}$ into a new superset $\boldsymbol{Q} = \{\boldsymbol{Q}[i] : i \in \mathbb{N}_{\leq|Q|}\}$ (i.e. create a tree structure) such that the elements of $\boldsymbol{Q}$ are defined as

$$\begin{aligned}\boldsymbol{Q}[i] = \{&\boldsymbol{q_j}[i] \in \boldsymbol{C} : \boldsymbol{q_1}[i] \cap \boldsymbol{q_k}[i] \\ &= \boldsymbol{q_k}[i], \quad j \in \mathbb{N}_{\leq|Q[i]|}, k \in \mathbb{N}^{>1}_{\leq|Q[i]|}\}\end{aligned} \tag{12}$$

where the first set in $\boldsymbol{Q}[i]$ (i.e. $\boldsymbol{q_1}[i]$) is the main polygon whereas the remaining ones denote the hole/interior polygons. Note that a polygon tree with a depth of



(a) Tea Pot (b) Impeller (c) Worm Gear

**FIGURE 6.** **Decomposed solid geometric models and their relevant geometric entities.**

two is described here for the sake of clarity. However, the actual algorithm forms a tree with indefinite depth.

5) For every element in $\boldsymbol{Q}$,
   a) For intermediate layers, create the outer layers (i.e. shells) of $\boldsymbol{Q}[i]$ by employing a curve offset generation algorithm presented in [16] (i.e. **pcur** function). Using **infill** function of the library, compute the rectilinear paths with specified spacing (i.e. infill ratio) covering the remainder of the internal areas of $\boldsymbol{Q}[i]$;
   b) For top layers, compute all curve offsets associated with $\boldsymbol{Q}[i]$ where the distance between the generated offset curves ($\rho$) is considered to be a property of the equipment at hand (i.e. the nozzle diameter of the extruder head);
   c) Add the resulting data to the list $\boldsymbol{\Gamma}$ which serves the main repository to describe the tool trajectory in 3D Cartesian space. For the sake of convenience, the current elevation (z-coordinate) is appended to the coordinates of all points to be added to the list.
6) Increase the elevation: $z := z + \Delta z$;
7) If $z < z_{max}$, go to Step 3.

With the given feed-rate, the coordinates in $\boldsymbol{\Gamma}$ are interpolated to issue real-time position commands to the axis drivers (as well as PDs like extruder-head control system and filament feeder). Note that large gaps ($> \rho$) between the points in $\boldsymbol{\Gamma}$ signal the tool retraction and rapid travel between the accompanying points. Notice that the parameters $\Delta z$ and $\rho$ (and the data size for $\boldsymbol{Q}$ and $\boldsymbol{\Psi}$) directly govern the fabrication quality.

It is critical to note that in FFF, fabricating a single layer could typically take one to three minutes, depending on the size, complexity, and infill ratio of a given model. Hence, it is beneficial to send out promptly the interpolation data (associated with a particular layer) to the 3D printer processor. While the fabrication is in progress, the agent (serving as the integral part of the device) will have ample time to process the next layer despite its limited computational resources.

The agent for the DLP technology is much simpler if compared to the afore-mentioned counterpart. The entire cross-section at a particular elevation is to be transferred to the projector as a binary image. The material at the ‘‘fill’’ pixels is deposited at once. Hence, Step 5 of Algorithm 1 becomes unnecessary. The polygons representing the cross-sections of the object at a particular elevation (at Step 4) are basically converted to a binary bitmap image and is sent out to the digital light processor/projector of the machine through the designated video data transfer protocol.

## IV. EXPERIMENTS AND DISCUSSIONS

We are using three test pieces to assess the performance of the presented method:

1) (Utah) Tea Pot
2) Impeller (of a (Centrifugal Compressor)
3) Worm Gear

In the following subsection, we shall describe the modeling of these artefacts. Furthermore, we shall present an example for a user-friendly interface (i.e. dialog box) that enables users to make the design changes easily and efficiently without editing P3D scripts. Finally, the fabrication performance of our pipeline are compared to those of its counterparts in the last subsection.

