G-code generation in a NURBS workflow for precise additive manufacturing

Jesús Miguel Chacón, Javier Sánchez-Reyes and Javier Vallejo
Department of Applied Mechanics and Projects Engineering, IMACI, Universidad de Castilla-La Mancha, Escuela Tecnica Superior de Ingeniería Industrial, Ciudad Real, Spain, and
Pedro José Núñez
Department of Applied Mechanics and Projects Engineering, INEI, Universidad de Castilla-La Mancha, Escuela Tecnica Superior de Ingeniería Industrial, Ciudad Real, Spain

Abstract

Purpose – Non-uniform rational B-splines (NURBSs) are the de facto standard for representing objects in computer-aided design (CAD). The purpose of this paper is to discuss how to stick to this standard in all phases of the additive manufacturing (AM) workflow, from the CAD object to the final G-code, bypassing unnecessary polygonal approximations.

Design/methodology/approach – The authors use a commercial CAD system (Rhino3D along with its programming environment Grasshopper) for direct slicing of the model, offset generation and trimming. Circular arcs are represented as quadratic NURBSs and free-form geometry as quadratic or cubic polynomial B-splines. Therefore, circular arcs are directly expressible as G2/G3 G-code commands, whereas free-form paths are rewritten as a succession of cubic Bézier curves, thereby admitting exact translation into G5 commands, available in firmware for AM controllers, such as Marlin.

Findings – Experimental results of this paper confirm a considerable improvement in quality over the standard AM workflow, consisting of an initial polygonization of the object (e.g. via standard tessellation language), slicing this polygonal approximation, offsetting the polygonal sections and, finally, generating G-code made up of polyline trajectories (G1 commands).

Originality/value – A streamlined AM workflow is obtained, with a seamless transfer from the initial CAD description to the final G-code. By adhering to the NURBS standard at all steps, the authors avoid multiple representations and associated errors resulting from approximations.

Keywords Additive manufacturing, NURBS, Cubic Bézier curve, G-code, G2/3 command, G5 command

Paper type Research paper

1. Introduction

1.1 The non-uniform rational B-splines standard for computer-aided design (CAD) representation

The input data in any additive manufacturing (AM) process consists of a CAD model, typically a boundary representation (Shapiro, 2002) whose faces are trimmed non-uniform rational B-spline (NURBS) surfaces (Farin, 2002). A NURBS-based representation enjoys the following advantages, relevant for AM:

- *De facto standard*: NURBSs are implemented in data exchange formats [initial graphics exchange specification (IGES), STEP], allowing an exact transfer between dissimilar CAD–computer aided manufacturing (CAM) systems or a unified database in geometric libraries (SMS, 2022).
- *G-code support*: Cubic integral (i.e. polynomial) curves, the most popular subset of NURBS for free-from geometry, admit an exact conversion into cubic Bézier curves. In turn, these cubics are expressible as G5 commands, available in AM firmware. Circles and circular arcs also admit an exact representation as NURBS and, hence, a direct implementation as G2/3 code.

The current issue and full text archive of this journal is available on Emerald Insight at: **https://www.emerald.com/insight/1355-2546.htm**

Rapid Prototyping Journal
28/11 (2022) 65–76
Emerald Publishing Limited [ISSN 1355-2546]
[DOI 10.1108/RPJ-09-2021-0254]

The authors thank the Precision Center of Hexagon Manufacturing Intelligence in Vitoria-Gasteiz (Álava, Spain) for providing the necessary equipment for measuring the NURBS and STL models, and the Applications Engineer Mr Manuel Castaño Hipólito for his professional assistance, and Dr R. Dorado for several helpful discussions on how to enable the G2/3 and G5 commands. Authors are also grateful to the referees for their valuable suggestions, which improve this article.

This research was supported by grant PID2019-104586RB-I00, funded by MCIN/AEI/10.13039/501100011033; and grants SBPLY/19/180501/000247, SBPLY/19/180501/000170, Consejería de Educación, Cultura y Deportes (Junta de Comunidades de Castilla-La Mancha), co-financed by the ERDF (European Regional Development Fund).

Received 24 September 2021
Revised 21 December 2021
8 June 2022
9 August 2022
Accepted 10 August 2022

- *Precision*: NURBS, based on weights and control points, can express not only free-from geometry but also analytic shapes in exact fashion.
- *Graphics Processing Unit (GPU) support*: Graphic libraries (OpenGL, DirectX) directly support NURBS, which can, hence, be manipulated and visualized in real-time thanks to the GPU. Furthermore, there exist GPU implementations of some geometric computations, such as area integrals or Hausdorff distances.

Software developers have invested heavily in NURBSs for all these favorable properties, so downstream applications should be adapted to this representation, as Allen (2013) notes. Such applications include AM-related software, but unfortunately, this is not the case. Consequently, the potential of NURBSs has been entirely untapped in AM.

1.2 Conventional additive manufacturing workflow: problems

Recently, AM hardware capable of generating three-dimensional (3D) trajectories has been introduced, which adds flexibility to the specification of AM paths (Ezair *et al.*, 2018). However, as most AM is still restricted to planar paths, the AM workflow proceeds in two steps (Figure 1):

1 Generating a polygonal approximation to the exact NURBS model within a CAD system, typically a standard tessellation language (STL) file, setting a maximum deviation ε_{STL}.
2 Transferring the polygonal model to a 3D printing software that performs three operations:
   - Slicing the model along horizontal planes Π, which furnishes its contours (sections);
   - Path planning for outer and inner walls, which involves offsetting and infill patterns; and
   - G-code generation, usually based on linear interpolations (G1 instruction).

The first problem stems from the discretization in Step 1, which may generate invalid tessellations, especially for complex models. The STL file may need post-processing to fix errors (Kumar and Dutta, 1997), such as gaps, overlaps or degenerate faces, resulting in slicing failures. The required file checks and manual repairs (Starly *et al.*, 2005) significantly slow the workflow.

Replacing the exact CAD model with an error-free polygonal approximation allows a straightforward computation of the contours (Step 2a) as polylines. However, even trying to reduce the effect of the discretization, the resulting deviation considerably exceeds the geometric tolerance of the machine for complex geometries (Bertacchini *et al.*, 2021). Moreover, a tighter ε_{STL} could make matters worse by increasing the chances of errors. The discretization leads to downstream problems, as surveys on AM (Gao *et al.*, 2015; Qin *et al.*, 2019) highlight:

- *Slicing*: As Figure 2 illustrates, the vertices $\mathbf{V}_i$ of the polygonal section S_{STL} (in red) do not lie, in general, on the actual CAD section S_{CAD}, for the vertical curvature of the surface, which introduces an error. Even if these vertices $\mathbf{V}_i$ lie on S_{CAD}, a chordal error results from the horizontal curvature of S_{CAD}. Thus, the total error ε between the section S_{CAD} and that S_{STL} from the polygonal file could grow larger than the error ε_{STL} in Step 1, as Jamieson and Hacker (1995) remark.
- *Path planning*: It feeds on inaccurate polygonal geometry, which leads to downstream errors when offsetting polyline contours and generating the corresponding strands, whose width is determined by the nozzle diameter $Ø_n$. As depicted in Figure 3, voids between strands may appear, compromising the mechanical integrity of the object.
- *G-code generation*: As the final trajectory amounts to polylines, they only display positional continuity at the joints, thereby lacking the higher smoothness of circular arcs and cubic Bézier curves available in existing AM firmware.

