Python ライブラリを使用した手続き的な 3D 印刷法

金田 泰[1,a)]

概要：機械加工や 3D 印刷をコンピュータを使用しておこなうとき，工作機械や 3D プリンタを手続き的に制御するためのプログラムが必要になる．この目的のためにひろく使用されているのが G コードである．G コードはもともと切削加工の制御のために開発され，当初は設計者がそれによるプログラムを記述していたが，現在は設計者は CAD によって宣言的なモデルを記述し，それをコンピュータが G コードに変換する．しかし，3D 印刷のような付加加工のプロセスは切削加工より直観的なので，設計者が抽象化された手続き的記述をすることが場合によっては利点があるとかんがえられる．そこでこの論文では手続き的な 3D 設計用ライブラリを使用した Python プログラムによって G コードのプログラムを生成し 3D プリンタで印刷する方法とその使用例を示す．この方法では印刷可能な形状は限定されるが，層をなくして層のつぎめもなくし，従来の方法においてはオーバハングがあるときに必要だった支持材料 (サポート) もなくして，シームレスでより美的な印刷を実現した．

キーワード：3D 印刷, 付加加工, 宣言的モデル, 宣言的記述, 手続き的記述, 3D プリンタ, G コード

Method for Procedural 3D-printing Using a Python Library

Yasusi Kanada[1,a)]

Abstract: When manufacturing or 3D-printing a product using a computer, a program that procedurally controls manufacturing machines or 3D-printers is required. G-code is widely used for this purpose. G-code was developed for controlling of subtractive manufacturing, and a designer historically wrote programs in G-code; however, in recent development environments, the designer describes a declarative model by using CAD, and the computer converts it to a G-code program. However, because the process of additive manufacturing, such as 3D printing, is more intuitive than subtractive manufacturing, it sometimes seems to be advantageous to describe an abstract procedural program by the designer for this purpose. This paper, thus, proposes a method for generating G-code by describing a Python program using a library for procedural 3D-design and for printing by a 3D printer, and shows use cases. Although shapes printable by this method are restricted, this method can eliminate layers and layer seams and eliminate support material, which is necessary for conventional methods when an overhang exists, and it enables seamless and artistic printing.

Keywords: 3D printing, Additive manufacturing, Declarative model, Declarative description, Procedural description, 3D printer, G-code

1. 緒言

3D 印刷をふくむ機械加工をコンピュータによっておこなうとき，加工の手続きを G コードという言語によって記述する．3D 印刷 (3D printing) は付加加工 (additive manufacturing, AM) という機械加工の一種とかんがえられていて，機械加工 (manufacturing) を制御するためのプログラムが必要になる．この目的のために G コード [17] というアセンブラ風の言語が使用される．G コードはもともと工作機械の刃の動作を記述するので手続き的である．

かつては G コードや 2 章で紹介する APT のような手続き的な言語による制御プログラムを機械加工の設計者が記述していたが，現在では設計時には computer-aided design (CAD) によって宣言的なモデルが記述される．G コードはもともと切削加工 (現在の用語では subtractive manufacturing) の制御のために開発され，当初は設計者

[1] Dasyn.com Dasyn.com, Nakano, Tokyo 164-0013, Japan
[a)] yasusi@kanadas.com

図 1 手続き的に 3D 印刷した中空球の例 (表面と内部)

がそのプログラムを記述していた．しかし，現在では設計者は CAD によって宣言的なモデルを記述し，それをコンピュータが手続き的な G コードに変換する．

切削加工においては手続き的な設計が宣言的な設計によってすっかりとってかわられたが，付加加工においては，以下に記述するような理由によって，手続き的な設計法に利点があるとかんがえられる．切削加工の手続きは制約が多くて複雑なため手続き的な記述が適さず，G コードのような言語は低水準で抽象化機能がなかったゆえにプログラミングが困難だった．しかし，3D 印刷のような付加加工は切削加工より直観的なので，宣言的な方法ではうまく印刷できないが設計者が抽象化された手続き的記述をすることによってうまく印刷できる場合がある．たとえば，図 1 のような中空の球はつぎの理由によって，層ごとに印刷する通常の 3D 印刷でつくるのは困難である．すなわち，球の最上部は急角度のオーバーハング (張り出し) が必要なため球の内部にサポートが必要になり，中空にすることは困難である．しかし，直観的な手続き的記述をすることによって図 1 のようにきれいな中空球が印刷できる．これは，ソフトウェア開発において宣言的な言語を使用するとかゆいところに手がとどきにくくて手続き的な言語を使用するほうがうまくいくのと同様である．ソフトウェア開発においては機械加工とはちがって現在でも手続き的な方法が主流である．

そこでこの論文では 3D 印刷用ライブラリを使用し手続き的に抽象化された Python プログラムによって G コードのプログラムを生成し 3D プリンタで印刷する方法とその使用例を示す．この論文で提案する方法とくにそれをプログラミングという視点から記述すること，およびそれを手続き的な機械加工の歴史や従来の方法のなかに位置づけることがこの論文のおもな目的である．以下，2 章では手続き的な切削加工の歴史，3D 印刷法とそのプログラミングについてのべる．3 章では著者が開発した Python ライブラリを使用した手続き的な 3D 印刷法についてのべ，4 章においてそれにもとづく 3D 印刷の実践についてのべ，5 章で関連研究にふれて 6 章でまとめる．

2. 従来の切削加工と付加加工

手続き的な 3D 印刷の実践について記述するまえに機械加工における手続き的な記述の価値をみなおすため，この章ではまず手続き的な切削加工の歴史をみなおし，つぎに現在の付加加工とくに 3D 印刷法と 3D プリンタのプログラミングについて記述する．

