

# G-coordinator: G-code 生成の新たな手法とその可能性

谷口朝洋\*

January 4, 2024

G-coordinator: A New Method of G-code Generation and Its Possibility  
Tomohiro TAGUCHI

大阪府立大学 工学域 機械系学類  
機械工学課程 機械力学研究室  
〒 599-8531 大阪府堺市中区学園町 1-1

## Abstract

This study focuses on the novel generation method of G-code, a program loaded into the machine, for use in the fabrication process using MEX-type 3D printers. We have developed an open-source software called G-coordinator (<https://github.com/tomohiron907/G-coordinator>), which directly creates G-code itself using Python and mathematical functions, enabling the construction of three-dimensional shapes. This approach, employing a path generation method distinct from traditional "slicing," allows for more precise adjustment of printing conditions. Furthermore, it facilitates the realization of shapes that were previously challenging and the straightforward implementation of mathematical shapes. In this paper, we elucidate the features of G-coordinator and provide an overview of its fabrication method and implementation using the gcoordinator library. Additionally, we delve into specific use cases that showcase the application of these techniques.

keywords: G-code, 3D Printing, Python, Open-source

## 要旨

本研究は、MEX型3Dプリンタを用いて造形を行う際に、機械に読み込ませるプログラムであるG-codeの新たな生成方法に関するものである。Pythonと数学的な関数を用いてG-codeそのものを直接作成し、3次元形状を構築することできるG-coordinator(<https://github.com/tomohiron907/G-coordinator>)というオープンソースソフトウェアを開発した。従来の「スライス」とは異なる手法によるパス生成により、より精密な印刷条件の調整が可能になるとともに、いままでは難しかった形状や、数理的な形状の容易な実現が期待できる。本論文では、G-coordinatorの特徴を述べるとともに、その造形方法やgcoordinatorライブラリを用いた実装方法についてその概要を述べる。また、それらの技術を用いた具体的なユースケースについても詳述する。

キーワード: G-code, 3D プリント, Python, オープンソース

# 1 はじめに

## 1.1 G-coordinator とは

G-coordinator は、Python を用いて 3D プリンタ用 G-code を生成できるソフトウェアである。従来のスライスソフトによる G-code 生成と異なり、造形から印刷条件の調整まで 3D プリントに必要な工程の全てを G-coordinator ひとつで完結させることができる。Windows, Mac に対応した GUI アプリケーションに加えて、Python 用のライブラリも公開している。どちらも Python を用いて造形と印刷条件の設定を行うが、GUI アプリに関しては、Python 環境を整える必要がなく、誰でも簡単に G-code を生成することができる。Figure1 に G-coordinator (GUI アプリ) のスクリーンショットを示す。G-code を書き出す手順は主に 4 つのステップから構成される。

1. コードを書いて、造形を決定
2. グラフィックスビューで詳細を確認
3. 印刷条件の設定
4. G-code の出力

## 1.2 背景

一般的に、3D プリンタを用いる際には CAD 等で 3D データを作成し、その 3D データをスライスソフトで処理し、G-code を生成する。スライスソフトは、3D プリンタのノズルパスを自動で算出することができ、多くのユーザにとって、より簡単に G-code を作成することができる。しかし、より細かい制御やマニュアルでの制御を求めるユーザにとっては、スライスソフトでは 3D プリンタの挙動や印刷設定のマニュアル制御ができない。そこで、近年、3D モデルを介さず直接 G-code を生成する G-code Modeling[1] の手法が台頭している。G-coordinator は、G-code Modeling の新たな手法として位置付けられる。

## 1.3 目的

本研究は、広範なユーザに向けた高い自由度を有する G-code モデリングを実現し、これにより従来の CAD やスライサでは到達できなかった造形を具現化することを目的としている。G-coordinator により、3D プリントにおける新たな可能性が開かれ、現行の技術では実現が難しい形状や構造の物体を容易に製造することが可能となる。

## 1.4 G-code とは

G-code は、コンピュータ制御された機械操作を指示するための標準的なプログラミング言語であり、その本質はツール (MEX 型 3D プリンタにおいてはノズル)

の位置を記述する座標列にある。G-code の基本的な構造は、各行においてツールの移動先の座標が指定され、これによってツールの位置が制御される。たとえば、以下は一般的な G-code の例である。