1.3 Our proposal: adhering to non-uniform rational B-splines at all steps

The conventional AM workflow incurs an error larger than that derived from hardware limitations, hindering the manufacturing of truly functional parts. Our proposal addresses this issue by sticking

Figure 1 Conventional generation of G-code for additive manufacturing, based on an initial polygonal approximation

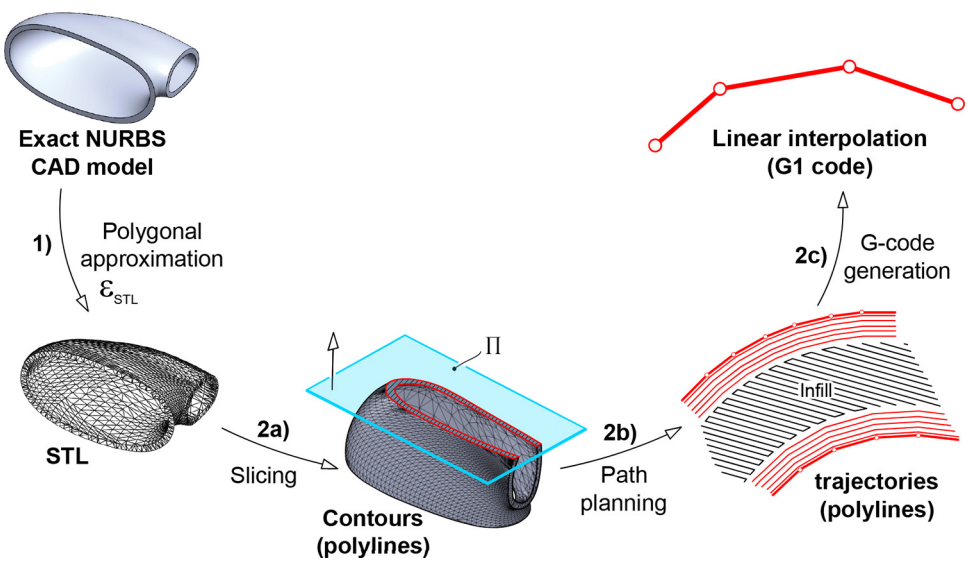


Figure 2 Errors incurred by slicing a polygonal approximation through a plane Π

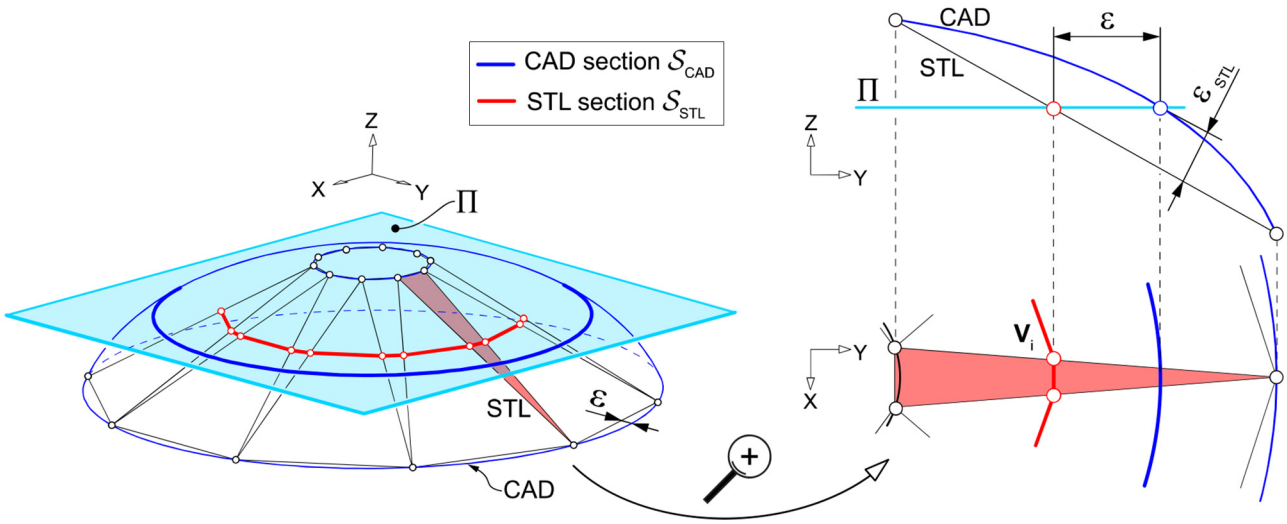
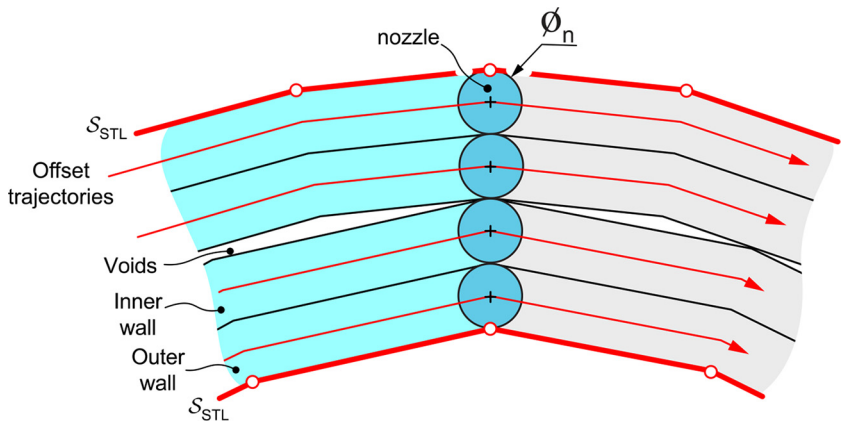


Figure 3 Path planning: Offsetting polygonal inner and outer walls



to a NURBS representation at all steps. Clearly, we have to bypass the polygonal approximation (Step 1) and then proceed as follows:

- direct slicing of the model in the NURBS environment provided by a NURBS-based CAD system;
- path planning, including offset generation, in this NURBS environment; and
- G-code generation in NURBS format; circles and circular arcs (G2/3 code) and Bézier cubics (G5), in addition to polylines (G1).

Thus, the paper is arranged as follows. First, in Section 2, we compare our approach with alternative proposals. Section 3, focusing on slicing and path planning, outlines our implementation in the NURBS-based commercial software (Rhino3D and Grasshopper). Then, in Section 4, we discuss how to exploit the untapped NURBS capabilities of G-code generators. In Section 5, we describe implementation details in existing AM firmware to encourage the reproducibility of our work. We also show a real example of a final part manufactured with a fused filament fabrication (FFF) 3D printer, which corroborates the improvement in geometric quality. Finally, in Section 6, conclusions are drawn.

2. Alternatives to conventional polygonization

2.1 Alternative formats

A first option to overcome the inaccuracies of the STL format is replacing it with a more advanced one. Qin *et al.* (2019) compare alternative AM formats, concluding that none of them fulfills all the requirements needed. This conclusion applies to the most supported commercial formats:

- *3MF*: Although it compares favorably against STL for incorporating data such as color or material, unfortunately, it is still based on triangular meshes. Therefore, it does not tackle the critical deficiency of STL.
- *OBJ*: It allows meshes as well as NURBS, but AM controllers do not support the latter for its complexity, so once again, it does not provide a practical solution.
- *Additive manufacturing file format (AMF)*: It not only uses triangular meshes, like STL, but also admits curved triangular faces. However, Yu *et al.* (2017) warn that the tangent definition for these faces results in a slicing error (Step 2a) similar to that incurred by STL and then put forward improvements to tackle this deficiency.

Following the idea of the open standard AMF, a low-degree patch would be a reasonable compromise between a polygonal model and complex NURBSs (Allen, 2007). For instance, Paul and Anand (2015) propose Steiner patches, a subset of quadratic rational Bézier triangles that admits accurate slicing, as any plane intersects a Steiner surface in a NURBS curve. However, Allen (2007) also notes that this strategy would entail rewriting existing software and hamper data exchange capabilities. Other formats exist, but none has gained widespread acceptance, so they do not stand as a viable option.

2.2 Direct slicing

As concluded in the previous subsection, no format provides a practical alternative to current tessellated approximations. The only option is bypassing the initial conversion (Step 1) to an intermediate format and direct slicing the original CAD model, an idea that goes back almost three decades (Carleberg, 1994).

Slicing a NURBS surface does not yield, in general, a NURBS curve, aside from simple cases, such as transversal sections of swung surfaces (Piegl and Tiller, 1997), which encompass revolution and extrusion surfaces or arbitrary sections of the above Steiner patches. These patches encompass quadrics, which also furnish simple conic sections, admitting an exact quadratic NURBS representation and hence simplifying path planning (Farouki and König, 1996). Although direct slicing is not without its own challenges (Oropallo and Piegl, 2016), it boils down to intersecting the CAD model with a plane, a fundamental operation implemented in existing geometric kernels, which furnish excellent NURBS approximations within the tolerance of the CAD model.