### A. DEFINITION OF THE TEST MODELS

The test parts are decomposed into a number of subcomponents as illustrated in Figure 6. Employing the geometric entities in Figure 6, three P3D Python scripts are created to define the corresponding geometric models. Table 1 lists the Python script for the impeller. Note that the data associated with some of the geometric entities, such as paths and profiles, are directly loaded from a MAT data file for the sake of keeping the Python listing short. In the actual application, all lists can be included to the P3D file. The related files and some of the examples are available on GitHub [17].

By modifying the given script, a family of impellers with different geometric attributes can be developed. Figure 7 illustrates some of the modified cases. For instance, Figure 7(a) demonstrates the rendered image of the original model. By reducing the number of blades from 12 to 6 (See line 10 of Table 1), the geometric model in Figure 7(b) is obtained. Figure 7(c) shows another variant where the blades

**TABLE 1. P3D Python Script defining the impeller.**

```python
from LIPRO import *
PL = []; OL = []
profile1, profile2_1, profile2_2, path2 = Load('Compressor.mat')
### 1st part (main body)
path1 = polygon(10,100)
PL.append(sweep(profile1,profile1,path1)); OL.append(1)
### 2nd part (blades)
part2 = sweep(profile2_1,profile2_2,path,-20)
part2 = transform(part2,0,270,0,37.5,0,1)
for i in range(12):
    PL.append(transform(part2,0,0,i*30,0,0,0)); OL.append(1)
### 3rd part (central hole)
profile3 = polygon(4.5,100); path3 = [[0,0,0],[0,0,40]]
PL.append(sweep(profile3,profile3,path3)); OL.append(-1)
```

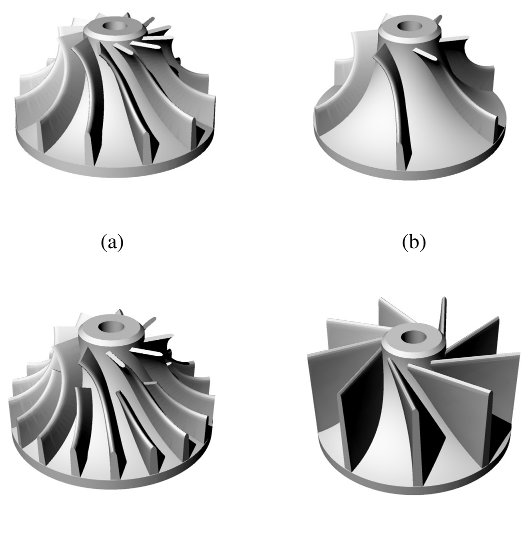
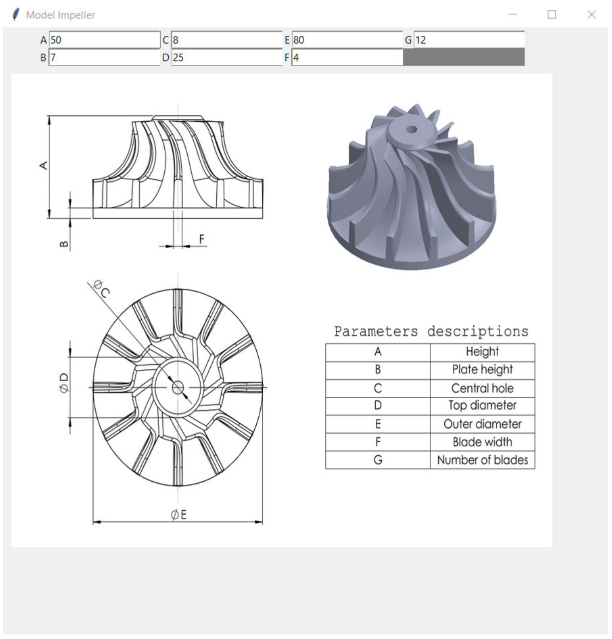


**FIGURE 7. Rendered images of the impeller with different parameters.**



A 50
B 7
C 8
D 25
E 80
F 4
G 12

Parameters descriptions

| | |
|---|---|
| A | Height |
| B | Plate height |
| C | Central hole |
| D | Top diameter |
| E | Outer diameter |
| F | Blade width |
| G | Number of blades |

**FIGURE 8. A sample mask (dialog box).**

in alternating order is reduced in length by changing the corresponding guide curve. Finally, modifying the guide curve associated with all the blades, the model shown in Figure 7(d) is attained.