2.1 手続き的な切削加工の歴史

コンピュータ数値制御 (CNC) による切削加工の技術とそのための言語 APT はすでに 1940 〜 50 年代に開発されている．数値制御 (NC) の技術は 1942 年に John T. Parsons によって発明されたが，CNC の技術はそれにもとづいて 1950 年代にマサチューセッツ工科大学 (MIT) において開発された [24]．この制御のため，MIT においては APT というプログラミング言語 [1][20][7] が開発され，切削加工のプログラムを記述するのに使用された．APT は基本的にはアセンブラ風の言語であり手続き的だった．Parsons は NC プログラムの記録にパンチカードを使用したが，MIT で紙テープが使用され，各種のサブルーティンが作成されるとともに，APT にはマクロや入れ子定義 (nested definitions) のような原初的な手続き抽象化機構がとりいれられた [20]．1970 年代には APT とデータ抽象化やオブジェクト指向との関係も議論された [20]．

しかし，その後，CAD 技術が開発され，APT のような手続き的な方法によって設計者が指示することはほとんどなくなった． 設計者は CAD によって宣言的なモデルを記述し，それをコンピュータが手続き的なプログラムに変換するようになった．そうなったのは，切削加工には手続き的記述が適さない複雑さがあるうえ，G コードや APT は低水準で抽象化機能がよわいため (APT に関してはさまざまな改良がくわえられたが，互換性の問題もあるためそれには限界があり) 設計者がプログラムするのは困難だからだろう．

2.2 従来の 3D 印刷と G コード

3D プリンタを使用して 3 次元のオブジェクトを造形するとき，通常は 3D CAD で設計したモデルをスライサとよばれるソフトウェアで水平にスライスして，その結果をプリンタにおくって印刷する．CAD においては通常は GUI によってモデルを設計するが，モデルそのものは宣言的である．OpenSCAD [23] という設計ソフトウェアではプログラミング言語によってモデルを記述するが，その記述は宣言的である．CAD が出力するファイル形式はさまざまだが，スライサにおくるには STL (Standard Triangulation Language または Stereo-Lithography)[25] という宣言的な形式のファイルが使用される．STL はモデルの表面形状を 3 角形の集合によって近似する (内部は表現できない)．

3D プリンタにはさまざまな種類があるが，安価なタイプは FDM (Fused Deposition Modeling，熱溶解積層) 型とよばれる，とかしたフィラメント (プラスティック) をノズルの先端から射出してかためるタイプである (図 2)．

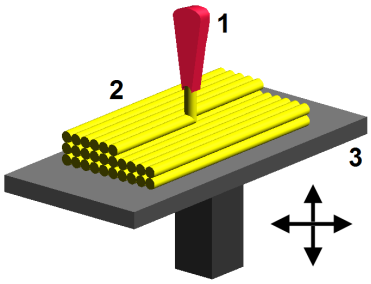


図 2 FDM 型 3D プリンタの原理 (FDM by “Zureks” by Zureks - Wikimedia Commons)

3D プリンタは通常は層ごとに印刷するが，3D プリンタのための言語である G コードは層という概念に制約されてはいない．スライスした結果は通常 G コードによって表現され，それによって印刷ヘッドの動作やプラスティックの射出速度などが指定される．3D プリンタでは通常，水平にスライスされた層ごとに印刷するので，層間の移動のとき以外はプリント・ヘッドが垂直方向にうごくことはない．しかし，G コードは層にしばられていないので，実はヘッドをもっと自由にうごかすことができる．

G コードによるコマンドの例として 2 つをあげておく．G0 というコマンドは単純なツールの移動を指令する．たとえば “G0 X1 Y2 Z3 F3600” というコマンドは分速 3600 mm で座標 (1, 2, 3) に移動することを指令する．また，G1 というコマンドは切削型の工作機械においては加工しながらの移動を意味し，付加型 (additive) の工作機械である 3D プリンタにおいては印刷しながらの移動を指令する．たとえば “G1 X1 Y2 Z3 F3600 E100” というコマンドを実行すると，E100 によって指定される量のフィラメントを射出しながら移動する．(フィラメントの量は指定により相対値または絶対値で指定される.) G0，G1 ともに移動方向に制約はない．

3. 手続き的な 3D 印刷法の開発

Python をベース言語とする手続き的な 3D 印刷の方法を開発したので，この章ではその言語とそれを使用した 3D 印刷の方法について記述する．

3.1 Python ベースの記述法

切削加工とはちがって 3D 印刷のような付加加工においてはより直観的に機械加工ができるので，切削加工の場合とちがって，1 章で記述したように設計者が手続き的に記述するのが有用だとかんがえられる．そのため，3D 印刷を手続きに記述するための Python のライブラリ (API, Application Programming Interface) を開発した．プログラミング言語としては当然の機能だが，Python がもつ手続き抽象化機能 (つまり関数やメソッド) を使用すれば G コードではできないモジュラーな 3D 印刷が実現される．

APT のように機械加工でこれまで使用されてきた言語でなく Python を使用した理由はつぎのとおりである．APT は改良・改訂されてある程度は抽象化の機能ももっているから，それを拡張して 3D 印刷に使用することも不可能ではない．しかし，手続き的な 3D 印刷のためには現在では APT を拡張するより Python のようなプログラミング言語をベースとするほうがよいとかんがえられる．それは，第 1 に現代的な構文・意味をもつ一般に普及した言語をベースにするほうがよいとかんがえられること，第 2 に Python などの言語ならば手続き抽象化機能をふくむ必要な言語機能をすでにもっているため，ライブラリを追加するだけでよいことである．この条件をみたす言語はほかにもあるが，より国際的に普及している Python をベースとすることにした．APT はそのままでは工作機械で実行できないので APT のプログラムを実行すると Cutter Location (CL) ファイルという形式の出力がえられるが，同様に Python ベースの言語を実行して G コードを出力するという方法をとった．