As Pandey *et al.* (2003) conclude, there is a need for modification in hardware that can drive the output of direct slicing instead of splitting it into polylines. Unfortunately, most previous proposals that advocate direct slicing do not exploit its true potential and incur inaccuracies, as they abandon the NURBS representation at some point in the AM workflow. For instance:

- Contours are not represented directly as NURBS curves but as discrete points (Ma *et al.*, 2004; Feng *et al.*, 2018), slice raster lines (Starly *et al.*, 2005) or bitmap images (Xu and Jing, 2011; Xu *et al.*, 2011).
- Contours are represented as NURBS, but their offsets for path planning are not generated using techniques tailored to NURBS (Patrikalakis and Maekawa, 2002; Elber *et al.*, 1997; Sánchez-Reyes and Chacón, 2015) but via less precise methods.

2.3 Our proposal

A sensible, more practical solution is adapting an existing commercial tool to perform direct slicing and path planning. A first possibility could be using commercial CAM software, adapting it from subtractive to AM and exploiting its functionalities. Actually, CAM software can be used to compute horizontal contours (Step 2a), whose definition does not depend on the manufacturing method. However, it aims at manufacturing by removing material, whereas, in AM, material is added, so path planning (Step 2b) from a given contour S_{CAD} differs significantly. More specifically, CAM path planning involves outer offsets to S_{CAD}, whereas AM involves inner offsets. Therefore, as a first step, the original CAD model should be transformed into its negative version, which boils down to a simple Boolean operation (Figure 4). We successfully tested a rear-view mirror housing with SolidCAM (SolidCAM, 2022), using its built-in functionalities to slice the model and offset the contours. Unfortunately, a CAM system is already designed to output G-code, usually as polylines, for a chosen CNC controller, a scheme too rigid for our purpose of controlling all steps in the AM process to maximize output accuracy.

The alternative we put forward is using a commercial NURBS-based CAD system for both direct slicing and path planning and, finally, custom generating the G-code, always in a NURBS framework (Figure 5). This proposal resembles modern approaches to streamline the CAD-to-Print workflow based on developing specific AM kernels. In particular (Dyndrite, 2022), the recent Dyndrite *Accelerated Computation Engine* tries to do for 3D printing what Adobe's PostScript page-description language (Adobe Systems, 1985) did for two-dimensional (2D) printing in the 1980s. Our goal is precisely to generate the final G-code for AM instead of printing on paper. The details of our system implementation are described in the next section.

3. System description

Our implementation was carried out by developing two software modules. First, a module in Grasshopper, the visual programming environment for Rhino3D, complemented with the add-on *Xylinus* (Hoover, 2022). This module performs the following tasks:

- *Slicing*: For simplicity, we used uniform slicing, that is, at constant vertical increments h (Figure 6), as our goal is not testing adaptive slicing but our proposal of a NURBS environment for AM workflow.

Figure 4 Generating a negative version of a CAD model for additive manufacturing path planning using a CAM system

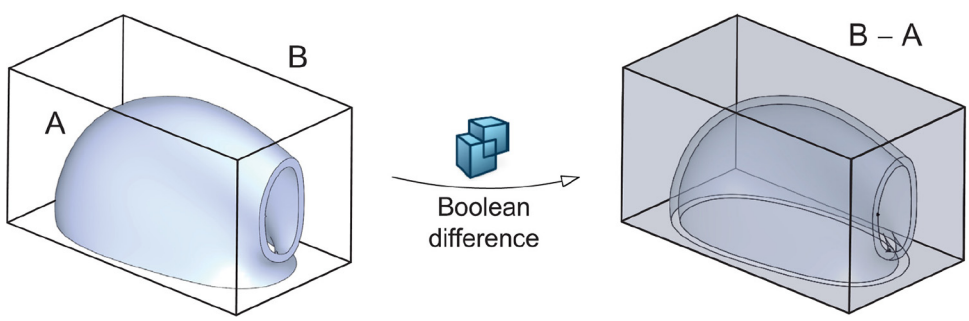


Figure 5 Our approach for generating accurate G-code for additive manufacturing

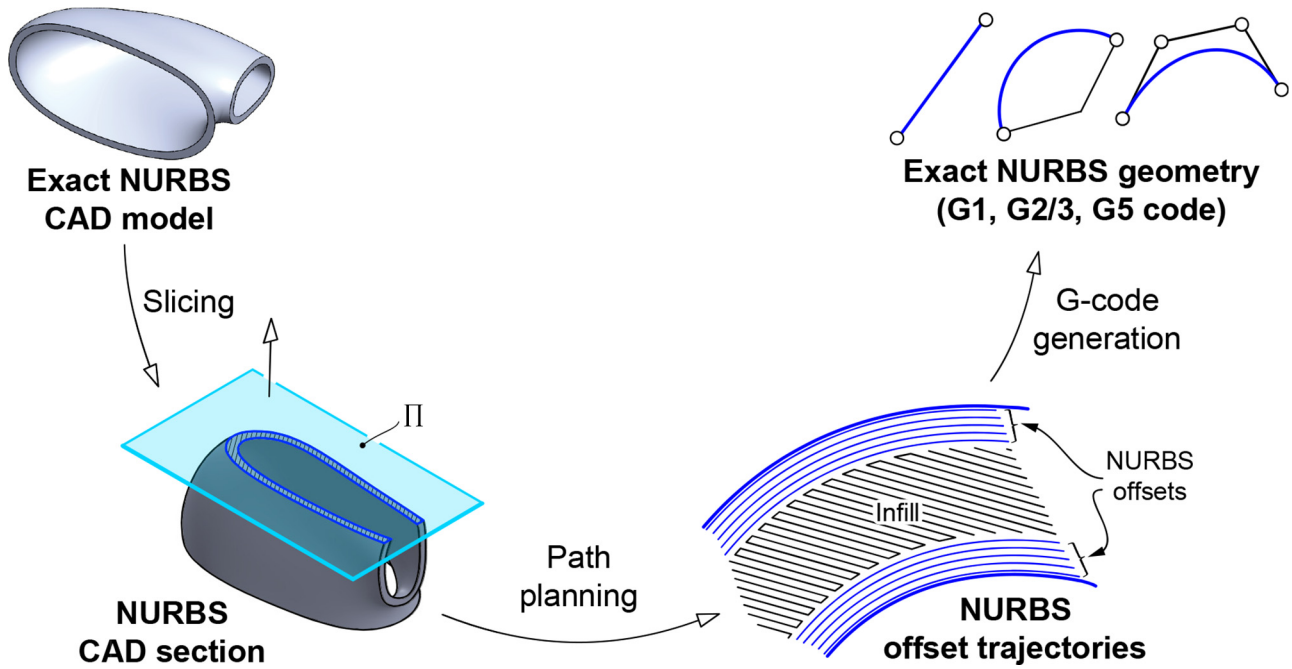
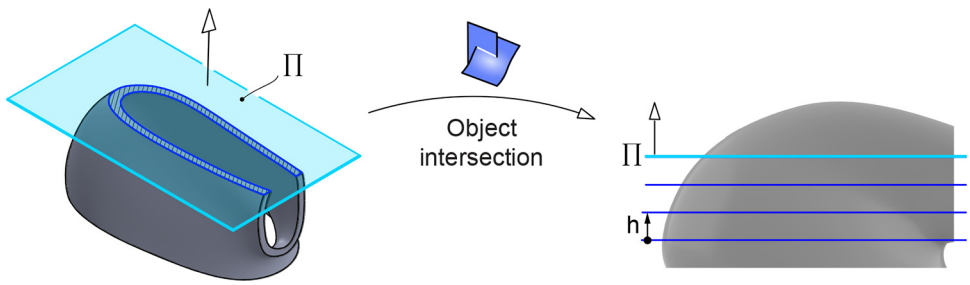


Figure 6 Slicing a CAD object with a horizontal plane Π



- *Path planning*: It involves offset generation for external and internal walls as well as generating the infill pattern.
- *IGES generation*: the resulting trajectories are exported as NURBS via an IGES file.

A second module, implemented in MatLab for convenience, interprets this IGES file and performs three tasks:

1 *Trajectory preprocessing*: At each layer, we concatenate the trajectories in an orderly manner to avoid non-extrusion movements in FFF by identifying coincident endpoints. This step is mandatory because Rhino3D exports the trajectories not necessarily in a connected way.
2 *Geometric processing of trajectories*: Rhino3D resorts to different NURBS curve geometries (polylines, circular arcs, quadratic or cubic integral B-splines). These different curves must be sorted out to take different actions that may include degree-elevation and splitting into Bézier cubics for the next operation.
3 *G-code generation*: In addition to geometric data (points and trajectory lengths) required in G-code commands, we add a header with the appropriate parameter settings (such as feed and flow rate or nozzle and bed temperature).

