

# G-coordinator

G-code 生成の新たな手法とその可能性

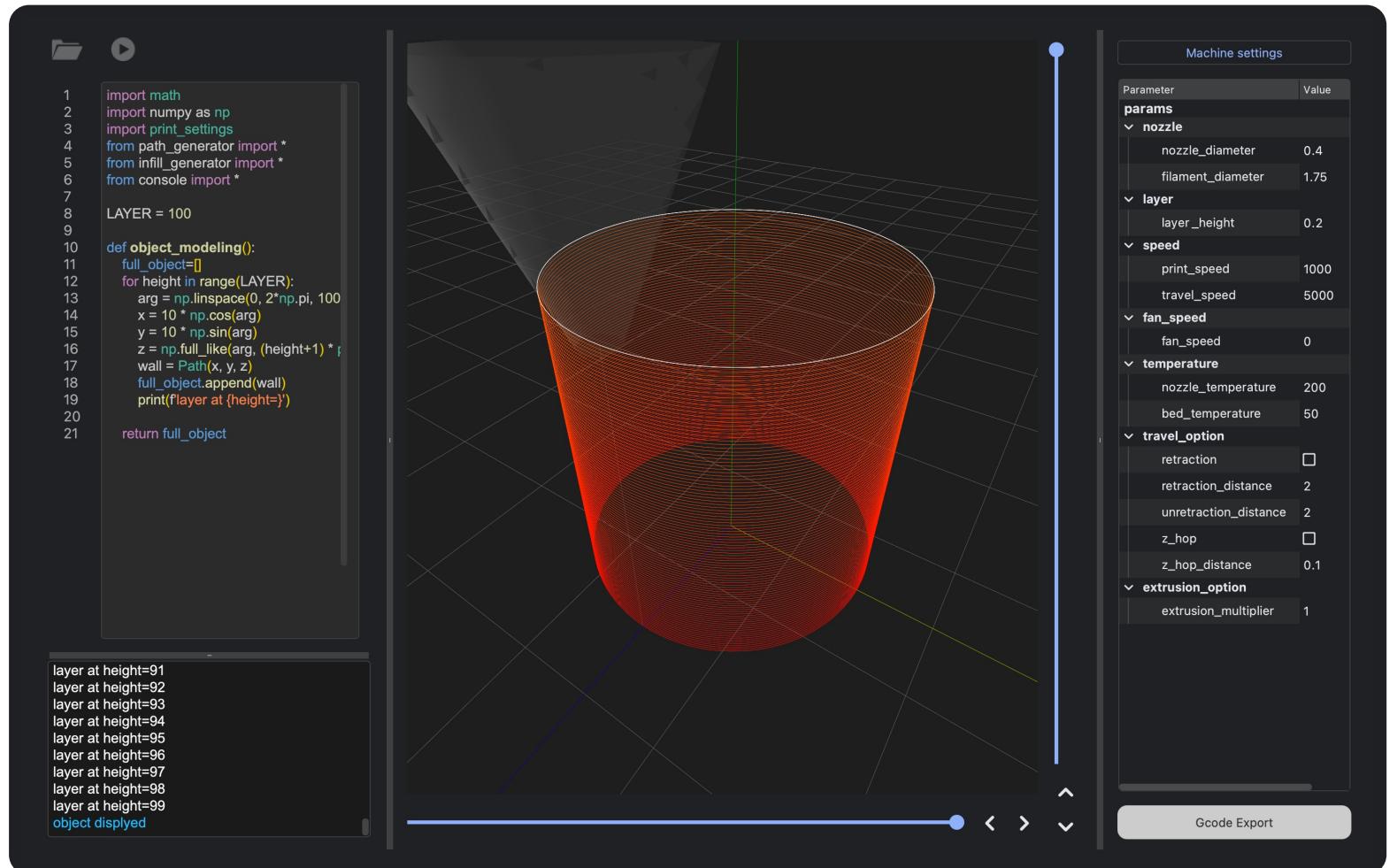
大阪府立大学 工学域 機械力学研究室  
B4 谷口朝洋

# What is G-coordinator?

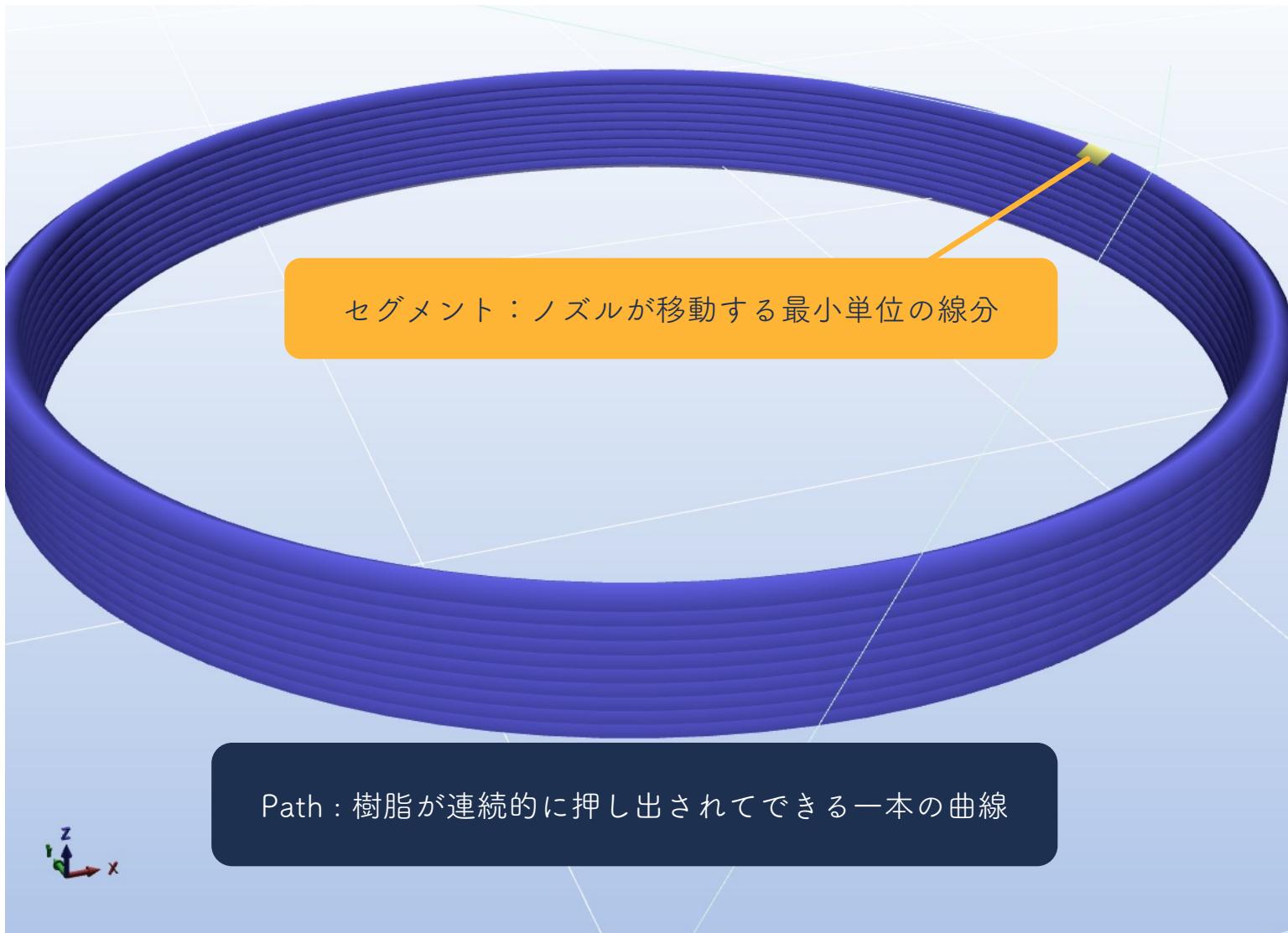
G-coordinatorは、3Dプリンタ制御コード(G-code)を出力するためのソフトウェア、ライブラリ。

ソフトウェア内で、pythonをもちいた造形と印刷条件の設定の2プロセスを同時に処理できる。

スライスソフトとは根本的に異なる手法によるパス(経路)生成により、従来では困難だった形状や、より詳細な印刷設定の指定が可能。



# Principle of G-code



G-codeは、ノズルが移動していく  
3次元の座標列 + F値+E値.  
押し出し量(E)や速度(F)は、  
ソフトで自動決定できる。  
考慮すべきはノズルの軌跡.

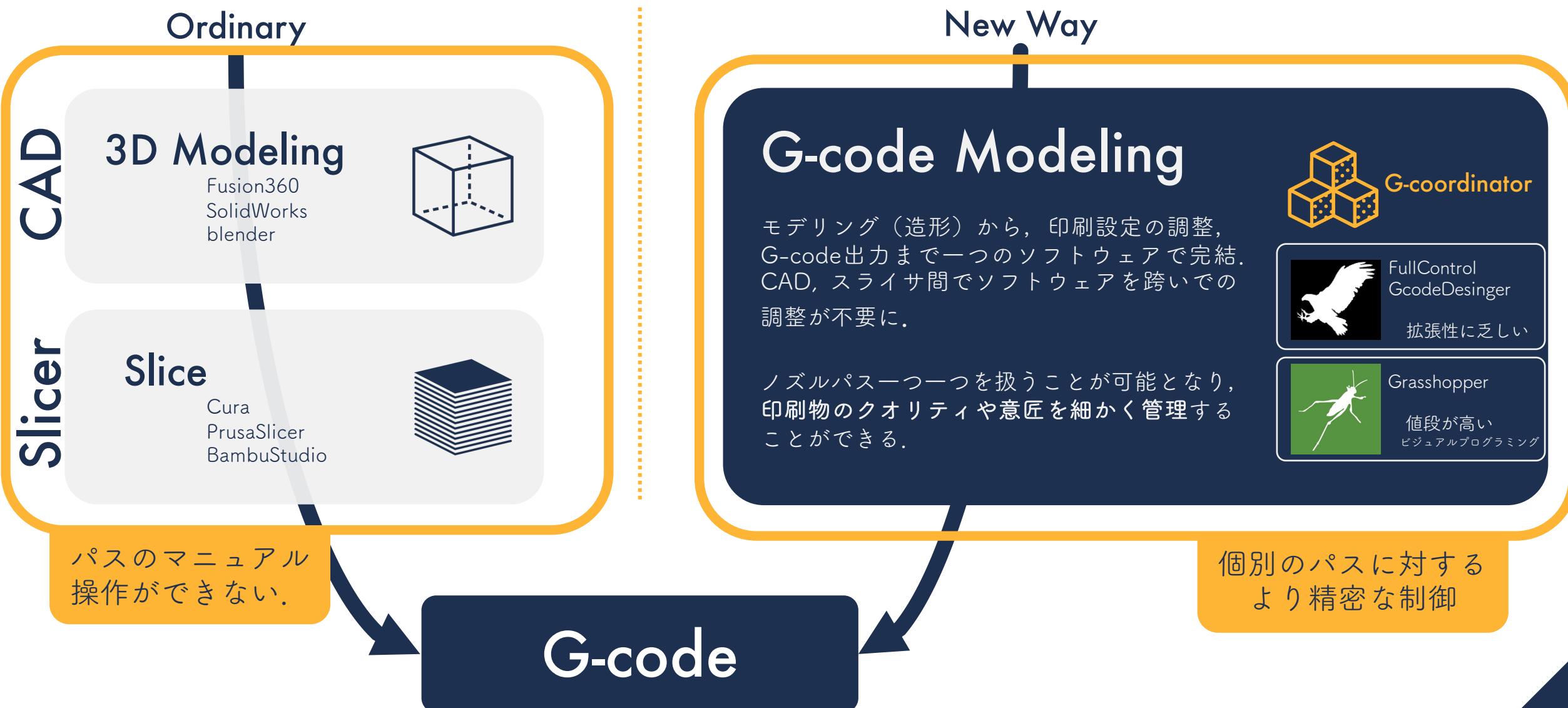
1018 G1 F800 X111.78509 Y97.65408 Z1.80000 E0.00589
1019 G1 F800 X112.23734 Y98.09921 Z1.80000 E0.00589
1020 G1 F800 X112.66044 Y98.57212 Z1.80000 E0.00589
1021 G1 F800 X113.05270 Y99.07092 Z1.80000 E0.00589
1022 G1 F800 X113.41254 Y99.59359 Z1.80000 E0.00589
1023 G1 F800 X113.73849 Y100.13803 Z1.80000 E0.00589
1024 G1 F800 X114.02927 Y100.70205 Z1.80000 E0.00589
1025 G1 F800 X114.28368 Y101.28338 Z1.80000 E0.00589
1026 G1 F800 X114.50071 Y101.87967 Z1.80000 E0.00589
1027 G1 F800 X114.67949 Y102.48852 Z1.80000 E0.00589
1028 G1 F800 X114.81929 Y103.10749 Z1.80000 E0.00589
1029 G1 F800 X114.91955 Y103.73408 Z1.80000 E0.00589
1030 G1 F800 X114.97987 Y104.36576 Z1.80000 E0.00589