手続き的な 3D 印刷用のライブラリとしては，部品のくみたてと変形によってオブジェクトを生成するための draw3dp.py と，3 次元タートル・グラフィクスによってオブジェクトを生成するための turtle.py とを用意した．このうち turtle.py (http://bit.ly/ZEyLzx または http://www.kanadas.com/program/2014/08/3d_3d_python.html にて公開) は印刷ヘッドの位置をつねに原点とし，その方向をきめてつねに前方にすすむようにしたライブラリ，つまり 3 次元のタートル・グラフィクスのためのライブラリである [8][9]．turtle.py を使用した描画法は，3D 印刷を目的としていることを除外すれば LOGO [18] による描画法にちかい．タートルを中心とする座標系 (フライト・シミュレータと同様の座標系) を使用して，3D 印刷する部品 (図形) を直接きめていく．

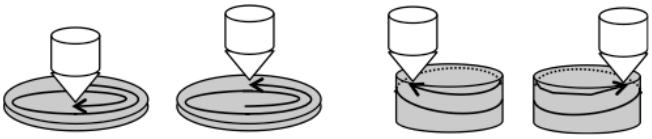
これに対して，draw3dp.py (http://bit.ly/1EBVbSB または http://www.kanadas.com/program/2014/10/3d_python.html にて公開) は直交座標を基本とし 3D 印刷する図形をきめていく．この点では Processing [19] や MetaPost [6], Asymptote [22] などと類似している．しかし，このライブラリとこれらの言語とのちがいは，後述するように前者においては向きのある糸 (線) をつみかさねるという 3D 印刷を指向した手続き的な方法によって図形を生成する，したがって面も糸をかさねたものとしてあつかうのに対して，後者においては面もプリミティブな要素であり，また線には向きがないということである．3D 印刷のためにも使用される CAD ソフトウェア OpenSCAD [23] のプログラミング言語もこれらの言語と同様である．したがって，draw3dp.py においては図形が 3D 印刷という手続きに対応しているのに対して，OpenSCAD もふくめて上記の各言語においてはそれらを対応させることがで

- 構築子: *part* = draw3dp.Trace(crossSection, x, y, z)
  空の部品 *part* を生成し，糸の始点 (現在位置) とその断面積を指定する．
- 低水準機能
  - 移動: *part*.move(x, y, z)
    現在位置から (x, y, z) まで直線的に移動する．(つぎの印刷位置を (x, y, z) とする.)
  - 線分の生成: *part*.draw(x, y, z)
    (x, y, z) まで直線的に移動しながら糸を生成し部品 *part* に追加する．
  - 糸の断面積の設定: *part*.setCrossSection(c) または *part*.thickness(c)
    部品 *part* で今後使用する糸の断面積を c とする．
  - 印刷速度の設定: *part*.setVelocity(v) または *part*.speed(v)
    部品 *part* の今後の印刷速度を v とする．
- 2 次元部品生成 (部品くみたて)
  - 円の生成: *part*.circle(r, x, y, z)
    (x, y, z) を中心とする半径 r の円を部品 *part* に追加する
  - 平面らせんの生成: *part*.spiral(r, hpitch, x, y, z)
    (x, y, z) を中心とする最大半径 r の平面らせんを部品 *part* に追加する．(hpitch は糸の水平ピッチ)．
- 3 次元部品生成 (部品くみたて)
  - 立体らせんの生成: *part*.helix(r, h, vpitch, x, y, z)
    (x, y, z) を中心とする最大半径 r, 高さ h の立体らせんを部品 *part* に追加する (vpitch は糸の垂直ピッチ)．
  - 円柱の生成: *part*.cylinder(r, h, vpitch, hpitch, x, y, z)
    (x, y, z) を中心とする最大半径 r, 高さ h の円柱 (中心までつまったもの) を *part* に追加する (vpitch, hpitch は垂直および水平のピッチ)．
- 部品の変形
  - 直交座標による変形: *part*.deform_xyz(fd, fc, fv)
    関数 fd により部品 *part* の変形前の座標と変形後の直交座標を対応させ，関数 fc により糸の断面積，関数 fv により印刷速度を変換する．
  - 円柱座標による変形: *part*.deform_cylinder(fd, fc, fv)
    関数 fd により部品 *part* の変形前の座標と変形後の円柱座標を対応させ，関数 fc により糸の断面積，関数 fv により印刷速度を変換する．
- 部品 (表面) の変調
  - 円柱座標による変調: *part*.modulate_cylinder(fm)
    関数 fm により部品 *part* を変調する (部品表面にテクスチャを生成する)．
- G コード生成: *part*.draw()
  部品 *part* を印刷する G コードを生成する (部品を確定させる)．

図 3 手続き的な 3D 印刷のための主要な API

きない．OpenSCAD においては他の CAD ソフトウェアと同様に生成した図形を印刷できるようにするためにスライサというプログラムが必要になり，緒言でのべた中空球においてそうであるように，スライサが意図とことなる変換をするとうまく 3D 印刷できなくなる．これに対して draw3dp.py を使用すれば意図したとおりの印刷が実現できる．

図 3 に draw3dp.py がふくむ API をまとめたが，以下，部品くみたて (2 次元，3 次元の図形生成)[10] については 3.3 節，部品の変形 [12] については 3.4 節，また部品 (表面) の変調について 3.5 節においてのべる．



(a) Spiral (b) Helix

図 4 平面らせん (spiral) と立体らせん (helix)

3.2 部品の表現と生成

FDM 方式などの 3D 印刷においては糸状の材料 (FDM 方式においてはフィラメントという) をかさねて造形する．そこで，draw3dp.py においては部品をつぎのように，ふとさのある糸 (string) S_i のならび $(S_1, S_2, ..., S_n)$ として表現する [12]．

$$S_i = (Pstart_i, Pend_i, c_i, v_i)$$

ここで $Pstart_i$ は糸の始点であり $Pend_i$ はその終点である (両者は直線によってむすばれることを仮定する)．c_i は糸の断面積 (これはフィラメントの密度に関するパラメタによっておきかえることもできる), v_i は印刷速度 (秒速) つまりヘッドの移動のはやさである．v_i は概念的には不要だが，実装 (糸の印刷) のためには便利なパラメタである．糸のならびはオブジェクトの表現だが，この表現はオブジェクトが手続き的に生成されることを反映している．あるいは，糸およびそのならびはオブジェクトを手続き的に生成するプログラムであるともいえる．このプログラムは実行されるまえに G-code というプログラムにまで変換される．

この表現においては部品の各点におけるフィラメントの方向が規定され，厚みのある部品は内部のフィラメントの構造や密度も記述される．これらはいずれも従来の CAD や STL においては記述できない特性である．このような表現を選択した本来の意図は 3D 印刷におけるフィラメントの方向を印刷されるオブジェクトの表現 (美観など) にいかすことだった [10][12]．

部品は (オブジェクト指向の意味での) オブジェクトとしてあつかい，そのクラス名は Trace としている．そのため，まず構築子 draw3dp.Trace を使用して空の部品をつくる．(図 3 参照. 以下のメソッドに関しても同様.)