In the first module, Grasshopper takes care of all geometric operations, so there is no need to program complex routines for slicing or offset generation. In particular, offsetting (Elber *et al.*, 1997; Maekawa, 1999; Patrikalakis and Maekawa, 2002) is a challenging geometric problem in path planning, which in addition involves trimming (Figure 7), also handled by Rhino3D. The add-on *Xylinus* includes a *Curve to G-code* component, but unfortunately, it transforms all paths to polylines, as G1 code. Hence, there is the need for a second module for generating our customized G-code to avoid the error of these polyline approximations and replace them by exact conversion into G2/3 and G5 commands.

4. Geometric processing of non-uniform rational B-splines trajectories and G-code generation

4.1 Transforming IGES entities into G-code commands

As Rhino3D is a NURBS-based package, the IGES file it generates describes all trajectories as 2D NURBS curves (i.e. IGES entity 126). We checked out that all the trajectories have degree $n \leq 3$ and fall into one of these categories (Figure 8):

Figure 7 Offset trimming in path planning

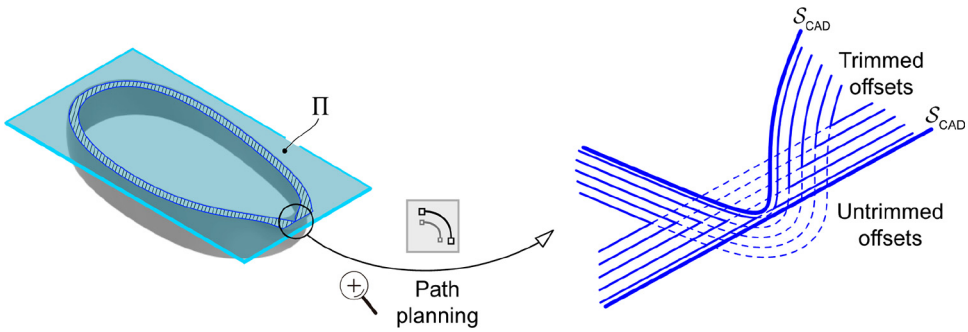
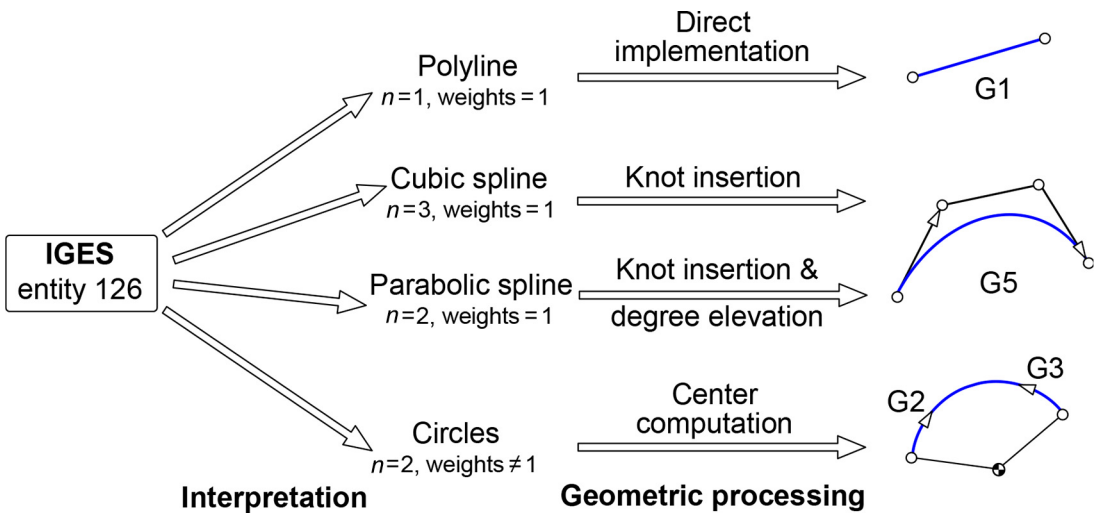


Figure 8 Transforming an IGES entity 126 into G-code commands



- *Polylines*: Polynomial NURBS (unit weights) are of degree $n = 1$. They are used for linear contours and infill patterns.
- *Cubic polynomial spline*: these include cubic NURBS with unit weights. They provide a unified representation to accommodate curves resulting from different geometry processing operations (in particular, offsetting).
- *Parabolic polynomial spline*: these include quadratic NURBS ($n = 2$) with unit weights. These curves are also used for offset approximation.
- *Circles*: these include quadratic NURBS ($n = 2$), with non-unit weights. They are used to represent holes, circular walls and fillets.

All this geometry admits an exact translation into a G-code command (G1, G2/3 and G5) without any approximation. However, aside from the simple case of a polyline, the geometric data the G-code needs do not coincide with the NURBS description provided in IGES (in terms of control points). Consequently, some geometry processing is required to translate the NURBS description into G-code. In addition, with FFF and a Marlin firmware in mind, the G-code requires a field with the accumulated length e of the extruded filament, which implies computing the arc length of each segment.

4.2 Splitting a cubic polynomial non-uniform rational B-splines into Bézier cubics via knot insertion

A cubic Bézier curve is implemented exactly as a G5 command. For a cubic with control points $\{\mathbf{b}_i\}_{i=0}^{3}$, this command requires the endpoint $\mathbf{b}_3 = \{x_3, y_3\}$, as well as the vectors $\mathbf{u} = \{u_x, u_y\}$ and $\mathbf{v} = \{v_x, v_y\}$ defining the end control legs $\mathbf{u} = \mathbf{b}_1 - \mathbf{b}_0$ and $\mathbf{v} = \mathbf{b}_3 - \mathbf{b}_2$, following the syntax:

$$\text{G5 I}u_x \text{J}u_y \text{P}v_x \text{Q}v_y \text{X}x_3 \text{Y}y_3 \text{E}e.$$

However, free-form curves resulting from geometry processing are represented as cubic polynomial NURBS $\mathbf{d}(u)$ instead of cubic Bézier curves. Consequently, to find the Bézier points $\mathbf{b}_i$, the spline curve $\mathbf{d}(u)$ must be split into its cubic Bézier components by inserting all inner knots (Hoschek and Lasser, 1993). A general polynomial B-spline curve of degree n (order $n + 1$) is defined by a knot vector $\mathbf{u} = \{u_0, \ldots, u_i, \ldots, u_K\}$ and $N + 1$ control points (de Boor points) $\{\mathbf{d}_i\}_{i=0}^{N}$, $N = K - n - 1$, $N \geq n$. In the IGES file exported from Rhino, B-spline curves have a *nonperiodic* knot vector (aka *clamped* or *open*), which means that the end knots have multiplicity $n + 1$, following the widespread convention in the CAGD community, as in the treatises on NURBS (Piegl and Tiller, 1997; Rogers, 2001) and the textbook on CAGD (Hoschek and Lasser, 1993):

$$\mathbf{u} = \{u_0 = \ldots = u_n, \ldots, u_{K-n} = \ldots = u_K\},$$

We adhere to this notation, warning that the Rhino3D command *Analyze* > *Diagnosis* > *List* shows only multiplicity n at the end knots, as in Farin's books on NURBS and CAGD (Farin, 1999; Farin, 2002).