1043 G1 F800 X114.81929 Y106.89251 Z2.00000 E0.00589
1044 G1 F800 X114.67949 Y107.51148 Z2.00000 E0.00589
1045 G1 F800 X114.50071 Y108.12033 Z2.00000 E0.00589
1046 G1 F800 X114.28368 Y108.71662 Z2.00000 E0.00589
1047 G1 F800 X114.02927 Y109.29795 Z2.00000 E0.00589
1048 G1 F800 X113.73849 Y109.86197 Z2.00000 E0.00589
1049 G1 F800 X113.41254 Y110.40641 Z2.00000 E0.00589
1050 G1 F800 X113.05270 Y110.92908 Z2.00000 E0.00589
1051 G1 F800 X112.66044 Y111.42788 Z2.00000 E0.00589
1052 G1 F800 X112.23734 Y111.90079 Z2.00000 E0.00589
1053 G1 F800 X111.78509 Y112.34592 Z2.00000 E0.00589
1054 G1 F800 X111.20552 Y112.76146 Z2.00000 E0.00589
1055 G1 F800 X110.80057 Y113.14576 Z2.00000 E0.00589

Show complete code First Layer: 0  
Show single layer Last Layer: 0  
Show layer range

C13 R1043/1181 Layer: 10 Top: 0 Fil:23.3 Printing time:0:01

# Back Ground





vs



# Difference

from other G-code Modeling

## Open Source

### コードの公開

G-coordinatorは、ソフトウェア、ライブラリ、造形のサンプルコードなど、あらゆるコードを公開。無料でソフトウェアをインストールできる。また、個人の要求に合わせて自由に改造可能

## Extension

### 自由な拡張性

Pythonで造形を行うことで、あらゆるライブラリやデータソースを造形のために活用できる。dxfなどの図面データ、テキストデータ、csvファイル、画像など、従来では、CADで直接扱えなかったデータも利用可能。

## Multi Axis

### 多軸印刷

多軸の3Dプリンタ（5軸、6軸）でも利用可能。x, y, zの座標値だけでなく、ノズルの回転や傾きを制御できる。この機能は、オープンソースの恩恵として、コンピュータによって実現。

## Infill

### インフィルの生成

G-coordinatorでは、インフィルを生成することが可能。現在は、line infillと、gyroid infillを生成することができる。

# How to Use

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. 印刷条件を設定
4. G-codeの出力

グローバルな設定をパラメータツリーで、パスごとの詳細設定はエディタからコードで行う。

The screenshot shows a software interface for 3D printing. On the left, a code editor displays Python code for generating a 3D model of a cup. The code uses libraries like math, numpy, and various path and infill generators. It defines a function `object_modeling` that creates a series of layers (100) and adds infill to them. A preview window in the center shows a wireframe of the cup's interior with a hexagonal infill pattern. A coordinate system (x, y, z) is visible. On the right, a panel titled "Machine settings" lists various parameters with their values, such as nozzle diameter (0.4), filament diameter (1.75), print speed (15000), and travel speed (25000). At the bottom right, there is a "Gcode Export" button.

```
1 import math
2 import numpy as np
3 import print_settings
4 from path_generator import *
5 from infill_generator import *
6 from console import *
7
8 LAYER = 100
9
10 def object_modeling():
11     full_object=[]
12     for height in range(LAYER):
13         arg = np.linspace(0, 2*np.pi, 99)
14         x = 10 * np.cos(arg)
15         y = 10 * np.sin(arg)
16         z = np.full_like(arg, (height+1) * 0.2)
17         wall = Path(x, y, z)
18         outer_wall = Transform.offset(wall, 0.4)
19         full_object.append(wall)
20         full_object.append(outer_wall)
21
22         if height<10:
23             infill = gyroid_infill(wall, density = 0.9)
24             full_object.append(infill)
25             print(f'layer at {height=}')
26
27     return full_object
```

layer at height=91  
layer at height=92  
layer at height=93  
layer at height=94  
layer at height=95  
layer at height=96  
layer at height=97  
layer at height=98  
layer at height=99  
object displayed