G1 F1000 X100 Y80 Z0.4 E0.5

このコマンドで、機械はノズルを  $(X, Y, Z) = (100, 80, 0.4)$  の座標に 1000mm/min の速さで移動させながらフィラメントを 0.5mm 送る。これらのコマンドが、G-code の内部では、数百から数万行繰り返される。G-code には印刷速度やフィラメント送り量など様々な要素があるものの、これらは、ソフトウェアで自動的に決定できるため、G-code Modeling においてユーザはノズルの移動する座標列のみを考慮すれば良い。

ここで、上記の G-code の性質から、G-coordinator でのモデリングに不可欠な二つの用語を定義する。

- セグメント：ノズルが移動する最小単位の線分。
- パス：樹脂が連続的に押し出されてできる一本の経路。

Figure2 は、6 角形のパスが下から積層されており、一番上のパスの最初のセグメントまでが表示されている。このように、セグメントが集まってパスになり、パスが集まってひとつの造形物になる。

## 2 関連研究

現代の 3D プリントにおいて、G-code Modeling の手法として Rhinoceros とそのプラグイン Grasshopper や FullcontrolGCodeDesigner などが広く利用されている。本章では、これらの手法と G-coordinator との比較に焦点を当て、それらの特徴と差異を明らかにする。

### 2.1 Rhinoceros+Grasshopper

Rhinoceros とそのプラグイン Grasshopper の組み合わせは、G-code Modeling の代表的な手法の一つであり、その特徴は以下の通りである。

オープンソース: 非公開

拡張性: 有り

多軸制御: 有り

インフィル: 一部制限あり

要求スキル: 低い

Rhino+Grasshopper はオープンソースではないが、アドオンやプラグイン等の拡張性は高く、多軸制御やインフィルに関してはアドオンで実現できるものも多い。また、プログラミングの形式がビジュアルプログラミングであることから、プログラミングに慣れていないユーザにも扱いやすいソフトウェアと言える。しかし、ソフトウェアの値段が比較的高価であり、一般的の利用者は容易にそのソフトウェアを試用することが難しい状況にある。

