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 subtractive manufacturing (cutting work), and designers have historically written programs in G-code, but, in recently developed environments, the designer describes a declarative model by using computer-aided design (CAD), and the computer converts it to a G-code program. However, because the process of additive manufacturing, of which FDM-type 3D-printing is a prominent example, is more intuitive than subtractive manufacturing, it is sometimes advantageous for the designer to describe an abstract procedural program for this purpose. This paper therefore 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 it presents use cases. Although shapes printable by the method are restricted, this method can eliminate layers and layer seams as well as support, which is necessary for conventional methods when an overhang exists, and it enables seamless and aesthetic printing.

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

1. Introduction

When a computer is used to process physical parts by machining, the machining procedure is usually described by a language called G-code. 3D printing is a form of additive manufacturing (AM), which is a type of machine processing that requires a program for controlling manufacturing. For this purpose, an assembly-language-like language called G-code [17] was used. G-code is originally used for describing the motion of blades of machine tools, so it is procedural.

Originally, programs written in G-code or APT, which is a procedural language introduced in Section 2, were described by the designer of physical parts. Today, however, the designer describes a declarative model of parts by computer-aided design (CAD). G-code was originally developed for cutting work (or subtractive manufacturing, to use the current term). A computer converted a model to procedural G-code.

In cutting work, procedural design has been completely replaced by declarative design, but the author suggests that, in additive manufacturing, procedural design methods still have advantages. Procedures of cutting work have many constraints and are complicated, so a procedural description is not suited for describing cutting work. Moreover, languages for describing it such as G-code are very low-level and have no abstraction mechanisms, so it was difficult to program machine work by using such languges. However, because AM, such as 3D printing, is more intuitive than subtractive manufacturing and because it may be difficult to print parts using declarative method, it is easier for the designer to design shapes using abstract procedural description. For example, it is difficult to generate an empty sphere, which is similar to the one shown in Figure 1, by normal layer-by-layer 3D printing, because of the following reason. Almost horizontal overhang of filament is required when printing near the top of the sphere, so support inside the sphere is inevitable, and it is difficult to make it empty. However, using an intuitive and elaborate procedural description, a clean-shaped empty sphere can be printed. This is similar to software development; that is, it is easier to describe the intended behavior of the printer with a procedural language than with a declarative language. In contrast to machining, in software development, procedural methods are still the mainstream.

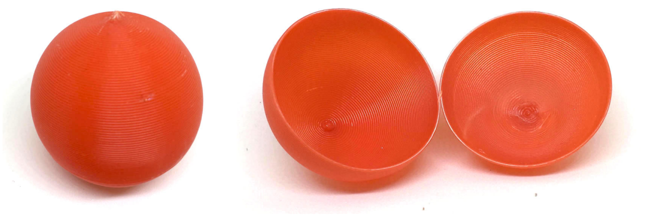


Fig. 1 Example of procedurally 3D-printed empty sphere (surface and inside)

This paper therefore presents a method for generating G-code by a Python program that uses a library for 3D printing and is procedurally abstracted, presents a method for printing using a 3D printer, and demonstrates the use of this method. The key aims of this paper are to describe the pro-

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

posed method from the viewpoint of programming and to position the method in the history of machining and in the range of conventional methods. In Section 2, the history of procedural cutting work, the basic method for 3D printing, and the programming method for 3D printing are described. The procedural 3D printing method using a Python library, which was developed by the author, is described in Section 3, and the experimental use of procedural 3D printing is introduced in Section 4. Related work is briefly described in Section 5, and Section 6 concludes the paper.

2. Conventional Cutting Work and Additive Manufacturing

This section examines the value of procedural description in machining and describes practices of 3D printing. First, the history of machining is reexamined, and second, the 3D printing method and programming 3D printers are described.