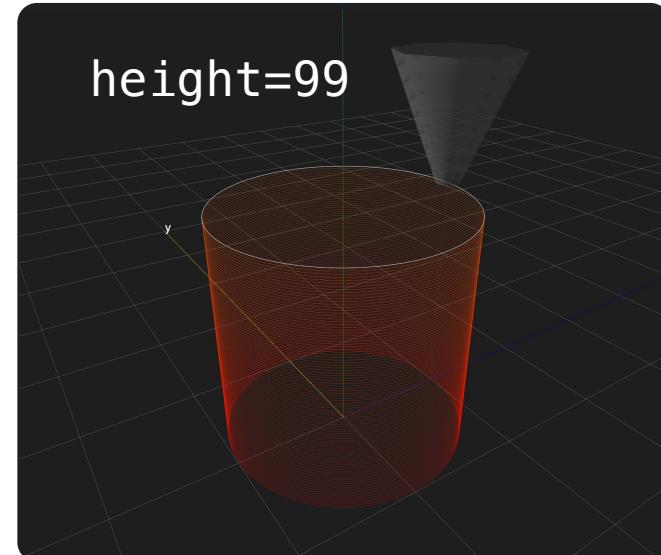
Machine settings

Parameter	Value
params	
nozzle	
nozzle_diameter	0.4
filament_diameter	1.75
layer	
layer_height	0.2
speed	
print_speed	15000
travel_speed	25000
fan_speed	
fan_speed	255
temperature	
nozzle_temperature	200
bed_temperature	35
travel_option	
retraction	□
retraction_distance	2
unretraction_distance	2
z_hop	□
z_hop_distance	0.1
extrusion_option	
extrusion_multiplier	1.4

Gcode Export

# How to Code

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



# Print Settings

Pathの基本設定を左のパラメータツリーで行い、個別に調整したい設定はエディタで行う。  
extrusion\_multiplier(射出係数)と、

print\_speed(印刷速度)  
は、Pathのセグメントと同じ要素数の数列で指定すれば、各セグメントごとに印刷設定を調整可能。

```
18  
19     wall = Path(x, y, z)  
20     outer_wall = Transform.offset(wall, 0.4)  
21  
22     outer_wall.print_speed = 5000  
23     if height == 0:  
24         wall.extrusion_multiplier = 1.2  
25
```

ファーストレイヤは、ベッド定着のために射出量を1.2倍に。

Local Settings

外壁は、外観を綺麗に印刷するために、速度を落として印刷。

Global Settings

Parameter	Value
params	
nozzle	
nozzle_diameter	0.4
filament_diameter	1.75
layer	
layer_height	0.2
speed	
print_speed	15000
travel_speed	25000
fan_speed	
fan_speed	255
temperature	
nozzle_temperature	200
bed_temperature	35
travel_option	
retraction	<input type="checkbox"/>
retraction_distance	2
unretraction_distance	2
z_hop	<input type="checkbox"/>
z_hop_distance	0.1
extrusion_option	
extrusion_multiplier	1