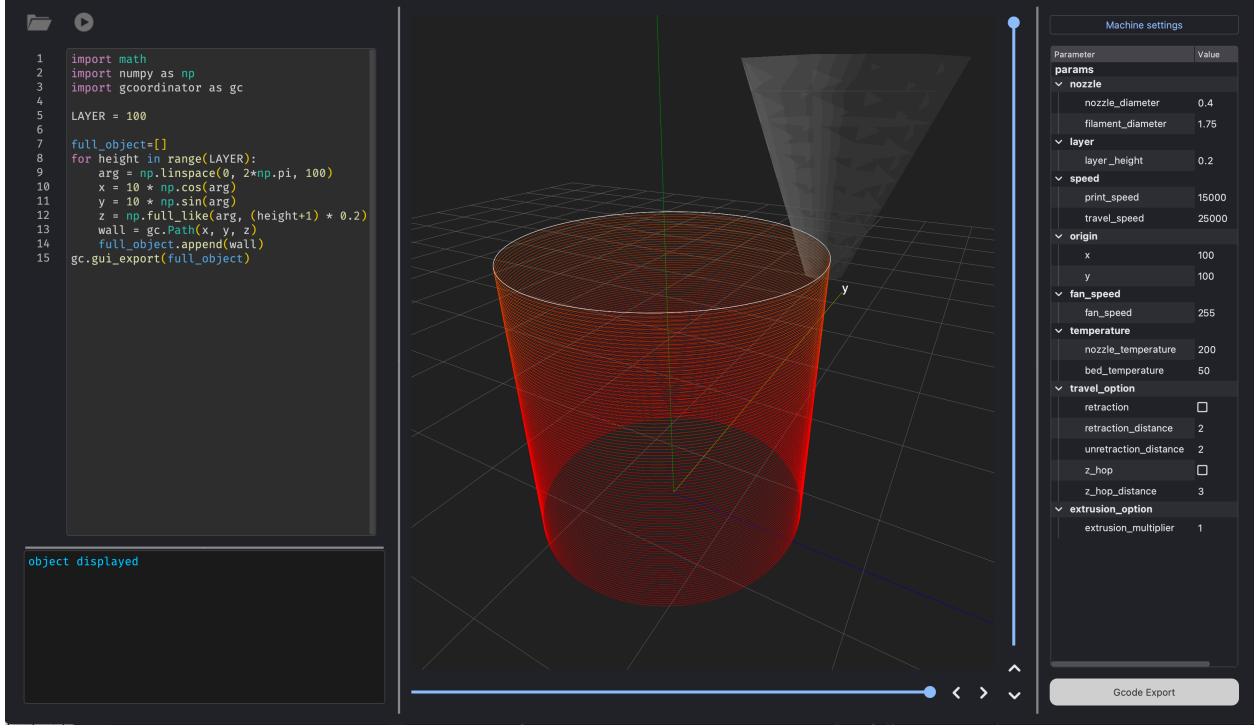


Figure 1: screenshot of G-coordinator

Table 1: ソフトウェアの比較

ソフトウェア	オープンソース	拡張性	多軸印刷	インフィル	スキル要求
Rhino+GH	×	○	△	△	低
FullControl	○	△	×	×	低
G-coordinator	○	○	○	○	高

## 2.2 FullControlGCodeDesigner

FullControlGcodeDesinger[2] は、Excel 上で動作するマクロとしてのソフトウェアである。

オープンソース: 公開

拡張性: 制限あり

多軸制御: 非対応

インフィル: 非対応

要求スキル: 低い

FullControlGcodeDesinger はオープンソースであるが、拡張性には限りがあり、多軸制御やインフィルの生成には対応していない。Excel シートであるため、新たなソフトをインストールする必要がなく、初め G-code Modeling を行うユーザに向いているが、Excel のみで造形を完成させる性質上、複雑な形状や関数の扱いには限界がある。また、その点を克服した python ライブリ、fullcontrol も登場しているが、python 環境をユーザが整える必要があるなどの弱点もある。

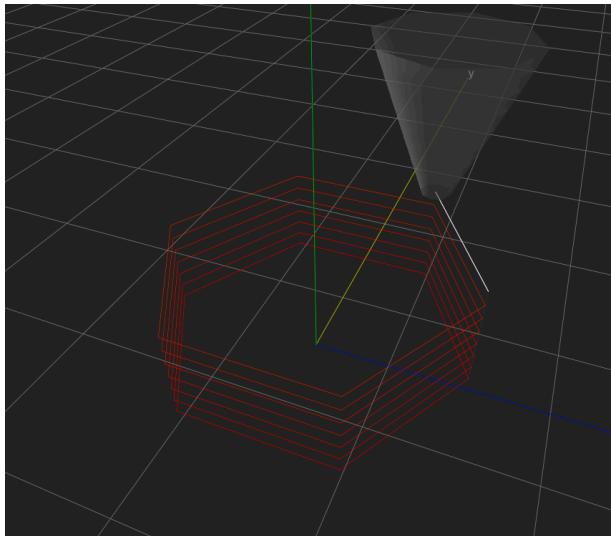


Figure 2: path and segment

## 2.3 G-coordinator

G-coordinator は、 G-code Modeling の新しい手法であり、以下がその特徴である。

オープンソース: 公開

拡張性: 有り

多軸制御: 有り

インフィル: 有り

要求スキル: 高い

G-coordinator はオープンソースであり、あらゆるコードやライブラリを公開している。拡張性に優れ、Python を用いた造形により、様々なデータソースを活用可能である。多軸制御やインフィルの生成にも対応している。しかしながら、G-coordinator での造形には多少の数学的知識とコードのスキルを要求してしまう側面がある。これらの特徴により、G-coordinator は G-code Modeling の新たな展開を可能にしており、従来の手法と比較して優れた柔軟性と拡張性を提供している。

```
1   wall = gc.Path(x, y, z)
2   full_object.append(wall)
3   gc.gui_export(full_object)
```

x, y, z 座標列を引数として、Path オブジェクトのインスタンスを生成する。生成した Path オブジェクトを full\_object に追加する。作成した full\_object を gui\_export 関数に渡すことで、GUI 上で確認することができる。

なお、厳密な表現をすれば、上記のコードで作成できるのは円ではなく、正 99 角形である。これは、arg が 0 から  $2\pi$  までで工数が 100 の等差数列であるから、角度列を順に繋いでできるセグメントは 99 個存在するからである。しかし、半径 10mm の場合、99 分割ほどの段階で、3D プリントにおいては無視できる程度にセグメントは小さくなる。もちろん、分割数を大きくすれば、精度は上昇するが、同時に計算時間も大きくなってしまう。G-code には、円弧補完のコマンド (G02, G03) があるが、G-coordinator では現在は未実装であるので、円を造形する場合はこのような方法で行う。