Next, we customize the general knot-insertion algorithm to our particular goal for cubics (n = 3), namely, taking an already existing inner knot u_i and inserting it up to multiplicity three:

- If the existing knot u_i is *simple* (i.e. of multiplicity one), then replace the original de Boor point $\mathbf{d}_{i-2}$ with three new control points $\mathbf{d}_A$, $\mathbf{d}_{AB}$, $\mathbf{d}_B$ [Figure 9(a)], computed as barycentric combinations of the original points $\mathbf{d}_{i-3}$, $\mathbf{d}_{i-2}$, $\mathbf{d}_{i-1}$:

$$\mathbf{d}_A = (1-\alpha_A)\mathbf{d}_{i-3} + \alpha_A\mathbf{d}_{i-2},\ \alpha_A = \frac{u_i - u_{i-2}}{u_{i+1} - u_{i-2}},$$

$$\mathbf{d}_B = (1-\alpha_B)\mathbf{d}_{i-2} + \alpha_B\mathbf{d}_{i-1},\ \alpha_B = \frac{u_i - u_{i-1}}{u_{i+2} - u_{i-1}},$$

$$\mathbf{d}_{AB} = (1-\alpha_{AB})\mathbf{d}_A + \alpha_{AB}\mathbf{d}_B,\ \alpha_{AB} = \frac{u_i - u_{i-2}}{u_{i+1} - u_{i-2}}.$$

- If the existing knot already has multiplicity two, so that $u_{i-1} = u_i$, then insert only the new control point $\mathbf{d}_{AB} = (1-\alpha_{AB})\,\mathbf{d}_{i-3} + \alpha_{AB}\mathbf{d}_{i-2}$ on the segment $\mathbf{d}_{i-3}\mathbf{d}_{i-2}$, where α_{AB} denotes the quotient above.

The sought consecutive Bézier segments, making the original cubic spline, span between pairs of consecutive distinct knots and have Bézier points $\mathbf{b}_i$ coinciding with sequences of four consecutive de Boor points. Figure 9(b) shows an example, a cubic B-spline $\mathbf{d}(u)$ with original knot vector $\mathbf{u}$ = {0,0,0,0,1/3,2/3,1,1,1,1}. Inserting the internal knots, up to multiplicity three, furnishes the three Bézier segments I, II and III making up $\mathbf{d}(u)$, which meet at points $\mathbf{d}_2$, $\mathbf{d}_5$ of the refined B-spline representation. Thus, segments I, II and III have Bézier polygons $\{\mathbf{d}_0,\mathbf{d}_1,\mathbf{d}_2,\mathbf{d}_3\}$, $\{\mathbf{d}_3,\mathbf{d}_4,\mathbf{d}_5,\mathbf{d}_6\}$ and $\{\mathbf{d}_6,\mathbf{d}_7,\mathbf{d}_8,\mathbf{d}_9\}$, respectively.

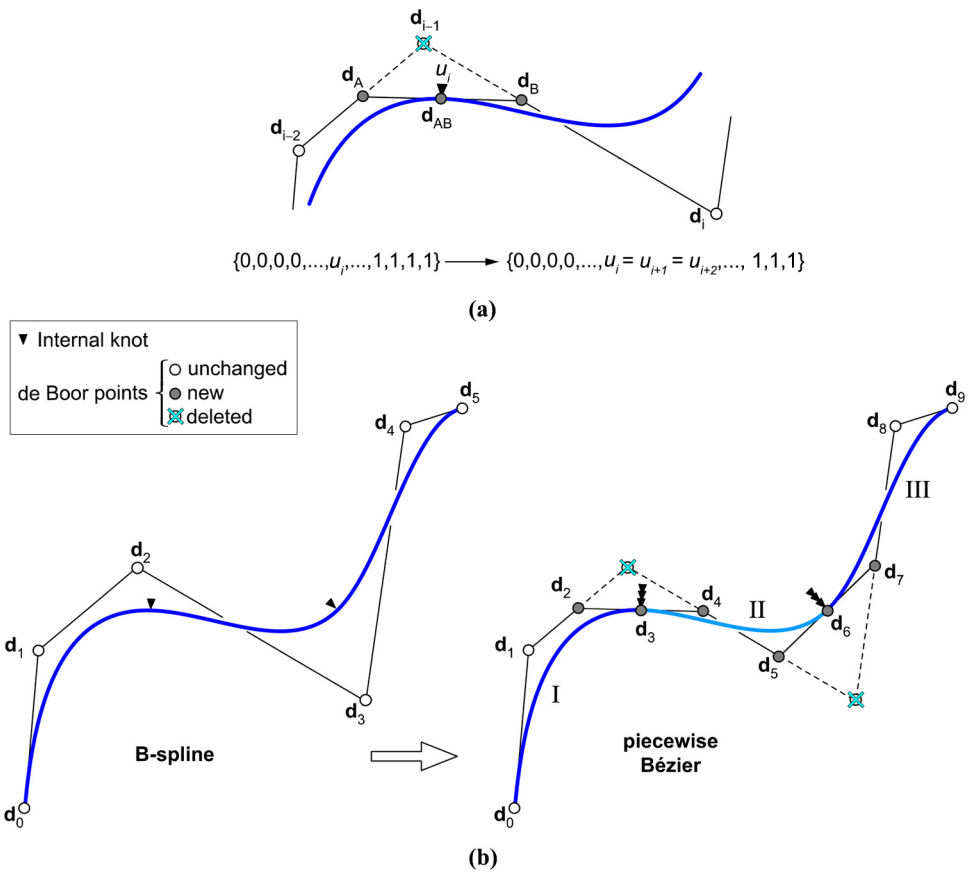
4.3 Transforming quadratic polynomial non-uniform rational B-splines

Some geometry processing operations in Rhino3D yield a quadratic (n = 2) polynomial B-spline instead of cubic. To transform this curve into Bézier cubics, we proceed in two steps:

1 Insert all simple inner knots u_i, from multiplicity one to multiplicity two, to split the curve into quadratic Bézier segments. As in the cubic case, we insert a new de Boor point $\mathbf{d}_C$ between the original points $\mathbf{d}_{i-2}$, $\mathbf{d}_{i-1}$ [Figure 10(a)]:

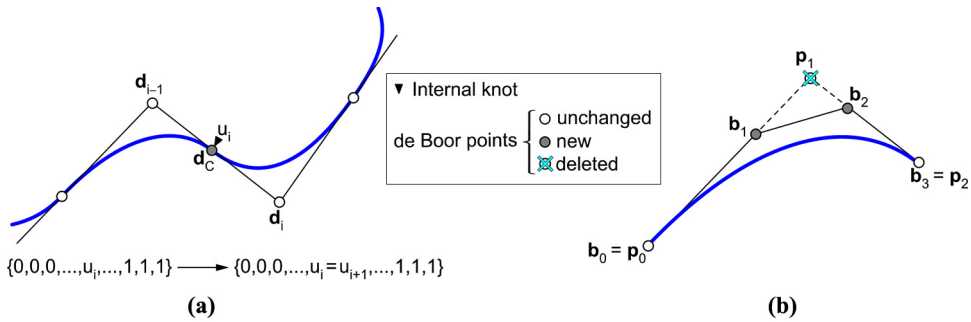
$$\mathbf{d}_C = (1-\alpha_C)\mathbf{d}_{i-2} + \alpha_C\mathbf{d}_{i-1}, \alpha_C = \frac{u_i - u_{i-1}}{u_{i+1} - u_{i-1}}.$$

Figure 9 Cubic polynomial B-spline curve



Notes: (a) Inserting an existing inner knot u_i; (b) splitting the spline into its Bézier segments, by inserting all inner knots

Figure 10 Transforming a quadratic B-spline curve into Bézier cubics



Notes: (a) Duplicating an existing internal knot; (b) degree elevation

2 Take triples $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$ of consecutive de Boor points, making up each quadratic Bézier (i.e. a parabola) and degree elevate it up to degree $n = 3$ [Figure 10(b)], to obtain the Bézier points $\mathbf{b}_i$ of its cubic representation:

$$\mathbf{b}_0 = \mathbf{p}_0, \quad \mathbf{b}_1 = \frac{1}{3}(\mathbf{p}_0 + 2\mathbf{p}_1)$$
$$\mathbf{b}_3 = \mathbf{p}_2, \quad \mathbf{b}_2 = \frac{1}{3}(\mathbf{p}_2 + 2\mathbf{p}_1)$$

4.4 Circular arcs

Circular arcs are implemented exactly as a G2 (clockwise) or G3 (counterclockwise) G-code command (Figure 11). For a circle centered at $\mathbf{C} = \{x_\mathbf{C}, y_\mathbf{C}\}$ and traced from the current point $\mathbf{A} = \{x_\mathbf{A}, y_\mathbf{A}\}$ to the endpoint $\mathbf{B} = \{x_\mathbf{B}, y_\mathbf{B}\}$, the I, J form has the following syntax:

$$\mathrm{G2/3\ X}x_\mathbf{B}\ \mathrm{Y}y_\mathbf{B}\ \mathrm{I}(x_\mathbf{C}{-}x_\mathbf{A})\ \mathrm{J}(y_\mathbf{C}{-}y_\mathbf{A})\ \mathrm{E}e.$$

In contrast, in the IGES file that Rhino3D generates, circular arcs are represented as quadratic NURBS, with all inner knots of multiplicity two, that is, a piecewise rational Bézier form. Thus, we compute the required data as follows:

Points **A** and **B**: They coincide with the first and last de Boor points $\mathbf{A} = \mathbf{d}_0$ and $\mathbf{B} = \mathbf{d}_N$, respectively. Center ***C***: Given by the intersection between a pair of lines $\mathcal{L}_i$ and $\mathcal{L}_j$ through points $\mathbf{d}_i$ and $\mathbf{d}_j$ (i, j even) and orthogonal to control polygon, which is tangent to the circle at these points. Two consecutive lines are guaranteed not to be parallel in this piecewise rational Bézier form (with positive weights).