2.1 History of procedural cutting work

The technologies of cutting work using computerized numerial control (CNC) and a programming language called APT for CNC were developed from the 1940s to the 1950s. The technology of numerical control was invented by John T. Parsons in 1942, and based on this technology, the technology of CNC was developed at the Massachusetts Institute of Technology (MIT) [24]. For the control of cutting work, a programming language called APT [1][20][7] was developed at MIT, and it was used for programming cutting work. Like an assembly language, APT is procedual. Parsons used punch cards to record CNC programs, but paper tapes were used at MIT for this purpose. MIT researchers developed various subroutines and primitive procedural abstraction mechanisms such as macros or nested definitions [20]. In the 1970s, the relationships between APT and data abstraction or object-orientedness were discussed [20].

However, after initial CAD technologies were developed, designers of physical parts seldom ran machines using procedural methods. The designers described declarative models, and computers converted them to procedural programs. Declarative methods were used probably because of the complex nature of cutting work, which is not amenable to procedural description or the low-level and weak abstraction functions of G-code and APT. Despite the various improvements to APT, certain limitations, such as compatibility, remained. It was likely difficult for designers to program cutting machines using APT.

2.2 Conventional 3D printing and g-code

When using a 3D printer to shape a 3D object, a model is typically designed by a 3D CAD tool and horizontally sliced by a program called a slicer; the result is sent to a 3D printer, and it is printed. When using a CAD tool, the model is usually *procedurally* designed using a graphical user interface, but the model itself is declarative. For example,

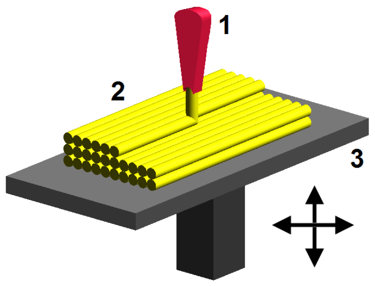


Fig. 2 FDM-type 3D printers (FDM by "Zureks" by Zureks - Wikimedia Commons)

a 3D design tool called OpenSCAD [23], which is a CAD tool used for 3D printing, is unique because, in this environment, a model is described using a programming language; however, this description is still declarative. The output file formats varies among CAD tools, but a standard declarative format called STL (Standard Triangulation Language or Stereo-Lithography) [25] is used to send data to a slicer. STL approximates the surface shape of models by a collection of triangles. It does not express the internal structure of models.

There are many types of 3D printers. The cheapest type, which is widely used (and used in this study), is called fused deposition modeling (FDM). This type of printers extrudes melted filament (plastic) from the nozzle and solidify it (Figure 2).

Conventional 3D printers basially print objects on a layer-by-layer, but G-code itself is not constrained by the concept of layer. The result of slicing is usually expressed by G-code, and it specifies the behaviour of the print head, the extrusion speed of plastic, and so on. 3D printers typically print horizontally and layer by layer, so they do not usually move the print heads vertically, except when transitioning between layers. However, because G-code is not restricted by the concept of layer, they do have the capacity to move the heads much more freely.

Two examples of G-code commands are shown below. First, a command named G0 specifies a simple tool (head) motion. For example, command "G0 X1 Y2 Z3 F3600" describes a motion at the rate of 3600 mm/min to location (1, 2, 3). Second, a command named G1 specifies cutting and moving operations for a cutting machine and printing and moving operations for a 3D printer. For example, command "G1 X1 Y2 Z3 F3600 E100" describes printing with the extrusion amount of filament specified by "E100" while moving to location (1, 2, 3). (The amount of filament is specified by an absolute or relative value.) In both G0 and G1, there are no constraints on direction of motion.

3. Development of Method for Procedural 3D Printing

A method of procedural 3D printing using Python as the base language is proposed. In this section, the language and the method for 3D printing using Python are described.

3.1 Python-based description method

In contrast to cutting work, the machining process in additive manufacturing is relatively intuitive, so it is useful for the designer to describe the model procedurally, as explained in Section 1. Therefore, a Python library (application programming interface (API)) is proposed for this purpose. By using a procedural abstraction function (i.e., functions and methods), which is common in programming languages, modular 3D printing, which cannot be described by G-code, can be achieved.