現在ライブラリで用意している単純な部品としては円，平面らせん，立体らせん (ヘリックス) などがあるが，これらのくみあわせではつくれない形状も，線分の生成をはじめとする低水準のメソッドをくみあわせて記述することができる．高水準のライブラリ関数 (メソッド) によっては記述できないプログラム (抽象化された高水準部品) を低水準のライブラリ関数を使用して記述するのは通常の手続き的なプログラム記述と同様である．確立された高水準部品はライブラリに登録するのがよい．

高水準部品はこのライブラリを実際に使用する際に必要になったものから順に実装している．今後も必要性のたか

い部品を追加して拡張していく予定である．一方，これらの部品からはつくれない形状たとえば円弧はプログラマが自分で記述する必要がある．すなわち，G コードと同程度の低水準のメソッドである draw つまり指定された点まで直線をひくメソッド (G1 に相当) や move つまり指定された点までフィラメントを射出せずに移動するメソッド (G0 に相当) などを使用して記述する．糸の断面積や印刷速度も setCrossSecton や setVelocity などの低水準のメソッドを使用して制御する必要がある．

円はライブラリがふくむメソッド circle によって容易に記述できる．糸は線分をであるから，円は線分によって近似される．3D プリンタは通常，円弧を正確にえがくことができないので，これは「糸」という表現の限界であると同時に現在の 3D プリンタの限界でもある．なお，ここには記述しなかったが，近似する線分の数もパラメタとしてあたえることができる．ただし，線分をあまりこまかくすると印刷速度は低下し，印刷が不安定になることもある．

また，メソッド spiral をよびだせば 1 重の平面らせんをえがくことができる (図 4(a) 参照)．より複雑な部品生成メソッドとして helix を用意している．helix は 1 重の立体らせんつまり底面がない円筒をえがく (図 4(b) 参照)．層なしにかつ層のつぎめもなしに，任意の高さのうすい円筒をえがくことができる．

一重の円筒が helix というメソッドによって生成されるのに対して，中心までつまった円柱を生成するには cylinder というメソッドを使用する．ただし，つまった円柱をつぎめなしに印刷することはできない．

上記のメソッドはいずれも下から順にらせん状 (helical または spiral) に印刷する．らせん状に印刷することによってフィラメントが支持されて落下することがないようにでき，また従来の 3D 印刷においてしばしば発生する層間のつぎめがなくせる [13] ため，らせん状の印刷は手続き的な 3D 印刷においてもっとも重要だとかんがえられる．

3.3 部品のくみたて

機械加工によって製品やプロトタイプをつくるときは，まずその部品を製造し，それをくみたてる (くみあわせる) ことが多い．切削加工の場合は加工後にくみたてるが，付加加工の場合は複数の部品を同時に製造することが多い．部品ごとに印刷するだけでなく，あらかじめすべての部品がくみあわせられたかたちで印刷することも多い．従来の 3D 設計・印刷においてはこのようなくみあわせが実現できるが，上記のライブラリにおいても (現在のところは範囲が限定されているが) このような場合を想定している．

上記のライブラリを使用するとき，部品がつぎの 2 条件をみたしていれば，部品を逐次に印刷することによってくみたてることができる．

- 印刷ヘッドがすでに印刷したフィラメントによって妨害されないこと．
- 印刷したフィラメントが印刷台 (プリントベッド) やすでに印刷したフィラメントによって支持されること．

ただし，これらの条件のチェックは自動化されていない．

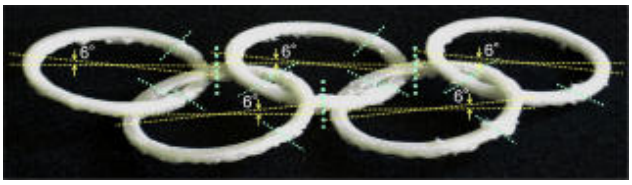


図 5 各リングを分解してくみたてた (印刷した) 五輪

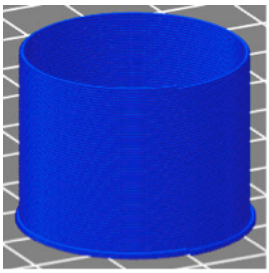
部品の印刷順序をどのようにきめてもこれらの条件をみたすことができないときは，部品を分割すれば条件がみたされるかどうかをしらべればよい [10]．鎖は分割しないかぎり部品 (輪) の印刷順序変更によって印刷可能にはならない．そこで，論文 [10] では分割点を (まだ自動化されていないため) 手動できめて，立体化した (鎖状にした) 五輪の印刷を試行した (図 5)．ただし，五輪は 3.2 節の部品のくみあわせとしては記述できないので，専用の部品 (生成関数) を用意する必要がある．また，この五輪はリングを水平面から浮かせて印刷するため，サポートが必要になる．