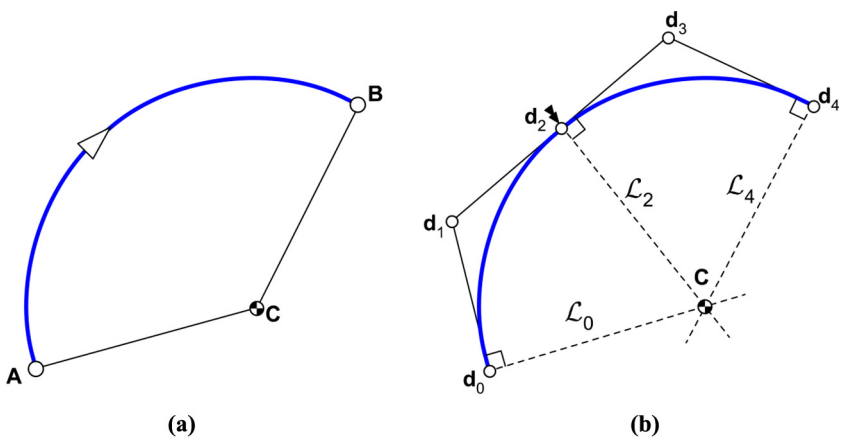
4.5 Computing the length of the extruded filament for fused filament fabrication

As already observed, all G-code instructions considered include a field with the accumulated length e of the extruded filament, which means computing the incremental length Δe of the filament consumed. Figure 12 depicts the geometry involved in extruding a strand of length L and layer height h, consuming a length Δe. A constant density of the deposited material implies that the volume V of the filament consumed equals that of the layer between endpoints. Assuming constant sections S_f and S_L of filament and layer, Δe turns out to be proportional to the trajectory length L:

$$V = S_\mathrm{f}\Delta e = S_L L \quad \rightarrow \quad \Delta e = L\frac{S_L}{S_\mathrm{f}}$$

The values S_f and S_L are obtained as follows:

Figure 11 Circular arc



Notes: (a) Data required in the G2 command; (b) finding the center **C** from a quadratic non-uniform rational B-splines form

Figure 12 Extruding a strand of length L and layer height h, consuming a filament length Δe

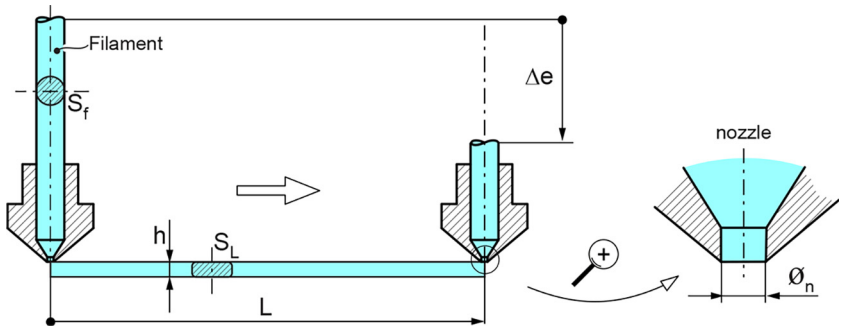
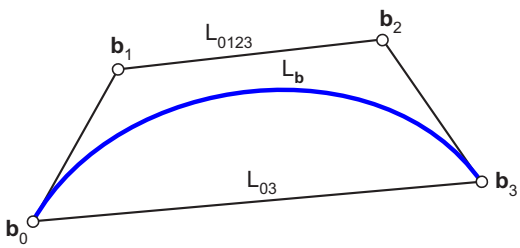


Figure 13 Bézier cubic: Exact arc length $L_{\mathbf{b}}$, Chord L_{03} and control polygon length L_{0123}



- $S_f = \pi \cdot Ø_f^2/4$ (circular section of a filament with diameter $Ø_f$).
- $S_L = kh$ is proportional to the layer height h, where the constant k must be estimated in terms of nozzle diameter $Ø_n$ and printing parameters (Hodgson, 2022).

Regarding L, straightforward trigonometry yields it for linear (G1) or circular arcs (G2/3). For cubics (G5), the exact arc length $L_{\mathbf{b}}$ of a Bézier segment $\mathbf{b}(u)$ involves integrating the module of its derivative, a quadratic curve $\mathbf{b}'(u) = \mathbf{q}(u)$ with Bézier points $\mathbf{q}_i$:

$$L_{\mathbf{b}} = \int_0^1 \|\mathbf{q}(u)\| du, \mathbf{q}_i = 3(\mathbf{p}_{i+1} - \mathbf{p}_i),\ i = 0, 1, 2.$$

Unfortunately, this integral incurs high computational cost (Guenter and Parent, 1990). Thus, for practical purposes, Gravesen (1997) puts forward a simple approximation, namely, the average $L = (L_{03} + L_{0123})/2$ between the length L_{03} of the chord $\mathbf{b}_0\mathbf{b}_3$ and the length L_{0123} of the control polygon (Figure 13). We checked that his approximation suffices for the short Bézier segments we are dealing with, making up the paths Rhino3D generates.

5. Implementation in commercial firmware and experimental results

5.1 Implementation in commercial firmware

The practical implementation of the proposed approach requires some customizing of the firmware controlling the 3D printer. This software reads the G-code commands and translates them into the electrical commands for the stepper motor driving the print head, among other tasks. We focus on the commercial firmware Marlin (2022) for its popularity and, more importantly, open-source character, allowing customization by editing the configuration files *Configuration.h*, *Configuration_adv.h* (header files in C++).

We used Marlin 1.1, supporting not only G2/3 (circular arc) but also G5 (Bézier cubic) commands. However, by default, these commands may not be activated, but this is fixed by editing the configuration files in the following way (Dorado-Vicente *et al.*, 2019):

- If an editable version of the firmware provided by the printer's manufacturer is available, in the file *Configuration_adv.h*, then simply check out that the following lines are activated by removing the comment tag (double forward slash) if necessary:
  //#define ARC_SUPPORT
  //#define BEZIER_CURVE_SUPPORT
- If not, then download a generic Marlin firmware and adapt it to the specific machine. This customization involves activating not only these two lines but also the more complex task of setting several parameters in the files *Configuration.h*, *Configuration_adv.h*.