# Works

G-coordinatorを用いてG-codeを作成し、印刷した作例も含めて公開している。

作例の写真はG-coordinatorリポジトリのディレクトリ、造形のためのコードはexamplesディレクトリに公開している。

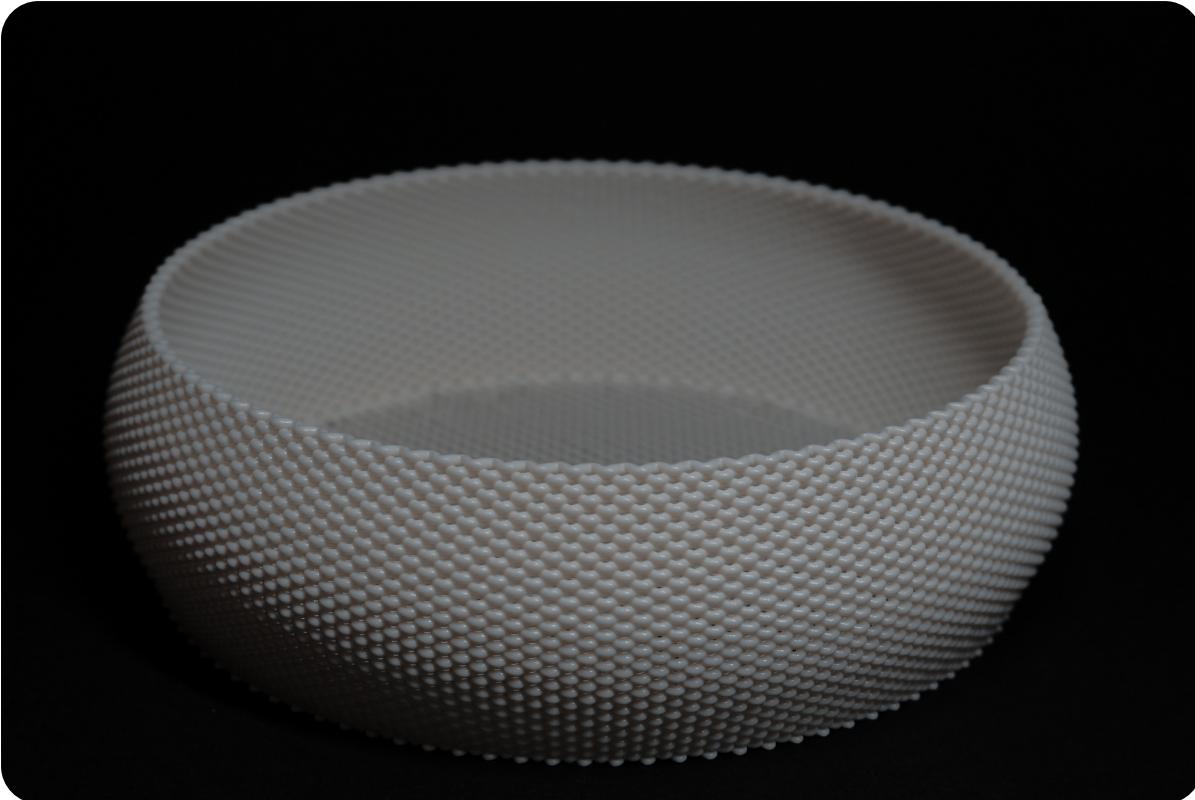
左の写真はランプシェードの作例。

<https://github.com/tomohiron907/G-coordinator/tree/main/img/works>  
<https://github.com/tomohiron907/G-coordinator/tree/main/example>

# Wave Texture

太ノズルで、ノズルを蛇行することで、積層痕を意匠化

stlモデルのスライスでは、ノズルパスを精密に制御することが難しく、網目のテクスチャを付与することができなかった。



rad = base\_rad

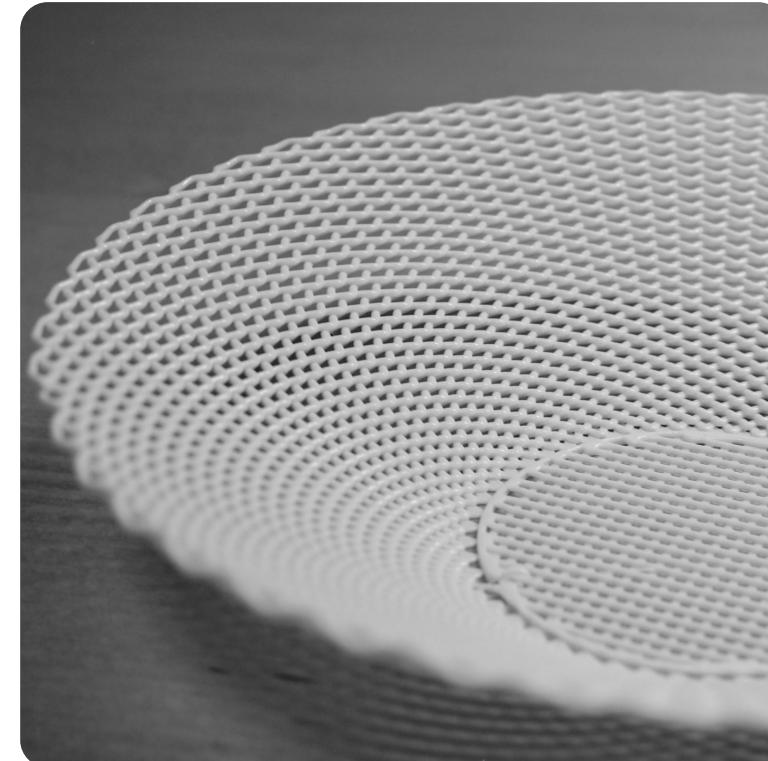
基本半径

rad += 7\*np.sin((z/LAYER)\*np.pi)

高さに応じた丸みを追加

rad += amp\*np.sin(arg\*100.5+np.pi\*height)