部品のくみたて手続きを関数化すれば複合部品を生成する関数となり，モジュラーな構造 (プログラムの構造および印刷された部品の構造) が実現される．

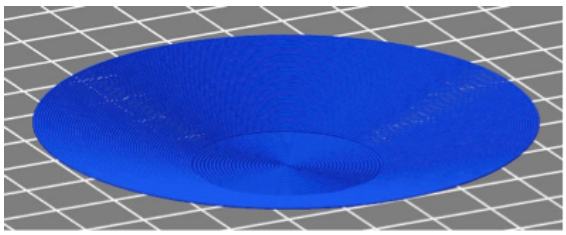
3.4 オブジェクトの変形

ライブラリ draw3dp.py においてはいったん生成した 3 次元または 2 次元の部品を印刷前に変形させられるようにしている．それは，このライブラリには単純な形状だけが登録されていて，それらのくみあわせだけでは多様な形状をつくることは困難だからである．部品あるいはそれをくみあわせたものを自由に変形させられれば，とくに非線形変換を使用すれば，比較的容易に多様な形状を造形できる．従来の CAD においても並進，回転，拡大，縮小などの線形変換が用意されている (3.1 節でふれた宣言型言語 OpenSCAD でもこれらの変換をおこなうことができる) が，これらが宣言的に定義された部品に作用するのに対して，draw3dp.py においては手続き的に糸のならびとして生成した部品を変形することができるという点がちがっている．前記のように部品は手続き的なプログラムだとみなすことができるが，そのときこの変形はプログラム変換だということができる．コンピュータ・グラフィクスにおいてはより自由な変形が重視されている [21][2] が，この場合も変形は宣言的に定義された図形に作用する．変形という操作は多様な形状がつくれるだけでなく，3D 印刷可能 [12] な部品を用意しておけば変形後も印刷可能性が維持しやすいという利点もある．

draw3dp.py においては変形のために deform_xyz と de-



(a) 変形前のカップ



(b) 変形後の皿

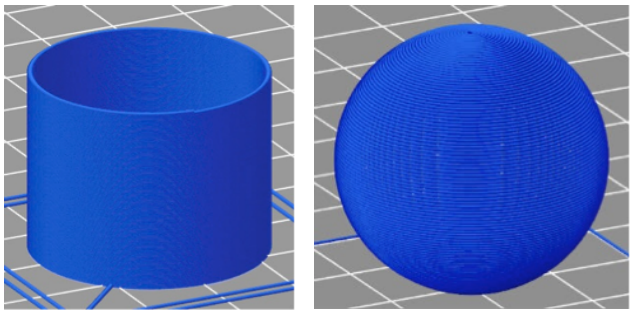
図 6 カップからの変形の例

form_cylinder という 2 つのメソッドを用意し，また部品のかたちを確定させる (G コードを生成する) ためにメソッド draw を用意している (図 3 参照)．変形のための 2 つのメソッドの機能は基本的にひとしいが，2 つ用意しているのは，場合によって直交座標のほうが記述しやすいこともあり，円柱座標のほうが記述しやすいこともあるからである [12]．

メソッド *part*.deform_xyz(fd, fc, fv) は直交座標にもとづいて部品 *part* を変形する．関数 fd(x, y, z) (最初の引数) は変形前の位置 (x, y, z) を変形後の位置にマップする．そのため関数 fd は 3 つの値をかえす．関数 fc(c, x, y, z) (2 番めの引数) は変形前の位置 (x, y, z) における断面積を変形後の位置における断面積に変換する．関数 fv(v, x, y, z) (3 番めの引数) は変形前の位置 (x, y, z) における印刷速度 (ヘッド移動速度) を変形後の位置における印刷速度に変換する．現在は 3D 印刷可能性を自動的に維持する方法がないので，fc および fv として適切な関数をあたえることによって，それを維持する必要がある．

メソッド *part*.deform_cylinder(fd, fc, fv) は円柱座標にもとづいて部品 *part* を変形する．機能は deform_xyz とかわらず，座標系だけがちがっている．すくなくとも現在のところはあつかう部品としてらせん状のものが多いので，直交座標より円柱座標にもとづくこのメソッドのほうが有用である．

これらの変形メソッドは糸の始点と終点の座標を変換する．糸が直線状であることはかわらないので，途中の点の誤差は変化する．変形後も印刷可能であるためには変形関数 (メソッド) は連続であるべきであり，また使用の際に変形による拡大・縮小はおさえるべきである．なぜならば，おおきく拡大・縮小すると誤差が拡大して，うまく印刷できなくなるからである．



(a) 変形前の立体らせん　　(b) 変形後の球

図 7 立体らせんからの変形の例

変形の例を図 6 と図 7 に示す．Repetier Host という 3D 印刷ツールを視覚化のために使用している．

図 6 は立体らせんとうすい円柱 (底) とで構成されるカップとそれを変形してえられた図形を示す．図 6(a) のカップは立体らせんの部分につぎのような変形を適用すると図 6(b) のような皿になる:

deform_cylinder(fdd, fcd, fvd), ここで
fdd(r, θ, z) = (r + 1.05 z, θ, 0.3 z),
fcd(c, r, θ, z) = 0.96 c, fvd(v, r, θ, z) = v.

ただし，底のサイズは立体らせんを変形してえられた図形にあわせる必要がある．

図 7 は立体らせんとそれを変形してえられる図形を図示する．図 7(a) はもとの立体らせんである．この立体らせんから図 7(b) の球がえられる．この変形はつぎの式にもとづいているが，ここではフィラメントのピッチが保存される (すなわち，立体らせんを縦方向には伸縮せずに球にまきつける変形をする):

deform_cylinder(fds, fcs, fvs), ここで
fds(r, θ, z) = (Radius * sin(πz/cylinderHeight),
θ, r — Radius * cos(πz/cylinderHeight)),
fcs(c, r, θ, z) = 1.2 c,
fvs(v, r, θ, z) = 1.2 * ((fr(r, θ, z)/Radius)**2+0.1) v.