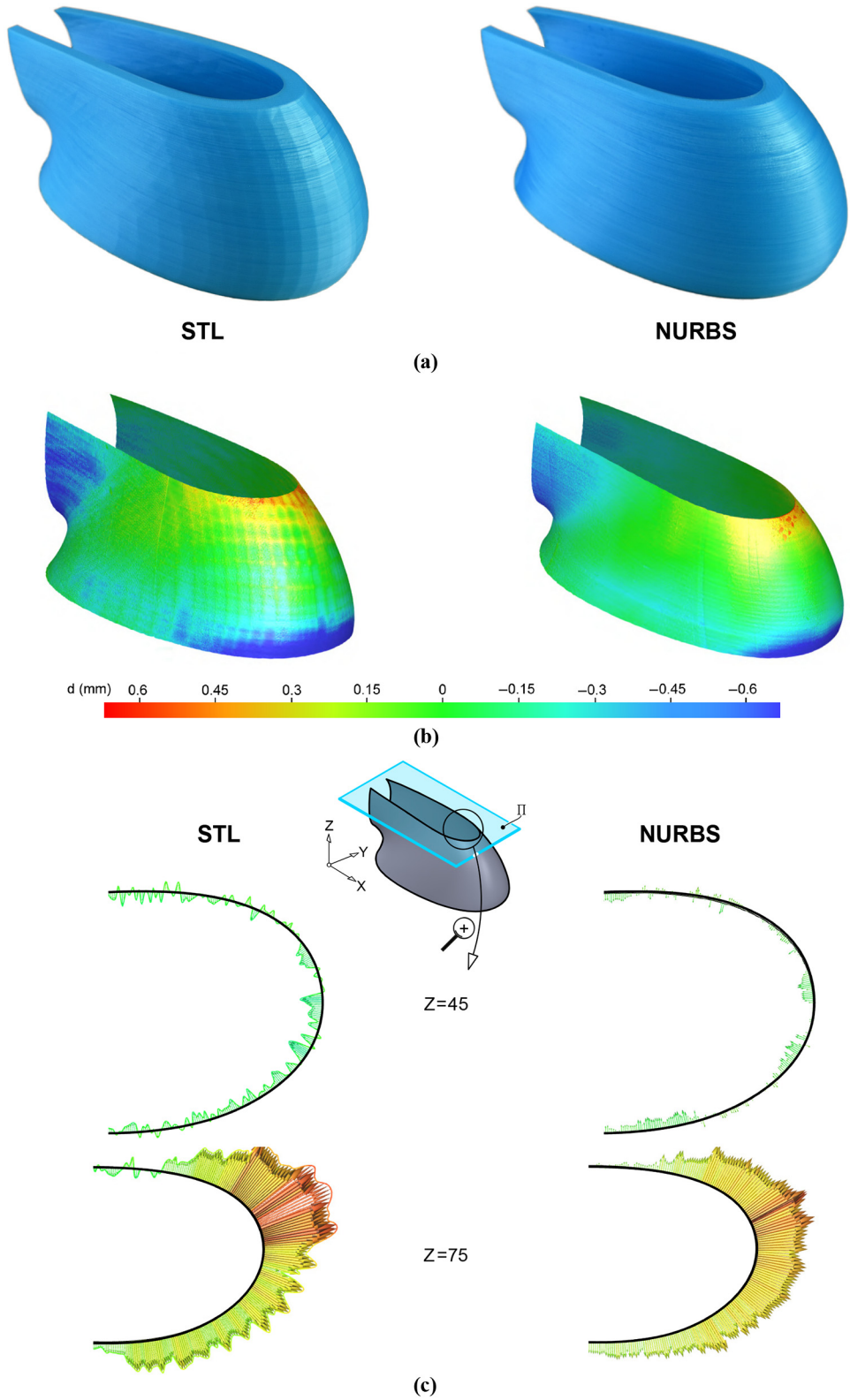
Once the configuration files have been edited, recompile to generate the executable firmware, for instance, using Arduino IDE and upload the firmware in the 3D printer. We have implemented procedure a) using BQ WitBox 3D and CreatBot PEEK-300 printers and procedure b) with Ultimaker 2+.

5.2 Experimental results

We tested our proposal with free-form printouts using PLA filament, feed rate 50 mm/s and constant layer height $h = 0.15$ mm. As tolerance distribution (Ma *et al.*, 2004), we opted for a *mixed tolerance*, that is, slicing the model at the middle of each layer. This choice corresponds to the recommended *middle* setting of the *slicing tolerance* parameter in the *Cura* slicing application (Ultimaker, 2022).

The example of Figure 14, a rear-view mirror housing designed with SolidWorks and printed with CreatBot PEEK-300, compares the traditional method (via STL file) and our direct NURBS-based approach. We generated in SolidWorks both the STL file and the .SLDPRT (SolidWorks) CAD file imported into Rhino3D for slicing. Table 1 lists the corresponding file sizes, along with those of the resulting G-

Figure 14 Example (CreatBot PEEK-300) comparing traditional additive manufacturing via STL and our non-uniform rational B-splines-based workflow



Notes: Actual pictures and a) deviations from the exact CAD model (outer surface); b) three-dimensional plot; and c) partial horizontal sections ($z = 45$, $z = 75$ mm) with magnified deviations

Table 1 File sizes for the STL and non-uniform rational B-splines models of Figure 14 (rear-view mirror housing)

Model	File size and type	G-code commands	G-code size
STL	205 KB (*.stl)	646 KB	20.8 MB
NURBS	320 KB (*.sldprt)	586 KB	41.2 MB

code files fed to the controller and the number of G-code lines they contain. For STL and CAD files with comparable sizes, the G-codes have roughly the same number of lines, albeit the file size for NURBS doubles that of the STL case. Note that the G5 instruction (Bézier cubic) is longer than the G1 (line segment), as it requires three pairs of coordinates instead of just one.

As Figure 14(a) shows, the STL facets are visible, whereas the NURBS-based approach results in a noticeable improvement in surface quality. In large-scale printing (Roschli *et al.*, 2019), these artifacts would be even more noticeable, and large STL files would be needed to achieve a precision comparable to that of the 3D printer. Figures 14(b) and (c) depict the dimensional deviations d between the outer surface of the STL/NURBS model and that of the theoretical CAD model, with a common color code. We display a 3D plot [Figure 14(b)] as well as two partial horizontal sections z = constant [Figure 14(c)] with magnified errors, showing that the NURBS printout enjoys a notably smoother error distribution. The models were measured using a Hexagon GLOBAL-S Blue coordinate measuring machine. The sensor was a Hexagon HP-L-10.10 laser scanner with an 8 μm probing form error capturing up to 600 K points per second. The Hexagon PCDMIS 22021.1 scanning software was used to obtain the 3D color map and sections.

6. Conclusions

The traditional AM workflow, involving an initial polygonization of the object via STL, does not meet the requirements for truly functional parts regarding quality and precision, especially in large-scale 3D printing. This software limitation wastes the possibilities of existing AM hardware and precludes them from furnishing such functional parts. As Dyndrite (2022) highlights, modern hardware has outpaced software. The situation epitomizes a case of technological *lock-in* (Arthur, 1989), whereby a well-established standard becomes obsolete yet remains in use for compatibility reasons and the reluctance to change, trapping an industry (Farrell and Saloner, 1985). In the long term, the savings of switching to a modern standard would outperform the initial investments required in updating existing software. To break out of the locked-in STL and ease the transition process, we propose switching to the current NURBS standard, already used in the initial description of the CAD model.

The fundamental tenet in our proposal is the avoidance of multiple representations in the AM workflow by adhering to the NURBS standard at all steps, namely, slicing, path planning and G-code generation. Both slicing and offsetting, the most complex geometry operation in path planning, are already available, in a reliable way, in any NURBS-based CAD system. Therefore, there is no need to develop ad hoc procedures, as we can access these capabilities through a suitable programming environment. Direct slicing and offsetting could also be accessed by interacting directly with an existing commercial kernel, like the already mentioned Dyndrite *Accelerated Computation Engine* (Dyndrite, 2022), built with AM in mind.

Our main contribution is sticking to the NURBS standard at the last step of AM, namely, G-code generation, a possibility overlooked in both the literature and commercial applications. To this aim, we exploit the possibilities of exact conversion of NURBS trajectories into G2/3 (circular arcs) and G5 (cubic Bézier curves) commands, available in existing firmware controlling 3D printers. As these NURBS trajectories are usually restricted to polynomial cubic or quadratic splines and quadratic circles, the exact conversion is readily performed via standard NURBS geometry processing (knot-insertion and degree-elevation). General NURBS curves, of arbitrary degree and weights, could be handled with high-fidelity approximation techniques (Piegl *et al.*, 2007), which incur negligible errors. Remarkably, this idea of accommodating non-linear trajectories to either circles or Bézier cubics resembles that implemented in the PostScript page-description language (Adobe Systems, 1985), yet it also admits ellipses, a geometry not available as a G-code. PostScript triggered the 2D printing revolution in the 1980s, whereas NURBS-based geometry at all steps of the AM process may help consolidate the current AM revolution.

We have implemented our ideas using the popular NURBS-based Rhino3D package as a testbed platform, with Grasshopper as a programming environment, but other combinations allowing customization would be feasible. As commercial firmware supporting the required G-code commands, we used Marlin. Preliminary examples manufactured with different 3D FFF printers corroborate the feasibility of our proposal and the resulting improvement in part quality.