Python is used for describing the library in stead of a language conventionally used for machining, such as APT, for the following reasons. APT was improved and updated to have some abstraction functions, so it is not impossible to be used for 3D printing. However, instead of extending APT, it is probably better to use a languge such as Python as the base for procedural 3D printing for two reasons. First, it is considered more effective to employ a widely used language based on modern syntax and semantics. Second, a modern language such as Python has required language functions such as procedural abstraction, so it is not necessary to extend the language but only to add a library. Many other languages satisfy these conditions, but Python is selected as the base language because it is more widely and internationally used than the alternatives. Because APT cannot be executed by manufacturing machines, a file with an intermediate format called Cutter Location (CL) is obtained when executing an APT program. In the same way, in the proposed method, a G-code file is generated by executing a program written in Python.

Two libraries for procedural 3D printing were developed. They are *draw3dp.py*, which is used for 3D model generation by assembling and deforming parts, and *turtle.py*, which is used for model generation by 3D turtle graphics. The latter (published in http://www.kanadas.com/program-e/2014/08/a_python_library_for_3d_turtle.html) is a library for 3D turtle graphics. The method for drawing graphics by using turtle.py is close to that of LOGO [18], but it is used for 3D printing. The coordinate used for this library is turtle centered (i.e., similar to that used for a flight simulator), and the turtle creates the shapes of parts directly. The location of the print head is always the zero point, and its direction of motion is always forward [8][9].

In contrast, *draw3dp.py* (published in http://www.kanadas.com/program-e/2014/10/3d_printing_library_for_parts.html) is based on Cartesian coordinate. This feature is similar to its counterpart in Processing [19], MetaPost [6], or Asymptote [22]. However, there is an important difference between this library and these languages. In the former, as described below, shapes are generated by a procedural method, that is, the stacking of directed strings (lines), which is 3D-printing oriented. In this library, an object surface is thus generated by stacking strings. In contrast, in the latter, surfaces are a primitive and lines have no direction. The programming languge available in OpenSCAD [23] is designed according to the same principle. In *draw3dp.py*, a shape can therefore be directly mapped to a 3D-prining procedure, but in the languages listed above, including OpenSCAD, it cannot be directly mapped to a 3D-printing procedure. Therefore, in the case of OpenSCAD, as in other CAD tools, to enable printing models, a program called a slicer, which converts a model to a printable form, is required. Printing fails when the slicer works in a manner that diverges from the intention of the model designer, as in the case of the empty sphere described in the introduction. In contrast, the printer works according to the designer's intention when using *draw3dp.py*.

Figure 3 summarizes the APIs included in *draw3dp.py*. The APIs for assembling parts (2D and 3D shape generation) [10] are explained in Section 3.3, and the APIs for deforming parts [12] are explained in Section 3.4. The API for modulating parts (part surfaces) is explained in Section 3.5.

3.2 Part representation and generation

3D printers, including the FDM-type, stack strings of materials (these strings are called "filaments" in FDM). In *draw3dp.py*, a part is thus represented by a sequence of strings S_i, $(S_1, S_2, ..., S_n)$ [12].

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

In this expression, $Pstart_i$ denotes the start point and $Pend_i$ denotes the end point of the string. (They are assumed to be connected by a straight line.) Moreover, c_i denotes the cross section of the string (which can be replaced by a parameter of filament density), and v_i denotes the printing speed (mm/sec), that is, the velocity of the head motion. Although v_i is conceptually unnecessary, it is practically convenient. A sequence of strings represents an object (model), and it depends on the procedural generation of the object. Each string and a sequence of strings can be regarded as programs. These programs are converted to G-code before executing them.

In this representation, the direction of the filament in each location inside the part is specified. If the part is thick, not only the surface shape but also the structure and density of the filament in each internal location of the part can be specified. These parameters cannot be described by conventional CAD models or STL. The original purpose of the string-based representation presented above is to utilize the direction of the filament in 3D prinitng for expressions of objects (e.g., for an aesthetic purpose) [10][12].