パラメタ cylinderHeight は変形前の立体らせんの高さを意味し，変形後の球の経線の長さの半分にひとしい．前論文 [12] にはほかの例もあげている．

3.5 印刷の変調によるテクスチャマップ

提案している 3D 印刷法においては，印刷プロセスを制御することによって，オブジェクトの表面に文字，画像，テクスチャなどをえがく (テクスチャマップする) ことが比較的容易にできる．この方法ではフィラメントの断面積を変化させて凹凸をつくる [14]．この方法を適用することを印刷を変調する (modulate) ということにする．印刷中にフィラメントの断面積を変化させれば，あまり大きな凹凸をつくることはできないが，比較的こまかい凹凸をつくることができる．

フィラメントの断面積を変化させる方法としてはつぎの

図 8 2 値のビットマップ地図

2 つがある．

- フィラメントの射出速度を変化させる．
- 印刷ヘッドの移動速度を変化させる．

第 1 の方法のほうが直接的だが，より応答性がよい第 2 の方法によって変調している．3D プリンタにおいてはフィラメントを押し出すエクストルーダ (extruder) の動作の変化がヘッド先端のフィラメントの動作に反映されるまでの遅延が大きい．場合によっては数秒かかる．そのため第 1 の方法は応答性がわるい．3D プリンタの印刷ヘッドは慣性が大きいが，それでも印刷ヘッドの速度のほうがはるかに応答性がよい．したがって，第 2 の方法のほうがこまかい凹凸をつくるという目的には適切である．そのため，この方法を使用している．変調のためには modulate_cylinder というメソッドを用意している (図 3 参照)．

以下, 例として地図による変調をとりあげる．地図によって平面を変調すれば地図がそのまま表現されるが, 球を変調すれば地球儀がつくれるはずである．図 8 は Celsia Motherload (http://www.celestiamotherlode.net/catalog/earth.php) というサイトからえた NASA のデータにもとづく正距円筒図法による世界地図を 2 値化した地図である．このサイトにはさまざまな加工をした地図があり，ビットマップのサイズもさまざまなものがあるが，図 8 に示した地図のサイズは 300 × 150 であるから，角度にすれば経度，緯度ともに 1.2° ごとにドットをマップすればよい．フィラメントの 1 周を基本的には 300 個の糸によって構成し，150 周で球を形成する．そして各糸にドットをマップする．すなわち糸の印刷速度を 2 つの選択肢 (2 値) からえらぶ．

3.6 プログラム例

これまでの説明だけでは処理のながれがつかめないので，球を印刷するための図 9 のプログラムを使用して印刷全体のながれ (正確には印刷用 G コード生成のながれ) を説明する．このプログラムにおいては定数やパラメタを設定したあと，init というメソッドをよびだして，全体的な初期化 (とくに 3D プリンタの初期化) をしている．そのあと「スカート」の印刷をおこなっているが，これは 3D プリ

```
import draw3dp
from math import sin, cos
## 定数 ##
PI = 3.14159265359
## プリンタ・パラメタ ##
IsABS = False       # 材料として PLA を使用する
DefaultVelocity = 40    # mm/sec
## 印刷パラメタ ##
x0 = 0; y0 = 0; z0 = 0.4
## 射出パラメタ ##
defaultCrossSection = 0.196 # mm^2 (直径 0.5 mm)
FilamentDiameter = 1.75 # mm (通常 1.75 mm または 3 mm)
## 温度パラメタ ##
if IsABS:
    HeadTemperature = 235 # ABS のほうがやや高温を要する
    BedTemperature = 90    # ABS はプリントベッド加熱要
else: # PLA
    HeadTemperature = 220
    BedTemperature = 35    # PLA のときは室温程度

## 初期化 ##
draw3dp.init(FilamentDiameter, HeadTemperature,
                BedTemperature, DefaultVelocity)
## スカートの生成と印刷 ##
sk = draw3dp.Trace(defaultCrossSection, 0, 0, 0.4)
skirt2(sk)    # skirt2 の定義は省略
sk.draw(0.4)

## 印刷するべきオブジェクトの生成 ##
obj = draw3dp.Trace(defaultCrossSection, x0, y0, 0.4)
radius = 25.0
helixHeight = PI/2 * radius
rmax = 30.0
vpitch = 0.2; x0 = 0; y0 = 0; z0 = 0.4
obj.setVelocity(36)             # 初期印刷速度の設定
obj.helix(radius, helixHeight, vpitch, x0, y0, 0)
                                  # 立体らせんの生成
obj.deform_cylinder(
    lambda r, theta, z:
      (radius * sin(PI*z/helixHeight), theta,
      r - radius * cos(PI*z/helixHeight)),
    lambda v, r, theta, z: v,
    lambda c, r, theta, z:
        0.35 * ((0.5 * r + radius) / radius) * c)    # 変形 1
obj.deform_cylinder(
    lambda r, theta, z: (r, theta, z + z0),
    lambda v, r, theta, z: 0.6 * ((r/radius)**1.5 + 0.2) * v,
    lambda c, r, theta, z: 2.0 * c)                  # 変形 2
## 印刷 ##
obj.draw()
```

図 9 中空の球を印刷するプログラム


(a) 4 サイクルの皿


(b) 3 サイクルの皿


(c) ハート型の皿

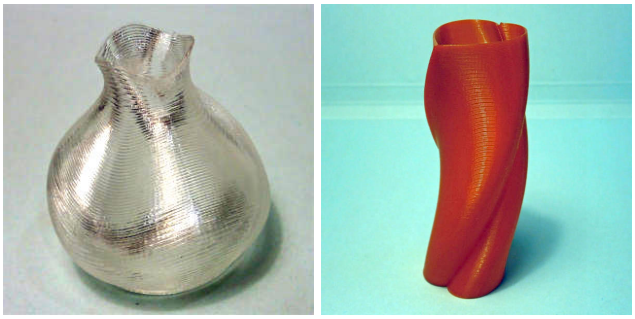

(d) 花瓶

図 10 皿と容器

ンタではいきなり印刷してもうまくいかないので，フィラメントの状態をととのえるために，印刷するべきオブジェクトの周囲にフィラメントをはきだすためにおこなっている．obj が印刷するべき部品であり，helix メソッドによって obj を立体らせんにした (立体らせんを構成する糸を obj にくわえた) あと，それに 2 回 deform_cyliner メソッドを適用して球に変形している．おもな変形は最初の適用においてなされるが，2 回めにはそれをすこし z 軸方向に移動させ，印刷速度とフィラメント射出速度を調整している． 最後に draw メソッドによって糸に対応する G コードを生成する．

