Method for Procedural 3D-printing Using a Python Library

YASUSI KANADA¹,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 manu-
facturing, 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
te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]
というアセンプラ風の言語が使用される。G コードはもと
もと工作機械の方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
コードはもとと切削加工 (現在の用途では 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
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3DCAD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、その結果をブリンタにおきかけて印刷する。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なが、スライサにおいてはSTL（Standard Triangulation Language）またはStereo-Lithography [25]という宣言的形状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, 熱溶融積層）型とよばれる。たとえしたフィラメント（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）。
3D ブリクタは通常は層ごとに印刷するが、3D ブリク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いない、スライシング結果は通常 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 ブリクタでは、水準にスライシングされた層ごとに印刷するので、層間の移動のとき以外はブリクタ・ヘッドが垂直方向にうごくことはない。しかし、G コードは層にしようとしていないので、実はヘッドをもっと自由にうがすことができる。

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離 3600 mm での座標 (1, 2, 3) に移動することを指令する。また、G1 というコマンドは垂直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 ブリク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 によって指定された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

切削加工はちがって 3D 印刷のような付加加工においてはより直接的に機械加工がでるが、切削加工の場合ときがて、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からとがえる、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 (APT, 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 コードではないモジュールな 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・改訂されてい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 を拡張するよりも Python のようなプログラミング言語ベースとするのがよいとかがえられる。それは、第 1 に現代的な構文・意味をもつに一般に普及した言語をベースにするほうがよいとかがえられるから、第 2 に Python などの言語ならば手続き抽象化機能をよくも必要な言語機能をすぐにでもついているため、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oda Cutter Location (CL) や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、部品のくみ立て変形によってオブジェクトを生成するための draw3dp.py と、3 次元アール・グラフィックスによってオブジェクトを生成するための turtle.py 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.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 次元のアール・グラフィッ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と除外すれば LOGO [18] による描画法にちかい、アー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 (图形) を直接生成していく。

これに対して、draw3dp.py (http://bit.ly/IEBVbSB 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かいは、後述するように前者においては向きのある線 (線) つつきがされるという 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。したがって面向を含められたものとしてあるからのに対して、後者がにおいては面もアフィニティな要素であり、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 もあくまでも上記の各言語においてはそれらを対応させることがで
情報処理学会論文誌 プログラミング

・構築子: `part = draw3dp.Trace(crossSection, x, y, z)`
空の部品 `part` を生成し、実の始点 (現在位置) とその断面数を
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.drawLine(x, y, z)`
   (x, y, z) まで直線的に移動しながら線を生成し部品 part に追
   加する。
  - 空の断面の設定: `part.setCrossSection(c)` または `part.thickness(c)`
   部品 part で今後使用する空の断面数を 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 (部品くみで)
  - 円の生成: `part.circle(r, x, y, z)`
    (x, y, z) に平行する半径 r の円を部品 part に追加する
  - 平面をベースの生成: `part.spiral(r, h, pitch, x, y, z)`
    (x, y, z) に中心とする最大径 r の平面をベース部品 part に
    追加する。（pitch は円のピッチ。）
  - 3 次元部品生成 (部品くみで)
    - 立体部品の生成: `part.helix(r, h, pitch, x, y, z)`
      (x, y, z) に中心とする最大径 r, 高さ h の立体を部分部品
      part に追加する。（pitch は円のピッチ。）
    - 円柱の生成: `part.cylinder(r, h, pitch, x, y, z)`
      (x, y, z) に中心とする最大径 r, 高さ h の円柱 (中心までつ
      まったもの) を part に追加する。（pitch, pitch は垂直およ
      び水平のピッチ。）

・部品の変形
  - 直交座標における変形: `part.deform_xyz(fd, fc, fv)`
    関数 fd により部品 part の変形前の断面と変形後の直交座
    標を対応させ、関数 fc メンタリーより断面数, 関数 fv により印刷速
    度を変換する。
  - 円柱座標による変換: `part.deform_cylinder(fd, fc, fv)`
    関数 fd により部品 part の変形前の断面と変形後の円柱座
    標を対応させ、関数 fc メンタリーより断面数, 関数 fv により印刷速
    度を変換する。

・部品（表面）の変換
  - 円柱座標による変換: `part.modulate_cylinder(fm)`
    関数 fm により部品 part を変換する (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 3 手続き的なる 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、著作でのべた中空球
においてそうであるように、スライサが図意をとることを変
換をするよう 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、
部品くみで (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) の変換について 3.5 節においてのべる。