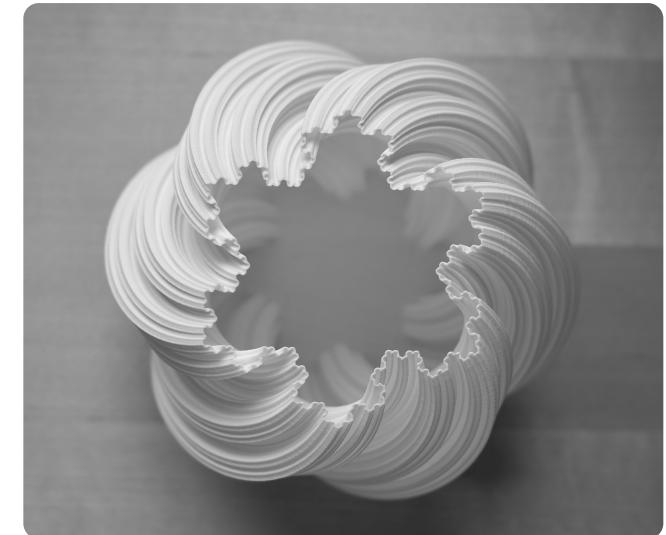
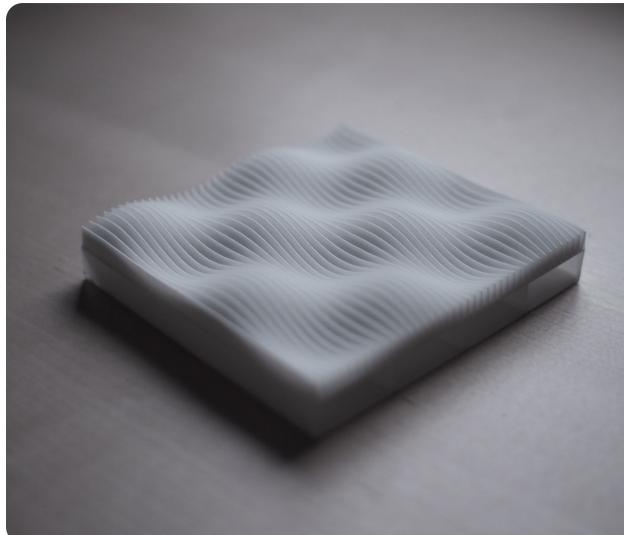
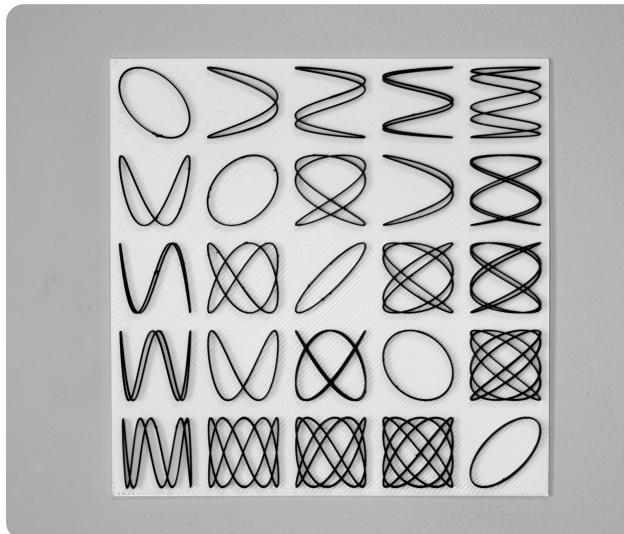
一周の円を描く間に100.5回の振動を追加



# Mathematical Design

G-coordinatorは、pythonで造形を行う性質上、数学的、数理的な形状の実現が得意。

この他にも、一葉双曲面や、立体包絡線、ヒルベルト曲線などを描画するサンプルも公開している。



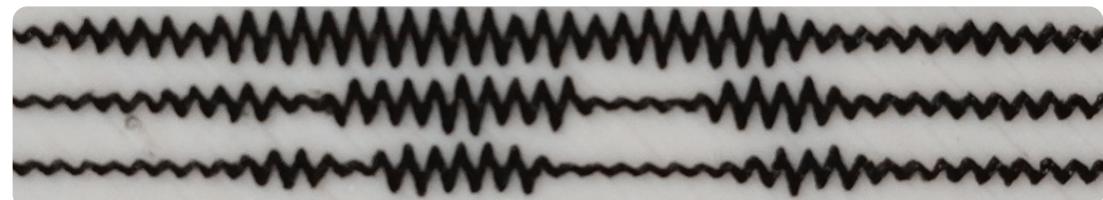


# Picture Into G-code

Pythonを用いることで、様々なデータソースを造形に活用できる。

OpenCVのライブラリを用いて、モノクロ写真をx, yの2つの引数があり、0から1のピクセルの濃さを返す関数ととらえる。そのピクセルの濃さを単純なsin波に重みとして掛け合わせている。