4. 手続き的な 3D 印刷の実践

前章の方法によって，10 〜 20 分程度で印刷できる小型の皿，容器，球，地球儀などの製造をためした．まだ実用品を製造しているとはいえないが，この章で紹介するオブジェクトのサンプル品はすべて読者が入手可能である (http://bit.ly/1EZ4SZI または http://store.shopping.yahoo.co.jp/dasyn/)．

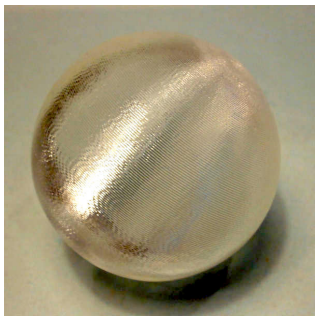



図 11 球と変形球

4.1 カップ形をもとにした皿と容器

3.4 節で説明した方法でカップを変形してえられたさまざまな形状の皿や容器を図 10 に示す．図 10(a), (b) は立体らせんを図 6(b) のようにひらたく変形してつくった皿 [3] だが，フィラメントを水平にちかい角度でまきつけるときも，印刷物をよごす原因となるサポートは不要である．(a) は 4 サイクルつまり印刷ヘッドが 1 周するあいだに三角関数にもとづく 4 回の周期的な動作をするように変形したもの，(b) は 3 サイクルつまり同様に 3 回の周期的な変形を適用したものである．いずれも deform_cylinder メソッドの引数 fd において三角関数を指定している．場所によって皿の角度やフィラメントの密度がことなるために光の反射に変化が生じている [12]．このような輝きは，純粋な透明の PLA (ポリ乳酸) を使用することによって，またサポートをなくすことによって，実現されている．着色されたプラスティックではこのような輝きは生じない．

図 10(c) は立体らせんに円をハート型に変換する変形を適用してつくった皿 [4] である．円をハート形に変形させるつぎの関数を使用している．

fdh(x, y, z) = (x + b z sqrt(abs(y) / radius), y, z)

この曲線のもとになったのは「ハート形曲線の方程式」[26] である．パラメタ b の妥当な範囲は 0 〜 1.2 くらいである． b z = 0 なら恒等変換になり，b z が大きいほど鋭利なハート形になる．この関数と deform_xyz を使用して変形している．底面では b z = 0 となるのでその形状は円形になり，b z の値は z に対して単調増加なので上部にいくほど鋭利なハート形になる．b の最大値を変化させることにより図 10(c) のさまざまな形状ができる．さらに，図 10(c) においては上下の振動 (三角関数) もくわえることによって皿の角度と輝きに変化をつけている．図 10(d) には 2 種類の花瓶を示す．左の花瓶 [14] は三角関数により半径方向，高さ方向の変形をくわえている．右の花瓶は上記とおなじハート形をねじった (高さによってハート型の向きが変化するようにした) ものである．

4.2 立体らせんをもとにした球など

3.4 節で説明したように，立体らせんを変形して球をつくることができる．図 11 の左はこうしてつくった単純な

図 12 変調した球 (地球儀) と椀

図 13 小型照明器具用シェード

球である．印刷時に球を 1 点だけでは支持できないため，支持にすこしくふうが必要である．しかし，通常の意味のサポートは使用していない [12]．図 11 の右は図 10(a), (b) などと同様に三角関数によって球をさらに変形してつくったオブジェクトである．

図 12 には 3.5 節で説明したビットマップによる変調の技法を使用してつくったオブジェクトを示す．左は球を地図によって変調してつくった地球儀 [5] である．この球は下方から 1 個の LED の光をあてるだけで全体をかがやかせることができる [15]．右は半球 (正確には半球だけでなく高台をつけている) を英字によって変調してつくった椀である．(これは上下逆に印刷している.)

地球儀の印刷においては，それを構成する糸ごとにその印刷速度を 2 つの選択肢 (2 値) からえらぶ．海と陸のフィラメント断面積の比 (速度比) をあまりおおきくすると印刷がうまくいかなくなるので，1 対 0.6 ないし 0.7 くらいにしている．

前記のように，300 × 150 のビットマップを使用するときには，基本的にはどの緯度でも 1 周を 300 等分する．しかし，このままでは極のちかくでは細分化されすぎる．そこで，極のちかくでは線分をまびいて印刷する．地球儀の表現 (線分の列) を生成する際にそうすることもできるが，draw というメソッドにみじかい線分をまびく機能をいれているので，表現は変更しなくてもよい

図 13 は立体らせんから変形した LED 電球のための小型のシェード [11] である．左は球の一部をカットした形状に三角関数による変形をくわえてつくったものであり，右はおなじ形状を三角関数を使用して変調したものである．3D 印刷の材料のおおくは熱によわいため白熱電球の器具として使用するのは困難だが，LED は発熱がすくないので PLA でも使用可能であろう．このシェードはこの章で紹介したオブジェクトのなかでは最大のもの (それでも直径 100 mm ほど) だが，印刷時間は 20 分程度である．1 重のフィラメントでできているため印刷時間は比較的みじかいが，落としても容易にはわれないくらいの強度はある (この点は他のオブジェクトについても同様である).