## 3 実装

### 3.1 G-coordinator の造形方法

G-coordinator で造形を行うための方法を、最も簡易な円柱モデルを用いて述べる。なお、円柱壁の造形コードを補足資料に掲載している。いかに複雑な形状であっても、基本的な造形のプロセスは同じである。G-coordinator では、パスの x, y, z 座標列を計算して、それらの座標列から Path オブジェクトを作成する。より詳細な造形コードの作成に関しては G-coordinator<sup>2</sup> 公式ドキュメントを参照されたい。<sup>3</sup>

#### 3.1.1 円形パスの生成

円柱は円形のパスが縦に造形されたものであることが<sup>7</sup> から、まず円形のパスを生成する。

```
1   import numpy as np
2   import gcoordinator as gc
3   full_object = []
```

まず、numpy ライブラリ、gcoordinator ライブラリをインポートする。次に、full\_object という空のリストを作成する。これは、後に Path オブジェクトを格納するためのものである。

```
1   arg = np.linspace(0, 2*np.pi, 100)
2   x = 10 * np.cos(arg)
3   y = 10 * np.sin(arg)
4   z = np.full_like(arg, 0.2)
```

円を造形するので、0 から  $2\pi$  までの項数 100 の角度についての等差数列 (arg) を作成し、その角度の cos, sin がそれぞれ x, y 座標列になる。z 座表列に関しては、arg と同じ項数で、値が 0.2 (レイヤー厚さ) である数列を用意すればいい。

#### 3.1.2 円柱の造形

円柱を造形するには、前節で述べた円形のパスを縦に積み上げていけば良い。この処理は for ループを用いて用意に記述できる。

```
for height in range(100):
    arg = ...
    x = ...
    y = ...
    z = np.full_like(arg, (height+1) * 0.2)
    wall = ...
    full_object.append(wall)
```

この for ループの中で、height は 0 から 99 まで 1 ずつ増加する。x, y 座標列は円形パスと同じで、z 座標列を height の値に応じて 0.2 ずつ大きくすれば良い。ここで、height に 1 を足してから 0.2 をかけているのは、ループの始めにおいて、height の値が 0 の時に、ノズルの高さが 0.2 にしたいからである。そして、for ループごとにパスオブジェクトを作成し、full\_object に追加すれば良い。即ち、full\_object には、印刷すべきパスが印刷順に 100 個のパスが格納されている。円柱を造形するための造形コード全体を付録に載せる。

#### 3.1.3 パスの変形

gcoordinator ライブラリの Transform クラスのメソッドを利用することで、作成したパスを変形させることができます。

- gc.Transform.move(): パスを平行移動、回転させる

- gc.Transform.rotate\_xy(): XY 平面上で回転させる
- gc.Transform.stretch(): パスを伸縮させる
- gc.Transform.offset(): パスを指定した距離だけオフセットさせる

### 3.2 個別印刷設定

G-coordinator では、以下に示す印刷設定を指定することができる。

1. nozzle
  - (a) nozzle\_diameter [mm]
  - (b) filament\_diameter [mm]
2. layer
  - (a) layer\_height [mm]
3. speed
  - (a) print\_speed [mm/mim]
  - (b) travel\_speed [mm/mim]
4. fan\_speed
  - (a) fan\_speed [0-255]
5. temperature
  - (a) nozzle\_temperature [celcius]
  - (b) bed\_temperature [celcius]
6. travel\_option
  - (a) retraction [true/false]
  - (b) retraction\_distance [mm]
  - (c) unretraction\_distance [mm]
  - (d) z\_hop [true/false]
  - (e) z\_hop\_distance [mm]
7. extrusion\_option
  - (a) extrusion\_miltiplier

G-coordinator では、パラメータツリーで設定したグローバルな印刷設定に対して、個別のパスへの詳細設定をコードから指定することができる。例として、ベッド定着のためにファーストレイヤにおいてのみ、印刷速度を遅くしたいという状況を考える。作成した Path オブジェクトの属性としてこれらの印刷設定の値を格納できる。従って、以下のコードを for ループの中に追加すれば、ファーストレイヤの印刷速度をマニュアルで指定できる。

```
1 if height == 0:
2     wall.print_speed = 5000
```