白のプレートを印刷後に、黒の樹脂で振動パターンを印刷することで、不規則な振動がマクロでは写真として見えるようになる。



# Infill

G-coordinatorでは、インフィルの作成が非常に簡単。

閉曲線のPathを関数に渡すと  
その中を埋めるインフィルPath  
が返ってくる。

インフィルが活用できること、実際に使うことを想定した実用品を作成することが可能になり、  
造形の幅が大きく広がる。



Tray with gyroid infill



Clock with line infill

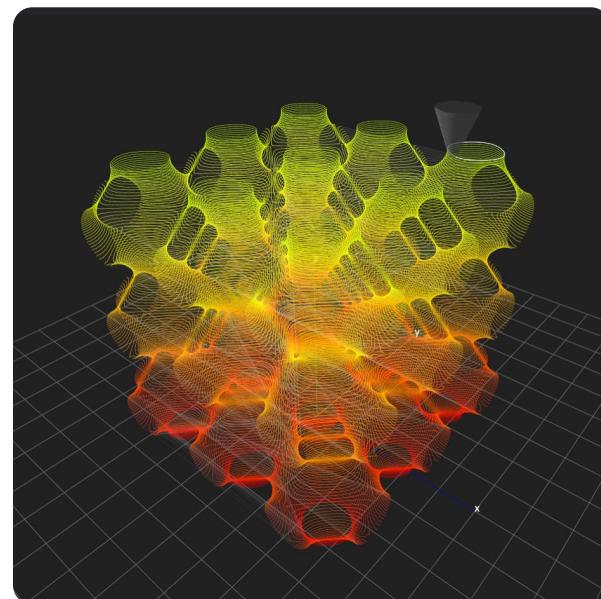
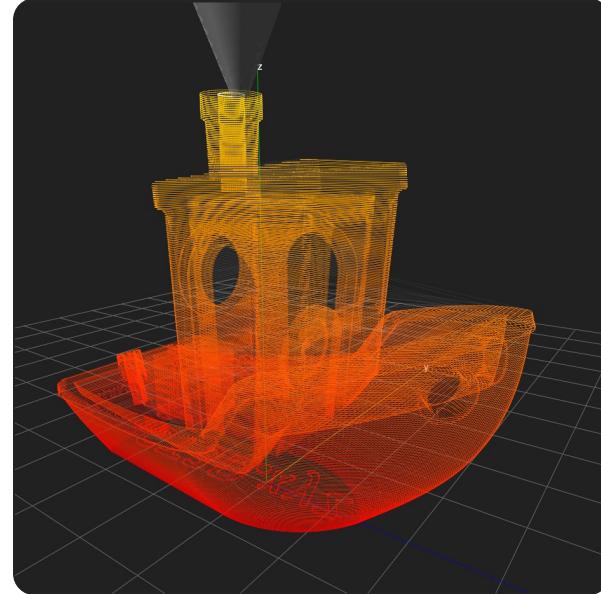
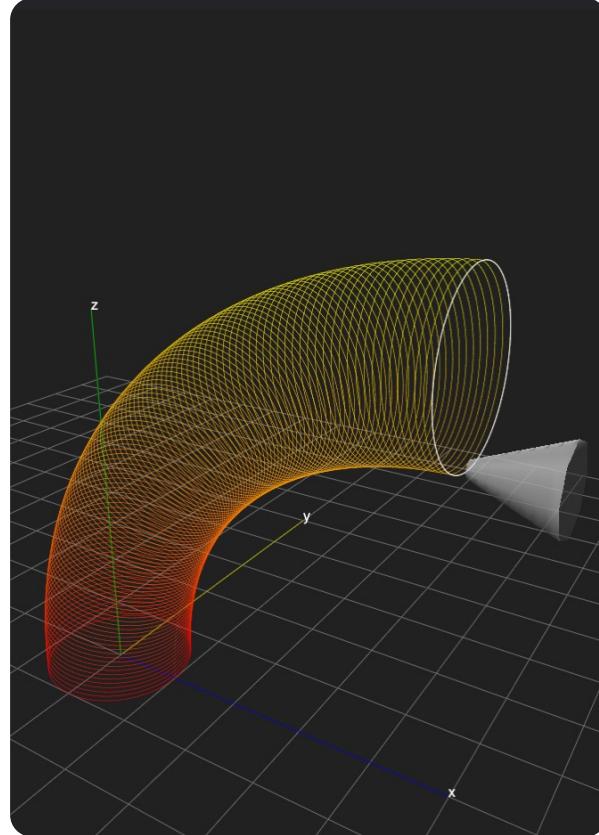
```
wall = {x⁴ + y⁴ = 30⁴}の軌跡}  
if height<10:  
    infill = gyroid_infill(wall)  
    full_object.append(infill)
```

## 多軸印刷

多軸制御は3軸プリンタに対して難易度が高い。

より簡単にその制御を可能にするための枠組みを開発中。

また、stlファイルの斜めスライスも計画中。



## stlスライス

複雑な形状などを実現するにはstlファイルのスライスがやはり便利。より高速なスライスとPath生成アルゴリズムを開発中。

## 数式処理

数式や関数をより簡単にPathに変換して印刷するためのクラスを開発中。

Alpha Version  
Features

```
1 import gcoordinator as gc
2 import numpy as np
3
4
5 full_object = []
6 for layer in range(100):
7     arg = np.linspace(0, 2*np.pi, 100)
8     x = 10 * np.cos(arg)
9     y = 10 * np.sin(arg)
10    z = np.full_like(x, layer * 0.1)
11    wall = gc.Path(x, y, z)
12
13    full_object.append(wall)
14    infill = gc.gyroid_infill(wall, infill_distance=2)
15    full_object.append(infill)
16
17 gc.show(full_object)
18
19 gcode = gc.GCode(full_object)
20 gcode.start_gcode("start_gcode.txt")
21 gcode.end_gcode("end_gcode.txt")
22 gcode.save('/Users/taniguchitomohiro/Documents/test.gcode')
```

# gcoordinator library

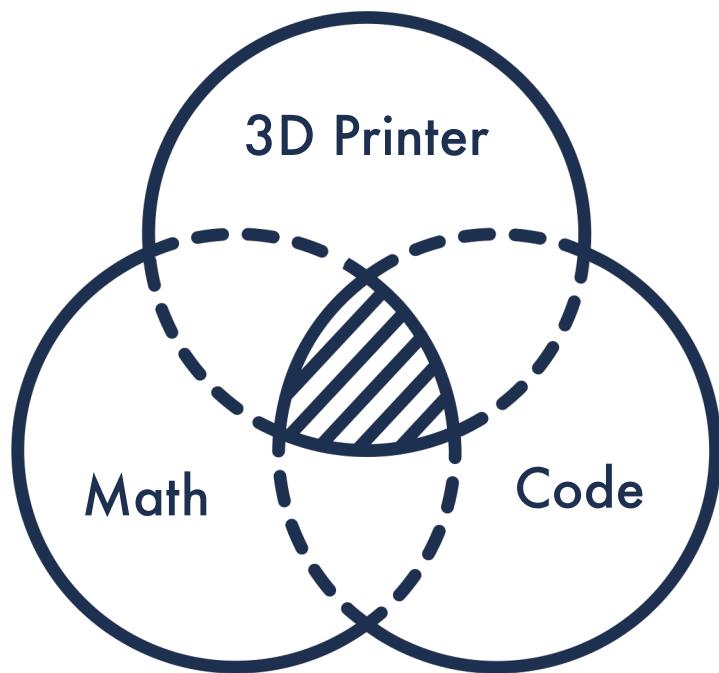
GUIアプリのG-coordinatorから、G-code生成エンジンのみを切り出したライブラリ。Python環境があれば、アプリをインストールしなくても、G-code生成が可能。