5. 関連研究

提案した方法はらせん状に印刷する手続き的な部品のくみあわせで 3D 印刷するものをモデリングするところに特徴があり，前章で紹介した例はこの方法により透明なフィラメントをシームレスにかつ美的につみかさねている．モデリングの方法に関する研究ではないが，MIT の Klein ら [16] は透明なガラスによるシームレスかつ美的な 3D 印刷を実現し，そこに掲載された写真においては光の反射や屈折をうまく利用している．しかし，それらの作品の設計法については論文には記述されていない．

6. 結言

現在の 3D 設計・印刷法は宣言的であり，設計者による手続き的な記述は切削加工においては成功していない．しかし，それが付加加工においては利点があると著者はかんがえて，3D 印刷用ライブラリを開発し，手続き的に抽象化された Python プログラムによって G コードのプログラムを生成して 3D プリンタで印刷する方法を開発し，実際に印刷してみた．この方法では印刷可能な形状は限定されるが，層をなくして層のつぎめもなくし，従来の方法においては必要だった支持材料 (サポート) もなくして，シームレスでより美的な印刷を実現した．この論文をきっかけにしてこの方法とライブラリをひろめていきたい．

参考文献

[1] Brown, S. A., Drayton, C. E., and Mittman, B.: A Description of the APT Language, *Communications of the ACM*, Vol. 6, No. 11, pp. 649–658 (1963).

[2] Coquillart, S.: Extended free-form deformation: a sculpturing tool for 3D geometric modeling, *ACM SIGGRAPH Computer Graphics*, Vol. 24, No. 4, pp. 187–196 (1990).

[3] Dasyn.com: Printing a dish by helical 3D-printing method, http://youtu.be/5P1vaahzW98

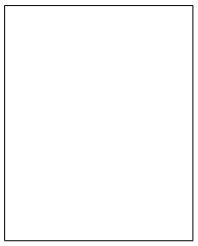
[4] Dasyn.com: 3D-printing a dish with various heart shapes, http://youtu.be/G9x14DZYN_8

[5] Dasyn.com: Creating a Globe by Helical 3D-printing Method, http://youtu.be/YWx1vqig2-o

[6] Hobby, J. D.: METAPOST: A User's Manual, version 1.999 (2014).

[7] ISO: Numerical Control of Machines – NC Processor Input – Basic Part Program Reference Language, ISO 4342:1985 (1985).

[8] 金田 泰: 3D プリンタによる “3 次元タートル・グラフィクス”, 情報処理学会夏のプログラミングシンポジウム (2014).
[9] Kanada, Y.: “3D Turtle Graphics” by using a 3D Printer, *Int. Journal of Engineering Research and Applications*, Vol. 5, No. 4 (Part-5), pp.70–77 (2015).
[10] Kanada, Y.: Method of Designing, Partitioning, and Printing 3D Objects with Specified Printing Direction, *2014 International Symposium on Flexible Automation (ISFA)* (2014).
[11] 金田 泰: 3D 印刷したシェイドをつけた LED 電球などの写真, カナダからのブログ, 2014-12-13 (2014).
[12] Kanada, Y.: Support-less Horizontal Filament-stacking by Layer-less FDM, *International Solid Free-form Fabrication Symposium 2015* (2015).
[13] 金田 泰: 3D プリンタ「サポート」なしで皿をつくる, I/O 2015 年 4 月号, pp. 15–17 (2015).
[14] Kanada, Y.: Creating Thin Objects with Bit-mapped Pictures / Characters by FDM Helical 3D Printing, *8th Int’l Conference on Leading Edge Manufacturing in 21st Century (LEM21)* (2015).
[15] Kanada, Y.: Designing 3D-Printable Generative Art by 3D Turtle Graphics and Assembly-and-Deformation, *XIIIV Generative Art Conference (GA 2015)* (2015).
[16] Klein, J., Michael, S., Giorgia, F., Markus, K., Chikara, I., Shreya, D., James, C. W., Peter, H., Paolo, C., Yang Maria, and Neri, O., *3D Printing and Additive Manufacturing*, Vol. 2, No. 3, pp. 92–105 (2015).
[17] Kramer, T. R., Proctor, F. M., and Messina, E.: The NIST RS274NGC Interpreter - Version 3, *NISTIR 6556* (2000).
[18] Papert, S. A.: Mindstorms: Children, Computers and Powerful Ideas: All About Logo, How It Was Invented and How It Works, Basic Books (1993).
[19] Pearson, M.: Generative Art: A Practical Guide Using Processing, Manning Publishing Co. (2011).
[20] Ross, D. T.: Origins of the APT Language for Automatically Programmed Tools, *ACM SIGPLAN Notices*, Vol. 13, No. 8, pp. 61–99 (1978).
[21] Sederberg, T. W. and Parry, S. R.: Free-form deformation of solid geometric models, *ACM SIGGRAPH Computer Graphics*, Vol. 20, No. 4, pp. 151–160 (1986).
[22] Staats III, C.: An Asymptote Tutorial, https://math.uchicago.edu/c̃staats/Charles_Staats_III/Notes_and_papers_files/asymptote_tutorial.pdf (2015).
[23] WikiBooks: OpenSCAD User Manual, available at http://en.wikibooks.org/wiki/OpenSCAD_User_Manual
[24] Wikipedia: History of Numerical Control, http://en.wikipedia.org/wiki/History_of_numerical_control
[25] Wikipedia: STL (file format), http://en.wikipedia.org/wiki/STL_(file_format)
[26] 山本信雄: ハート形曲線, http://www.geocities.jp/nyjp07/heart/index_heart.html



金田 泰 （正会員）

1956 年生．1979 年東京大学工学部計数工学科卒業．1981 年同大学大学院情報工学専門課程修了．同年 (株) 日立製作所入社後，中央研究所，システム開発研究所，テクノロジーイノベーションセンタ，カーネギーメロン大学，RWCP つくば研究所等において Fortran コンパイラ，ベクトル記号処理と論理型言語処理，創発的計算のモデル，情報抽出/検索/組織化，仮想環境型コミュニケーション, ネットワークとポリシー，ネットワーク仮想化，深層学習によるコンピュータビジョン等の研究開発に従事．博士 (工学)．工学院大学情報学部非常勤講師．ACM，IEEE (Computer, ComSoc, SMC)，ソフトウェア科学会，人工知能学会，日本機械学会各会員．