3.2 部品の表現と生成

FDM 方式などの 3D 印刷においては余分の材料 (FDM
方式においてはフィラメントという) をかせぐて作形する。
そこで, draw3dp.py においては部品をつぎのように、ふ
とさきのあるみ (string) Si のならば (Si, S1, ..., Sn) として
表現する [12]。

\[ S_i = (P_{start}, P_{end}, c_i, v_i) \]

ここで Pstart はみの始点であり Pend はその終点であ
る（両端は直線によってむすばえることを仮定する）, c_i は
みの断面数 (これはフィラメントの密度に関するパラメ
ター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やすさである。v_i は概念的には不
要だが、実装 (みの印刷) のためには便利なパラメータであ
る。み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、みおよびそのならば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
は実行されるまでに 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
ッ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
くられる (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
平面させ、立体させた (ヘリックス) などがあるが、これ
らのくみあわせではつかれない形状も、線分の生成をはじ
めと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間数 (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
水準のライブラリ間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だと仮定すると円弧はプログラム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もたくメソッド (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) などを利用して記述する。系の断面積や印刷速度も setCrossSection 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。系は線分をであるから、円は線分によって近似される、3D プリントは通常、円弧を正確にえがくことができないのので、これは「糸」という表現の限界であると同時に現在の 3D プリントの限界である。なお、ここには記述しなかったが、近似する線分の数をパラメータとして与えることができる。ただし、線分をあまりこまな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 をよそでは 1 重の平面上をえがくことができる (図 4a 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 重の立体らせん状底面がない円筒をえがく (図 4b 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、つまった円筒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できないようにでき、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なくなる [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思うがえられる。

3.3 部品のくみ立て

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、それをくみつける (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つけるが、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ていたかを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ず、上記のライブラリにおいても (現在のところは範囲が制限されている) 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 条件を見つめていれば、部品を逐次に印刷することによってくみたることができる。

• 印刷ヘッドがすでに印刷したフィラメントによって妨

図 5 各リングを分解してくみたてた (印刷した) 五輪

害されないこと。

• 印刷したフィラメントが印刷台 (プリンタヘッド) やすで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。

部品の印刷順序をどのようにじてはこれらの条件をみたすことができないとき、部品を分割すれば条件がみたされるかどうかをしらせればよい [10]。頭角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] では分割面を (まだ自動化されていないため) 手動で与えて、立体化した (黒状にした) 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。

部品のくみ立てで手続き的に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 オプジェクトの変形

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るようになっている。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る、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、これが宣言的に定義された部品に作用するのに対し、draw3dp.py においては手続き的に系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でみなし得ることができるが、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ックスにおいてはより自由な変形が可能である [21][22] 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をつくるだけでなく、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-
form_cylinder という 2 つのメソッドで用意し、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は 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
ために関数 fd は 3 つの値をもつ。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で
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 とな
がらず、座標系だけが変わっている。すぐ近くの現在の
ところはあっから部品として MacOS 版のものが多いので、
直交座標よりも円柱座標に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系の始点と終点の座標を変換す
る。系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・縮小は現されるべきである。なぜならば、
おおきに拡大・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。

(1) 变形前のカップ
(2) 变形後の皿
図 6 カップからの変形の例

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する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た変形図を示す。図 6(a) の
カップは立体らせんの部分につぎのよう変形を適用する
と図 6(b) のような図になる。

deform_cylinder(fd, fc, fv), そので
fdd(r, θ, z) = (r + θ + 1.0 z, θ, θ, 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た変形
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た変形図を図示
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ブロックが保存さ
れる (すなわち、立体らせんを縦方向には伸縮させずに球に
まきつける変形をする):

deform_cylinder(fds, fcs, fvs), そので

fds(r, θ, z) = (Radius * sin(πx/cylinderHeight)),

θ, r = Radius * cos(πx/cylinderHeight)),
fcs(c, r, θ, z) = 1.2 c,
fds(v, r, θ, z) = 1.2 * ((fr(r, θ, z)/Radius)**2+0.1) v.

パラメータ cylinderHeight は変形前の立体らせんの高さを
意味し、変形後の球の縦線の長さの半分にほしい。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レンジ

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 (テクスチャレンジする) 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、比較的まんや凹凸をつく
ることができると。

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値のピットマップ地図

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の方法は、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の方法は応答性が低い。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の方法のほうがはままなり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9は Cheerio サイトからえた NASA のデータのもとを正三角形網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、ピットマップのサイズもさまざまなものがある。が、図9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 個の点にによって構成し、150 周で球を形成する。そして各点にドットをマップする。すなわち点の印刷速度を 2 個の選択肢（2値）からおろべ。