操作画面右側のパラメータツリーでの print\_speed の値が 10000 の時、上のコードを実行すれば、第 0 層目(ファーストレイヤ) では印刷則と 5000mm/min、他の層では、10000mm/min となり、印刷速度の個別制御が実現できる。印刷速度に限らず、射出係数、リトラクション、z hop、などグローバル設定で指定できる設定値は、全て個別に値をマニュアル調整することができる。

### 3.3 G-code

#### 3.3.1 G-code 出力方法

前節までの内容で、ノズルが移動する座標列の情報とその時の印刷設定の情報を内部に保持したパスオブジェクトを作成することができた。本節では、それらのパスから G-code に変換する方法について述べる。

G-coordinator(GUI アプリ) では、Fig1 の画面の右下の G-code Export ボタンを押すことで、G-code を出力することができる。なお、スタート、エンド G-code は、画面右上の machine settigns ボタンを押して出てくるポップアップウィンドウで設定することができる。

python ライブドリののみを使用する場合には、以下のコードを造形コードの最後に追加する。

```
1 gcode = gc.GCode(full_object)
2 gcode.start_gcode("path/to/
3     start_gcode.txt")
4 gcode.end_gcode("path/to/end_gcode
5     .txt")
6 gcode.save('your.gcode')
```

#### 3.3.2 G-code 出力の内部処理

座標列とパスに紐づけられた印刷設定の情報をもとに G-code を生成する手順について述べる。パスを構成する各セグメントに対して射出量を以下のように計算される [3] :

$$E_i = \frac{4whL_i}{\pi D^2} \cdot k_i \quad (1)$$

ここで、各文字の意味は以下の通りである。

- $E_i$ : i 番目のセグメントの射出量
- $D$ : フィラメントの直径
- $L_i$ : i 番目のセグメントの長さ
- $w$ : セグメントの横幅(ノズル直径)
- $h$ : パスの厚み(レイヤー厚さ)
- $k_i$ : i 番目のセグメントの射出係数

## 4 ユースケース

以下に, G-coordinator を用いて造形したユースケースを示す.

### 4.1 網目テクスチャ

京都の新工芸舎 [4] の Tilde[5] にインスピアイアを受けて作成した作成を Fig.3 に示す. ノズルをジグザグに蛇行させ, 一層ごとに位相を反転させることで, 積層痕を意匠化させた. スライスソフトでは, 直接ノズルパスを編集できないため, テクスチャを付与することが困難であった.

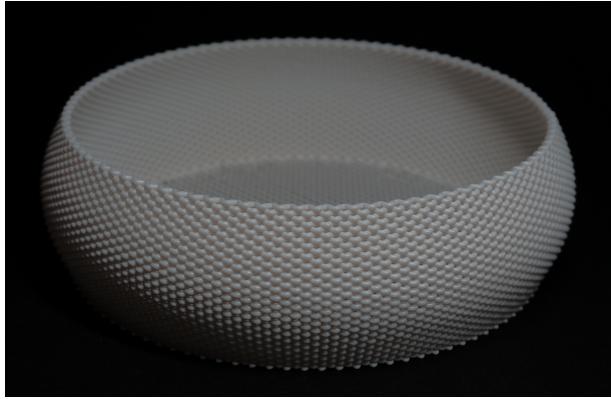


Figure 3: Woven textured tray

### 4.2 数理的な形状

G-coordinator ではコードと関数を用いてモデリングを行うため, 数理的な形状の実現が極めて容易にできる. Figure4 は, フラクタル形状を 3 次元に押し出したもの. 関数を再帰的に呼び出すことで, フラクタルのような複雑な形状を短いコードで実現することができる. 他にも, 一葉双曲面や, リサージュ曲線を造形するためのコードも公開している.

### 4.3 振動による図柄表示

Python の持つ豊富なライブラリを用いれば, あらゆるものを作成するため活用できる. Figure5 は, OpenCV のライブラリを用いて, モノクロ写真から, ある座標でのピクセルの濃さを抽出, その値を振動の振幅に重みとしてかけている. 白のプレートを印刷後, 黒の樹脂で振動パターンを印刷している. その結果, ミクロでは不規則な振動が, マクロでは写真に見える状況を実現している. 従来の CAD による造形



Figure 4: koch snowflake pot

では, 写真を参考にして造形を行うことはできるが, 写真データを直接扱うことはできなかった. python を用いて造形を行う恩恵により, 今までよりも多彩なデータソースを扱うことができる.