For better interaction with the users, a dialog box (a.k.a. "mask") can be conveniently incorporated to the model defined in P3D with the utilization of PyGUI library. A simple dialog box whose parameters describe the main dimensions (A to G) of the Impeller model is demonstrated in Figure 8. Hence, the users can safely modify the geometric model without editing P3D script.

Next is the fabrication process with the P3D definitions as inputs.

## B. FABRICATION OF THE PARTS

The FFF agent discussed in Section 3 is used to fabricate the test parts on an Ultimaker 2 Go 3D printer. Since the firmware of this device is not open-source, a Type-I implementation [18] is made on a Raspberry Pi Model 3 single board computer. The agent has been developed in Python, and is now modified to generate the raw trajectory data for each layer offline. That is, the agent populates the $\boldsymbol{\Gamma}$ list layer by layer. The resulting data are then converted to NC commands needed to fabricate a particular layer on the 3D printer. Note that the NC codes produced must be sent out to the main controller of the printer (Arduino Mega 2560) in batches via USB. The NC code interpreter, by design, has a limited set of commands and a small buffer. Table 2 summarizes the fabrication parameters while Figure 9 shows the fabricated parts. For the tea pot case in Figure 9(a), two different P3D scripts are used, one for the teapot body, the other for the lid.

As can be seen from Figure 9, the printing quality of the fabricated test parts is comparable to that of the conventional

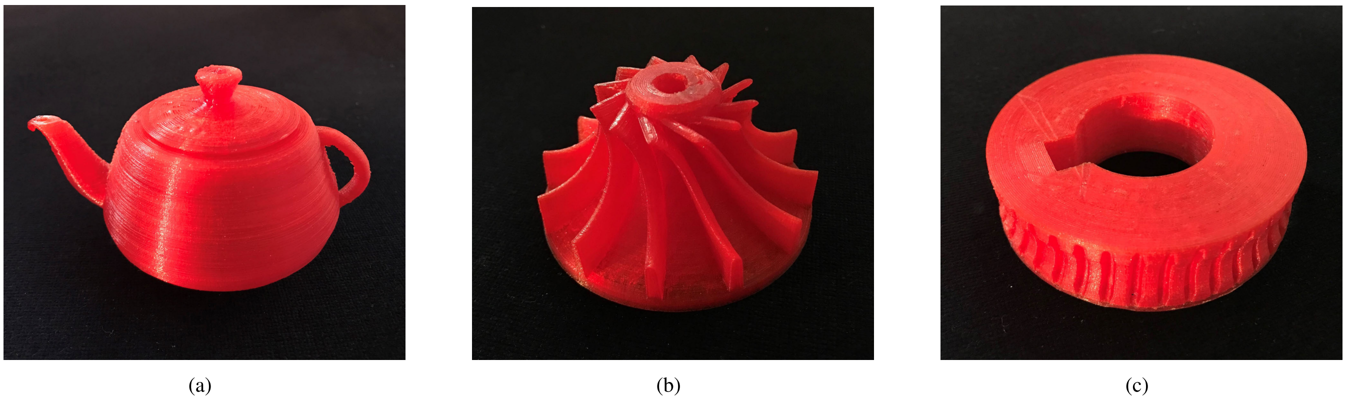


FIGURE 9. Fabricated test parts.

TABLE 2. Fabrication parameters of the test cases.

| | Tea Pot | Impeller | Gear |
|---|---|---|---|
| **Workpice Size [mm]** | 105×76×60 | 76×76×40 | 87×87×23.5 |
| **Material** | PLA | | |
| **Filament Diameter [mm]** | 2.85 | | |
| **Nozzle Diameter [mm]** | 0.4 | | |
| **Layer Thickness [mm]** | 0.1 | | |
| **Feed-rate [mm/min]** | 1800 | | |
| **Retraction Speed [mm/s]** | 6000 | | |
| **Number of Shells** | 5 | | |
| **Infill Density [%]** | 30 | | |
| **Printing Time [min]** | 304 | 431 | 279 |

pipeline. The CURA open-source CAM software was used to generate the overall NC Code. Note that even a simple, small change of process parameters, such as modifications in layer depth, nozzle diameter, feed-rate etc.) will require the regeneration of the entire NC Code, from scratch. For instance, in the CURA software tool, this simple modification alone would require a chain of changes, including

1) loading the STL model,
2) regeneration of the NC code,
3) saving the resulting file onto a SD card,
4) transferring the code to the printer, and
5) initiating the fabrication.