References

Adobe Systems (1985), *PostScript Language Tutorial and Cookbook*, Addison-Wesley, Reading, MA.

Allen, G. (2007), "Geometric modeling problems in industrial CAD/CAM/CAE". *Technical report, Siemens PLM Software*, available at: www.scribd.com/document/233753777/Allen

Allen, G. (2013), "Geometry in CAD systems: past, present, and future", *presented at the 2013 SIAM Conference on Geometric and Physical Modeling (GD-SPM13)*, 11–14 Nov. 2013, Denver, CO.

Arthur, W.B. (1989), "Competing technologies, increasing returns, and lock-in by historical events", *The Economic Journal*, Vol. 99 No. 394, pp. 116-131.

Bertacchini, F., Bilotta, E., Carnì, L.C., Demarco, F., Pantano, P., Scuro, C. and Lamonaca, F. (2021), "Preliminary study of an innovative method to increase the accuracy in direct 3D-printing of NURBS objects", *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0 & IoT)*, pp. 94-98.

Carleberg, P. (1994), "Product model driven direct manufacturing", *Proceedings of the 1994 International Solid Freeform Fabrication Symposium, Austin, TX*, pp. 270-276.

Dorado-Vicente, R., Medina-Sánchez, G., García-Collado, A., Carou, D. and Pérez, M. (2019), "How to use and compare interpolations schemes in fused deposition modeling", *Procedia Manufacturing*, Vol. 41, pp. 343-350.

Dyndrite (2022), "Dyndrite", available at: www.dyndrite.com (accessed 8 August 2022).

Elber, G., Lee, I.-K. and Kim, M.-S. (1997), "Comparing offset curve approximation methods", *IEEE Computer Graphics and Applications*, Vol. 17 No. 3, pp. 62-71.
Ezair, B., Fuhrmann, S. and Elber, G. (2018), "Volumetric covering print-paths for additive manufacturing of 3D models", *Computer-Aided Design*, Vol. 100, pp. 1-13.
Farin, G. (1999), *NURBS: From Projective Geometry to Practical Use*, 2nd ed., AK Peters, Natick, MA.
Farin, G. (2002), *Curves and Surfaces for Computer Aided Geometric Design*, 5th ed., Morgan Kaufmann, San Francisco, CA.
Farouki, R.T. and König, T. (1996), "Computational methods for rapid prototyping of analytic solid models", *Rapid Prototyping Journal*, Vol. 2 No. 3, pp. 41-48.
Farrell, J. and Saloner, G. (1985), "Standardization, compatibility, and innovation", *The RAND Journal of Economics*, Vol. 16 No. 1, pp. 70-83.
Feng, J., Fu, J., Lin, Z., Shang, C. and Li, B. (2018), "Direct slicing of T-spline surfaces for additive manufacturing", *Rapid Prototyping Journal*, Vol. 24 No. 4, pp. 709-721.
Gao, W., Zhang, Y., Ramanujan, D., Ramani, K., Che, Y., Williams, C.B., Wang, C.C.L., Shin, Y.C., Zhang, S. and Zavattieri, P.D. (2015), "The status, challenges, and future of additive manufacturing in engineering", *Computer-Aided Design*, Vol. 69, pp. 65-89.
Gravesen, J. (1997), "Adaptive subdivision and the length and energy of Bézier curves", *Computational Geometry*, Vol. 8 No. 1, pp. 13-31.
Guenter, B. and Parent, R. (1990), "Computing the arc length of parametric curves", *IEEE Computer Graphics and Applications*, Vol. 10 No. 3, pp. 72-78.
Hodgson, G. (2022), "Slic3r manual", available at: https://manual.slic3r.org/advanced/flow-math (accessed 8 August 2022).
Hoover, R. (2022), "Xylinus", available at: http://grasshopperdocs.com/addons/xylinus.html (accessed 8 August 2022).
Hoschek, J. and Lasser, D. (1993), *Fundamentals of Computer Aided Geometric Design*, AK Peters, Wellesley, MA.
Jamieson, R. and Hacker, H. (1995), "Direct slicing of CAD models for rapid prototyping", *Rapid Prototyping Journal*, Vol. 1 No. 2, pp. 4-12.
Kumar, V. and Dutta, D. (1997), "An assessment of data formats for layered manufacturing", *Advances in Engineering Software*, Vol. 28 No. 3, pp. 151-164.
Ma, W., But, W.-C. and He, P. (2004), "NURBS-based adaptive slicing for efficient rapid prototyping", *Computer-Aided Design*, Vol. 36 No. 13, pp. 1309-1325.
Maekawa, T. (1999), "An overview of offset curves and surfaces", *Computer-Aided Design*, Vol. 31 No. 3, pp. 165-173.
Marlin (2022), "Marlin firmware", available at: https://marlinfw.org (accessed 8 August 2022).
Oropallo, W. and Piegl, L.A. (2016), "Ten challenges in 3D printing", *Engineering with Computers*, Vol. 32 No. 1, pp. 135-148.
Pandey, P.M., Reddy, N.V. and Dhande, S.G. (2003), "Slicing procedures in layered manufacturing: a review", *Rapid Prototyping Journal*, Vol. 9 No. 5, pp. 274-288.
Patrikalakis, N.M. and Maekawa, T. (2002), *Shape Interrogation for Computer Aided Design and Manufacturing*, Springer, Cambridge, MA.
Paul, R. and Anand, S. (2015), "A new Steiner patch based file format for additive manufacturing processes", *Computer-Aided Design*, Vol. 63, pp. 86-100.
Piegl, L. and Tiller, W. (1997), *The NURBS Book*, 2nd ed., Springer, Berlin, Heidelberg.
Piegl, L.A., Rajab, K. and Smarodzinava, V. (2007), "High fidelity conversion of NURBS curves for data exchange", *Computer-Aided Design and Applications*, Vol. 4 No. 5, pp. 683-693.
Qin, Y., Qi, Q., Scott, P.J. and Jiang, X. (2019), "Status, comparison, and future of the representations of additive manufacturing data", *Computer-Aided Design*, Vol. 111, pp. 44-64.
Rogers, D.F. (2001), *An Introduction to NURBS. With Historical Perspective*, Morgan Kaufmann, San Francisco, CA.
Roschli, A., Gaul, T., Boulger, A.M., Post, B.K., Chesser, P. C., Love, L.J., Blue, F. and Borish, M. (2019), "Designing for big area additive manufacturing", *Additive Manufacturing*, Vol. 25, pp. 275-285.
Sánchez-Reyes, J. and Chacón, J.M. (2015), "A polynomial Hermite interpolant for C2 quasi arc-length approximation", *Computer-Aided Design*, Vol. 62, pp. 218-226.
Shapiro, V. (2002), "Solid modelling", in Farin, G., Hoschek, J. and Kim, M.-S. (Eds), *Handbook of Computer Aided Geometric Design*, Elsevier, Amsterdam, pp. 473-518.
SMS (2022), "Solid modelling solutions", available at: www.smlib.com/products.html (accessed 7 August 2022).
SolidCAM (2022), "SolidCAM", available at: www.solidcam.com (accessed 7 August 2022).
Starly, B., Lau, A., Sun, W., Lau, W. and Bradbury, T. (2005), "Direct slicing of STEP based NURBS models for layered manufacturing", *Computer-Aided Design*, Vol. 37 No. 4, pp. 387-397.
Ultimaker (2022), "Ultimaker cura", available at: https://support.ultimaker.com/hc/en-us/sections/360003548339-Ultimaker-Cura (accessed 7 August 2022).
Xu, G. and Jing, T. (2011), "Research of slicing CAD models with pro/TOOLKIT for integral stereolithography", *Advanced Materials Research*, Vol. 321, pp. 226-229.
Xu, G., Zhang, J., Luo, S. and Jing, J. (2011), "Direct slicing CAD models with SolidWorks for integral stereolithography system", *Advanced Materials Research*, Vols 148/149, pp. 818-821.
Yu, K.M., Wang, Y. and Wang, C.C.L. (2017), "Smooth geometry generation in additive manufacturing file format: problem study and new formulation", *Rapid Prototyping Journal*, Vol. 23 No. 1, pp. 34-43.

Corresponding author

Javier Sánchez-Reyes can be contacted at: Javier.SanchezReyes@uclm.es