Matplotlibライブラリ3DP版。

G-coordinatorのGUIアプリのver3系列では、内部の生成ロジックをライブラリに完全移行

```
$ pip install gcoordinator
```

# Future Outlook



G-coordinatorは、数学とコードに関する多少の知識を求めてしまう。  
しかし、だからこそ作ることが可能な造形も多くある。

G-coordinatorの進む先は大きく二つ。

**Generative Art**

**Simulation into Reality**

G-coordinatorはビジュアルアートを実物化させるソフトとして利用できる。

Processingの3次元版のイメージ

物理シミュレーションや計算力学の結果を実物化させるソフトとしての利用。

例えば、G-coordinatorでトポロジー最適化→そのまま印刷。

# #Gcoordinator



# Thank You

本発表の資料



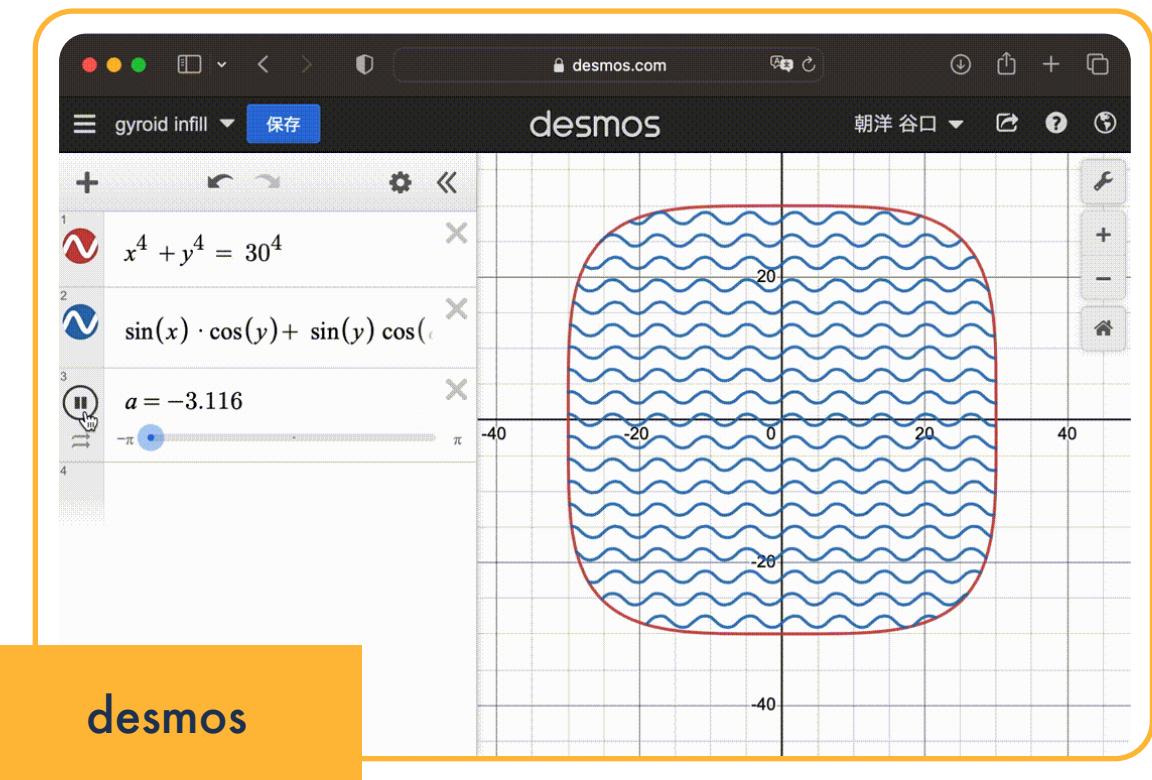
G-coordinator  
GitHubページ



G-coordinator  
インストール



# G-coordinator Modeling Tips



以下は、Matplotlibを使用してヒルベルト曲線を描画するコードです。

python

```
import matplotlib.pyplot as plt

def hilbert_curve(x, y, xi, xj, yi, yj, depth):
    if depth <= 0:
        # ヒルベルト曲線の点を返す
        return [x + (xi + yi) / 2], [y + (xj + yj) / 2]

    # 四分割するためのパラメータ
    xn = x + (xi + yi) / 2
    yn = y + (xj + yj) / 2

    # 四隅の座標を再帰的に計算する
    x1, y1 = hilbert_curve(x, y, yi / 2, xj / 2, xi / 2, xj / 2, depth - 1)
    x2, y2 = hilbert_curve(x + xi / 2, y + xj / 2, xi / 2, xj / 2, yi / 2, yj / 2, depth - 1)
    x3, y3 = hilbert_curve(x + xi / 2 + yi / 2, y + xj / 2 + yj / 2, xi / 2, xj / 2, -yi / 2, -yj / 2, depth - 1)
    x4, y4 = hilbert_curve(x + xi / 2 + yi, y + xj / 2 + yj, -yi / 2, -yj / 2, depth - 1)

    # 点の順番を修正して返す
    return x1 + x2 + x3 + x4, y1 + y2 + y3 + y4

def draw_hilbert_curve(length, depth):
    # 初期位置と方向を設定
    x = 0
    y = 0
    xi = length
    xj = 0
    yi = 0
    yj = length

    # Matplotlibのplt.plot()関数の引数はx座標列,
    # y座標列である。z座標列をレイヤー高さに設
    # 定し, 生成されたコードのplot()の部分を
    # Path()に置き換えればいい。
    # ヒルベルト曲線を計算する
    x, y = hilbert_curve(x, y, xi, xj, yi, yj, depth)
```