These steps could take 2 to 3 minutes depending on the architecture of the printer and on the complexity of the STL model. The situation would get worse if the user were to make small changes on the geometry of the part. This alteration would then necessitate re-using a CAD software tool. The revisions on the part and the creation of the STL file could also take a considerable amount of time. None of the traditional methodologies described would challenge the proposed approach in view of the issues mentioned above. When using our methodology, these modifications are promptly carried out and there is no need to re-engage any CAD/CAM software tools.

## C. COMPARISONS

To evaluate the performance of the proposed method quantitatively, the Impeller model (Case 2) was designed using 3 different AM pipelines: **i)** Proposed method (a.k.a. "LIPRO"); **ii)** Conventional (i.e. SolidWorks CAD software + CURA software); **iii)** IceSL. Implementation details are summarized in Table 3. Here, the term labelled as the "Param. CAD" refers to a parametric SolidWorks model whose geometric parameters (a total of 7) are read externally from an Excel spreadsheet. Similarly, the first row of this table presents the size of the script/file defining the 3D solid geometric model while the second row ("Design Time") denotes the elapsed time to design the part from the scratch. The third row ("Learning Time") indicates the "rough" estimate on the period for a novice to learn the required skills to design the part using the corresponding software tools. Likewise, the fourth row denotes the specifications of a typical hardware to run the accompanying software packages. Finally, the last row ("The Size of the NC Code") denotes the size of the NC Code generated by the above-mentioned pipelines. As can be seen from the table, the proposed approach appears to be on the leading edge at almost every category:

- The presented method yields the smallest model size since only a few functions in P3D script are employed to generate a relatively complex 3D model.
- The learning curve for LIPRO appears to be quite identical to that of the IceSL and is not steep due to the fact that only the basic knowledge in Python programming language is sufficient. Hence, it is expected to take less time to commence fabrication in LIPRO if compared to the conventional techniques.
- LIPRO can run on almost any small-form-factor computer whereas working with modern CAD software tools in conventional AM pipelines oftentimes necessitates powerful PCs. Similarly, IceSL demands a high-end GPU due to its utilization of advanced rendering techniques.
- With respect to the size of the NC code, unlike other techniques, LIPRO does not generate a NC Code to be downloaded to the 3D printer. In this Type-I implementation, LIPRO directly generates the tool trajectory for each layer and the corresponding NC code does not need to be stored externally.

**TABLE 3.** Comparison of the proposed pipeline with similar approaches.

| | LIPRO (P3D) | Conventional | Conventional (Param. CAD) | IceSL |
|---|---|---|---|---|
| **Size of the script/file (KB)** | 1.2 | 1046 | 2213 | 1.29 |
| **Designing time (minutes)** | 30 | 40 | 60 | 30 |
| **Learning time** | Mastery on Python basics | Minimum 10 days | | Mastery on Lua basics |
| **Hardware** | Raspberry Pi Hardware:<br>Processor: Quad Core 1.2GHz<br>Broadcom BCM2837 64bit CPU<br>Memory: 1GB RAM | Processor: 3.3 GHz or higher<br>Hard Drive: Solid State Drive (SSD), maintaining at least 20GB free space<br>Memory: 16GB minimum<br>Graphics Card:NVIDIA Quadro P1000 or AMD Radeon Pro WX 4100 | | Graphics Card: GeForce GTX 480 / 580 / 680 / 970 / Titan / 1080<br>GeForce GT 555M / 610M / GTX 1660 Ti Mobile<br>Intel HD Graphics 630, 4600, 4400, 4000*<br>AMD Radeon 290X |
| **Size of the NC code (KB)** | N/A | 5506 | | 11807 |

## V. CONCLUSION AND FUTURE WORK

We have introduced a new design- and fabrication paradigm for 3D printers. Our intention is to overcome the drawbacks of the conventional 3D printing pipeline. The key contributions of the proposed process are as follows:

- A Python script-based implicit geometric modeling technique is presented. The method employs a single, versatile function called **sweep** to generate solid geometric models piecewise using the Boolean operators of CSG. Users can easily define complex geometric parts. Since all of the library functions (including sweep) mentioned in the paper are open-source, users may tailor them to accommodate their own needs.
- The presented (path planning) agent, which we developed in the Python language using versatile LIPRO library functions, can easily be implemented on any computing platform with limited resources. In our experiments, we have been using a single-board computer Raspberry Pi 3. All computers with small form-factors would also be suitable since they can be conveniently integrated to the AM printing equipment. Similarly, our (open-source) path-planning agents can be adapted to any AM technology.
- The method eliminates the need for third-party CAD software. The user can create/edit/modify the solid models using a simple text editor. As a verification tool, our system includes a simple 2D geometric viewer as a utility.
- Our approach advocates the elimination of CAM software as well as intermediate low-level codes like the NC Code. It offers data portability since the only input to the proposed AM pipeline is P3D Python script that defines the geometry along with the properties of the solid part.
- We build our geometric models as the decomposition of subparts. The advantage is that the decomposition of the geometric model simplifies the path planning procedure, as well as later stages of the AM pipeline.
- For the time-being, the solid geometric models inside P3D are implicitly represented as triangle meshes (i.e. STL models) for the sake of easy implementation. From the standpoint of users, this indirect use of STL models does not alter the way the geometric entities are defined. This is partly due to the fact that the STL data are employed in the subsequent slicing operations while processing P3D. Such intermediate operations and their outcome are not transparent to users. Evidently, advanced data structures to represent geometric entities more accurately (such as the ones offered by the AMF, 3MF, OBJ etc.) could be smoothly incorporated to P3D in the near future as the need arises.

Our research continues with the development of the followings:

- The development of a web-based CAD tool is pending to allow users design complex parts and save them as P3D files.
- The P3D file content will be augmented to include internal- and external properties of the solid models defined (including density, shell size, infill geometry, lattice structures, texture, surface quality, etc.).
- To edit the geometric parameters in P3D easily and efficiently, we plan to offer masks as an option for designers.
- The prototypes of path planning agents for all remaining AM technologies (e.g. FFF, SLA, PBF, DED, Material Jetting, Binder Jetting, and Sheet Lamination etc.) will be presented. Developers will be able to modify these open-source programs and adapt them to any AM machinery.

## REFERENCES

[1] A. Paolini, S. Kollmannsberger, and E. Rank, "Additive manufacturing in construction: A review on processes, applications, and digital planning methods," *Additive Manuf.*, vol. 30, Dec. 2019, Art. no. 100894.

[2] C. D. Toth, J. O'Rourke, and J. E. Goodman, Eds., *Handbook of Discrete and Computational Geometry*. Boca Raton, FL, USA: CRC Press, 2017.

[3] V. Chandru, S. Manohar, and C. E. Prakash, "Voxel-based modeling for layered manufacturing," *IEEE Comput. Graph. Appl.*, vol. 15, no. 6, pp. 42–47, Nov. 1995.

[4] M. C. Messner, "A fast, efficient direct slicing method for slender member structures," *Additive Manuf.*, vol. 18, pp. 213–220, Dec. 2017.

[5] R. Minetto, N. Volpato, J. Stolfi, R. M. M. H. Gregori, and M. V. G. da Silva, "An optimal algorithm for 3D triangle mesh slicing," *Comput.-Aided Des.*, vol. 92, pp. 1–10, Nov. 2017.

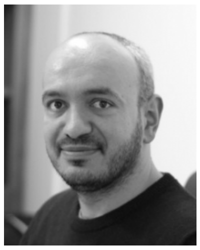
[6] B. Starly, A. Lau, W. Sun, W. Lau, and T. Bradbury, "Direct slicing of STEP based NURBS models for layered manufacturing," *Comput.-Aided Des.*, vol. 37, no. 4, pp. 387–397, 2005.

[7] M. Alexa, K. Hildebrand, and S. Lefebvre, "Optimal discrete slicing," *ACM Trans. Graph.*, vol. 36, no. 1, pp. 1–16, Feb. 2017.