3.6 プログラム例
これまでの説明だけでは処理のつながりがつかないのよので、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つながり (正確には印刷用 G コード生成のつながり) を説明する。このプログラムにおいては定数やパラメータを設定したあと、init というメソッドをよくして、全体的な初期化 (ときに 3D プリンタの初期化) をしている。その後「スカート」の印刷をおこなっているが、これは 3D プリンタで扱う。
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かいく角度でまきつけたもと、印刷物をよごす原因となるサポートは不要である。しかし4サーチ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、図10(c)は3サーチつまり同様に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id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違うため、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。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)は立体らせんを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。パラメータbの値の範囲は0〜1.2くらいである。b z = 0 なら恒等変換になり、b z が大きいほど鈍角なハート形になる。この関数とdeform_zyxを使用して変形している。面向面ではb z = 0 となるのでその形状は円形になり、b z の値が 0 に対して単調増加する上部にいくほど鈍角なハート形になる。b z 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)をもつもつことによって皿の角度と輝きに変化をつける。図10(c)は2種類の花瓶を示し、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
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フィラメントをシミュレーションにかつ美的にみかされている。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ミュレーションを実現し、そこに掲載された写真において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・印刷法は言説的であり、設計者の手続きの設計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、手続き的に抽象されたPythonプログラムによって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。この方法ではなく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、シミュレーションに印刷機器の実現し、この論文をきっかけにしてこの方法を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 sculpting 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://youtube.com/SP1vahzW98
[4] Dasyn.com: 3D-printing a dish with various heart shapes, http://youtube.com/G9x14DZYN_8
[5] Dasyn.com: Creating a Globe by Helical 3D-printing Method, http://youtube.com/YWx1vqg2-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).

3D印刷の材料のおほどくは熱に遊ばないため白熱電球の器具として使用するのは困難だが、LEDは発熱がするのでPLAでも使用可能である。このシェードはこの章で紹介したオブジェクトのかなでは最大のもの（それでも直径100mm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るとかいが、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）。

図12 変調した球（地球儀）と梅

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、支持にすこし足が必要である。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形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やませることができる[15]。右は半球（正確には半球だけでなく高台をつけている）を英字によって変形してつくった梅である。（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、基本的にはどの線度でも1周を300等分する。しかし、このままででは極のちかいは細分化されすぎる。そこで、極のちかいでは線をまきにして印刷する。地球儀の表現（線の列）を生成する際は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き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。
[8] Kanada, Y.: 3D プリンによる“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 Ap-
_plications, 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] Kanada, Y.: 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 Fab-
rication Symposium 2015 (2015).
[13] Kanada, Y.: Creating Thin Objects with Bit-mapped
Pictures / Characters by FDM Helical 3D Printing, 8th
Intl'l Conference on Leading Edge Manufacturing in
21st Century (LEM21) (2015).
[14] Kanada, Y.: Designing 3D-Printable Generative Art by
3D Turtle Graphics and Assembly-and-Deformation, XI-
IV Generative Art Conference (GA 2015) (2015).
[15] 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 Manu-
facturing, Vol. 2, No. 3, pp. 92-105 (2015).
[16] Kramer, T. R., Proctor, F. M., and Messina, E.: The
NIST RS274NGC Interpreter - Version 3, NISTIR 6556
(2000).
[17] Papert, S. A.: Mindstorms: Children, Computers and
Powerful Ideas: All About Logo, How It Was Invented
and How It Works, Basic Books (1993).
[18] Pearson, M.: Generative Art: A Practical Guide Using
Processing, Manning Publishing Co. (2011).
[19] Ross, D. T.: Origins of the APT Language for Automat-
ically Programmed Tools, ACM SIGPLAN Notices,
Vol. 13, No. 8, pp. 61–99 (1978).
[20] Sederberg, T. W. and Parry, S. R.: Free-form deforma-
tion of solid geometric models, ACM SIGGRAPH Com-
puter Graphics, Vol. 20, No. 4, pp. 151–160 (1986).
[21] Staats III, C.: An Asymptote Tutorial,
https://math.uchicago.edu/~cstaats/Charles_Staats_III/
Notes_and_papers_files/asymptote_tutorial.pdf (2015).
[22] WikiBooks: OpenSCAD User Manual, available at
http://en.wikibooks.org/wiki/OpenSCAD_User_Manual
[23] Wikipedia: History of Numerical Control,
http://en.wikipedia.org/wiki/History_of_numerical_control
[24] Wikipedia: STL (file format),
http://en.wikipedia.org/wiki/STL_file_format
[25] 山本 信雄: ハート形曲線,
http://www.geocities.jp/nyjp07/heart/index_heart.html