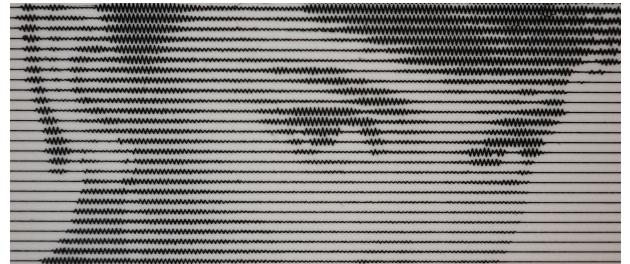


Figure 5: 画像の振動パターン化

### 4.4 インフィル

他の G-code Modeling と比較して, G-coordinator の大きな特徴のひとつとして, Fig.6 に示すようなインフィルを利用できることが挙げられる. インフィルを利用できるようになると, 実用性を意識したものを作成することができる. 現在は, line infil, gyroid infill の二種類を利用可能である.

以下のコードを for 文の中に記述することで, 最初の 20 レイヤーにインフィルを付与することができる.

```
1 if height<20:  
2     gyroid = gc.gyroid_infill(wall,  
3                                 infill_distance = 2)  
4     full_object.append(gyroid)
```

ここで, wall オブジェクトは外枠を表す Path クラスのオブジェクトである. gyroid\_infill 関数は

インフィルを計算する関数であり、外枠のパスを入力として、その内部を埋めるパスを返す関数である。ただし、インフィルは一本の曲線で表されるものではなく、複数のパスがまとまって、1レイヤー分のインフィルが構成されているため、厳密には関数からの返り値として返されるのは PathList クラスのオブジェクトである。しかし、PathList クラスのオブジェクトも Path クラスと同様に印刷設定の指定などができるため、ユーザは Path オブジェクトと同じように扱うことができ、そのクラスの違いを大きく意識する必要性は少ない。



Figure 6: 底部にインフィルを付与した小物トレー

## 5 将来展望

### 5.1 開発中のテスト機能

G-coordinator を用いて様々な造形を可能にするためにいくつかの機能を開発中である。何れの機能も既に現在の G-coordinator ver3.0.0 にはアルファ版の機能として実装されている。

#### 5.1.1 数式処理

4.2 節に示すように、G-coordinator の機能を利用して数式処理を行うことができる。現在は、さらにその点を生かすべく、様々な関数やクラスを開発中である。具体的な例として、陽関数のプロットやパスの生成は容易である一方で、陰関数のプロットは高い難易度を伴う。この課題に対処するため、numpy の meshgrid を活用し、陰関数によって表現された曲線や曲面をより容易に生成できる枠組みを開発中している。この取り組みにより、より広範で複雑な数学的表現を容易に扱えるようになり、研究や応用上での有用性が向上することが期待される。

#### 5.1.2 多軸制御

多軸 3D プリンタの制御においては、座標に加えてノズルの傾きなど、より複雑な要素が造形プロセスに関与するため、その難易度は著しく高まる。この課題に対処するため、より使いやすく効果的な多軸造形が可能となるようなフレームワークの開発を進めている。

具体的には、クオータニオン（四元数）を活用したノズルの姿勢制御の開発に焦点を当てている。現行の G-coordinator の多軸制御手法では、オイラー角や回転行列を使用して姿勢制御を行っている。それに対し、クオータニオンを導入することで、ジンバルロックなどの特異点をなくしつつ、より柔軟で安定した姿勢制御が可能となる。  
[6] さらに、計算量が削減され、処理時間が短縮されるという利点もある。

新たな手法による多軸制御 G-code の生成においても積極的に取り組んでおり、これによりユーザがより容易に多軸造形を行える環境を提供することを目指している。

#### 5.1.3 スライス機能

G-coordinator は基本的にすべてコードを用いて造形を行う。しかし、工業的な形状の造形をコードのみで行なうことは難しい。そこで、G-coordinator には stl データをスライスする関数も用意している。

```
1 wall = slice(mesh, height*0.2)
```

このように、スライス関数にスライス対象のメッシュとスライスする高さを入力することで、スライスした結果のパス（正確には PathList オブジェクト）が output される。純粋なスライスソフトとは異なり、スライスしたパスに対して、個別の印刷設定をマニュアルで付与することができる。しかし、現在実装しているスライスアルゴリズムではスライスの計算時間が比較的長いことや、トップ/ボトムの表面を手動で設定しなければならない等の問題もあり、まだ完全な実用段階ではない。将来的には、前節で述べた多軸制御と絡めて、多軸マシンのためのスライス機能もサポートする予定である。