[8] J. Xu, W. Hou, and H. Zhang, "An improved virtual edge approach to slicing of point cloud for additive manufacturing," *Comput.-Aided Des. Appl.*, vol. 15, no. 3, pp. 330–336, May 2018.
[9] S. Lefebvre and L. I. N. Grand-Est, "IceSL: A GPU accelerated CSG modeler and slicer," in *Proc. 18th Eur. Forum Additive Manuf. (AEFA)*, Jun. 2013, pp. 1–9.
[10] B. Zhao, Z. Lin, J. Fu, and Y. Sun, "Generation of truss-structure objects with implicit representation for 3D-printing," *Int. J. Comput. Integr. Manuf.*, vol. 30, no. 8, pp. 871–879, 2017.
[11] L. Zeng, L. M.-L. Lai, D. Qi, Y.-H. Lai, and M. M.-F. Yuen, "Efficient slicing procedure based on adaptive layer depth normal image," *Comput.-Aided Des.*, vol. 43, no. 12, pp. 1577–1586, 2011.
[12] J. C. Steuben, A. P. Iliopoulos, and J. G. Michopoulos, "Implicit slicing for functionally tailored additive manufacturing," *Comput.-Aided Des.*, vol. 77, pp. 107–119, Aug. 2016.
[13] L. Xia, S. Lin, and G. Ma, "Stress-based tool-path planning methodology for fused filament fabrication," *Additive Manuf.*, vol. 32, Mar. 2020, Art. no. 101020.
[14] Q. Li, Q. Hong, Q. Qi, X. Ma, X. Han, and J. Tian, "Towards additive manufacturing oriented geometric modeling using implicit functions," *Vis. Comput. Ind., Biomed., Art*, vol. 1, no. 1, pp. 1–16, Dec. 2018.
[15] W. Li and S. Mcmains, "A sweep and translate algorithm for computing voxelized 3D Minkowski sums on the GPU," *Comput.-Aided Des.*, vol. 46, pp. 90–100, Jan. 2014.
[16] U. Yaman and M. Dolen, "A novel command generation paradigm for production machine systems," *Robot. Comput.-Integr. Manuf.*, vol. 51, pp. 25–36, Jun. 2018.
[17] *LIPRO*. Accessed: Jun. 7, 2021. [Online]. Available:https://github.com/a106lipro/LIPRO
[18] J.-M. Cha, S.-H. Suh, J.-Y. Hascoet, and I. Stroud, "A roadmap for implementing new manufacturing technology based on STEP-NC," *J. Intell. Manuf.*, vol. 27, no. 5, pp. 959–973, Oct. 2016.

**VAHID HASELTALAB** received the B.Sc. degree in manufacturing engineering technology from Technical University, Iran, in 2011, and the M.Sc. degree in mechanical engineering from Middle East Technical University (METU), in 2018, where he is currently pursuing the Ph.D. degree with the Department of Mechanical Engineering. His research interests include CAD/CAM, additive manufacturing, and computational geometry.

**MELIK DOLEN** received the B.S. degree in mechanical engineering from Istanbul Technical University, in 1990, the M.S. degree from the University of New Hampshire, Durham, in 1994, and the Ph.D. degree from the University of Wisconsin–Madison, in 2000. He is currently a Professor with the Department of Mechanical Engineering, Middle East Technical University. His research interests include the design of advanced manufacturing systems, automation, precision engineering, and engineering optimization. He is the author or coauthor of numerous technical articles in this field.



**ULAS YAMAN** (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in mechanical engineering (mechatronics) from Middle East Technical University (METU), Turkey, in 2007, 2010, and 2014, respectively. He has been an Associate Professor of mechanical engineering with METU, since October 2015. He was a Visiting Assistant Professor of computer science with Purdue University, from July 2014 to October 2015. His research interests include 3D printing and additive manufacturing, CAD/CAM architectures, command generation, FPGA-based embedded systems, and computational geometry for design and manufacturing.

**CHRISTOPH HOFFMANN** is currently a Professor with the Department of Computer Science, Purdue University. Before joining the Purdue Faculty, he taught at the University of Waterloo, Canada. He has also been a Visiting Professor at the Christian-Albrechts-University of Kiel, West Germany, in 1980, and Cornell University, from 1984 to 1986. He is the author of *Group-Theoretic Algorithms and Graph Isomorphism* (Springer-Verlag) and *Geometric and Solid Modeling: An Introduction* (Morgan Kaufmann, Inc.). He has received national media attention for his work simulating the 9/11 attacks on the Pentagon and the World Trade Center. His research interests include geometric and solid modeling, its applications to manufacturing and science, simulation of physical systems, in particular, research on geometric constraint solving, and the semantics of generative feature-based design.