## 5.2 ソフトウェアの開発方針

G-coordinator は、数学とコードにより、今まで出来なかった形状を実現する一方で、同時に、ユーザには多少の数学的知識と、プログラムに関するスキルを要求するものであり、参入のための障壁は比較的高いと言わざるをえない。そ

こで、現在想定している G-coordinator の将来像は大きく分けて二つある。

一つは、Processing などを代表とするビジュアルアートを実物化させるためのソフトウェアとしての活用である。Processing は、プログラムを書いて画面上に図形や形状を描画することができるソフトウェアであるが、G-coordinator はプログラムを書いて実際に触れるモノを作成できるソフトウェアである。これにより、アーティストやデザイナーは数学的なアルゴリズムを用いて、従来不可能だった美しい形状を 3D プリンティングで具現化することが可能になる。

もう一つは、G-coordinator が数理的な形状得意な性質を生かし、物理シミュレーションや計算力学を応用して形状を実物化させるソフトウェアとしての利用である。具体的には、トポロジー最適化のアルゴリズムを G-coordinator の内部で実装し、最適化計算から G-code 生成までをシームレスに行うことができるソフトウェアの実現を目指している。これにより、エンジニアや科学者は複雑な物理的な現象をシミュレートし、それを具現化するのに G-coordinator を活用できる。

プログラムを使った 3D 形状の生成と制御が主眼となる点で、どちらの展望も共通している。

## 6 結論

MEX 型 3D プリンタユーザにとって、CAD などの造形手段と、印刷のための調整を行うスライサを同時に置き換える新たな手段としての G-coordinator のポテンシャルについて述べた。スライスソフトに比べて、全部を機械任せにできない欠点はあるものの、裏を返せば、3D プリンタの挙動の細部までマニュアル操作可能という大きな利点がある。あらゆる人々に最も自由度の高い G-code Modeling を提供する G-coordinator は、数学とプログラムの力を駆使して新しい形状を実現するための強力なツールとして、今後の 3D プリント分野でさらなる進化が期待される。

## 7 謝辞

田中浩也先生（慶應義塾大学）には多くの助言をいただきとともに、本論文の掲載料を提供していただいた。また、G-coordinator の多軸制御の開発において、反保紀昭氏に助言とコード提供をいただいた。両名の協力がなければ、本研究を今日の形で公表することはできなかった。この場を借りて感謝を申し上げる。

## References

- [1] 知念司泰. ポリゴンモデリング技法をツールパス表現に直接変換するソフトウェアの開発, 2020.
- [2] Andrew Gleadall. Fullcontrol gcode designer: Open-source software for unconstrained design in additive manufacturing. *Additive Manufacturing*, 46:102109, October 2021. Research Paper.
- [3] Behzad Akhoundi, Mohsen Nabipour, Farid Hajami, Saeed Shirazian Band, and Amir Mosavi. Calculating filament feed in the fused deposition modeling process to correctly print continuous fiber composites in curved paths. *Materials (Basel, Switzerland)*, 13(20):4480, 2020.
- [4] 新工芸舎. <https://www.shinkogeisha.com/>. Accessed: January 4, 2024.
- [5] Tilde printed. <https://www.tilde-printed.com/>. Accessed: January 4, 2024.
- [6] 山口 功, 木田 隆, 岡本 修, 狼 嘉彰, YAMAGUCHI Isao, KIDA Takashi, OKAMOTO Osamu, and OOKAMI Yoshiaki. クオータニオンとオイラー角によるキネマティックス表現の比較について. *航空宇宙技術研究所資料 = Technical Memorandum of National Aerospace Laboratory*, vol.636:15, 06 1991.

## 8 付録

以下に円柱を造形するためのサンプルコードを示す。ただし、以下のコードは G-coordinator の ver3 以降で動作することに注意されたい。

```
1 import numpy as np
2 import gcoordinator as gc
3
4 LAYER = 100
5 full_object=[]
6 for height in range(LAYER):
7     arg = np.linspace(0, 2*np.pi,
8                       100)
9     x = 10 * np.cos(arg)
10    y = 10 * np.sin(arg)
11    z = np.full_like(arg, (height
12                      +1) * 0.2)
13    wall = gc.Path(x, y, z)
14    full_object.append(wall)
15 gc.gui_export(full_object)
```