Parts are treated as objects (in the sense of object-oriented design), and the class name for parts is *Trace*. For this purpose, first, the constructor, *draw3dp.Trace*, is used to generate an empty part. (See Figure 3, which also illustrates the methods described below.)

In the current version, the library contains a limited number of simple parts, such as circle, spiral, and helix; however, shapes that are not combinations of these shapes can also be described using low-level methods such as line generation. Programs that represents abstract high-level parts but

- **Constructor**: *part* = *draw3dp.Trace*(*crossSection*, *x*, *y*, *z*)
  Generate an empty *part* and specify the start point (current location) and the cross section.
- **Low-level functions**
  - *Motion*: *part.move*(*x*, *y*, *z*)
    Move linearly from the current location to (x, y, z), which is the next location to extrude filament.
  - *Line generation*: *part.draw*(*x*, *y*, *z*)
    Generate a string and add it to the *part* while moving linearly to (x, y, z).
  - *String cross section configuration*: *part.setCrossSection*(*c*) or *part.thickness*(*c*)
    Set the cross section of the string used for *part* to c from now on.
  - *Print speed configuration*: *part.setVelocity*(*v*) or *part.speed*(*v*)
    Set the print speed of the *part* to v from now on.
- **Two-dimensional part generation (parts assembly)**
  - *Circle generation*: *part.circle*(*r*, *x*, *y*, *z*)
    Add a circle with center location (x, y, z) and radius r to the *part*.
  - *Spiral generation*: *part.spiral*(*r*, *hpitch*, *x*, *y*, *z*)
    Add a spiral with center (x, y, z) and radius r to the *part*. (*hpitch* is the horizontal pitch of the string).
- **Three-dimensional part generation (parts assembly)**
  - *Helix generation*: *part.helix*(*r*, *h*, *vpitch*, *x*, *y*, *z*)
    Add a helix with center (x, y, z), radius r, and height h to *part* (*vpitch* is the vertical pitch of the string).
  - *Cylinder generation*: *part.cylinder*(*r*, *h*, *vpitch*, *hpitch*, *x*, *y*, *z*)
    Add a filled cylinder with center (x, y, z), radius r, and height h to the *part* (*vpitch* and *hpitch* are vertical and horizontal pitch).
- **Part deformation**
  - *Deformation by Cartesian coordinates*: *part.deform_xyz*(*fd*, *fc*, *fv*)
    Map the Cartesian coordinates of *part* before and after the deformation by function *fd*, and convert the cross section by function *fc* and printing speed by function *fv*.
  - *Deformation by cylinder coordinates*: *part.deform_cylinder*(*fd*, *fc*, *fv*)
    Map the cylinder coordinates of *part* before and after the deformation by function *fd*, and convert the cross section by function *fc* and printing speed by function *fv*.
- **Modulation of part (surface)**
  - *Modulation by cylinder coordinates*: *part.modulate_cylinder*(*fm*)
    Modulate *part* by function *fm* (generate texture on the part surface).
- **G-code generation**: *part.draw*()
  Generate G-code to print *part* (finalize the part).

Fig. 3 Major APIs of procedural 3D printing

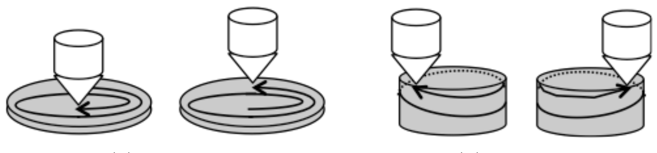


Fig. 4 Spiral and helix

that cannot be described by high-level library functions or methods can be described using low-level library functions in the same way as conventional procedural programs.

High-level parts were gradually added to the library. The addition is intended to extend the library by adding useful parts. However, in the current version, because shapes such as arcs are not supported, the programmer (designer) must describe them using the low-level APIs. Available low-level methods include *draw*, which draws a straight line to the specified location (which corresponds to G1), and *move*, which moves the head to the specified location without extruding filament (which corresponds to G0). The cross section of a string and the printing speed should also be controlled by using low-level methods.

The library includes a method called *circle*, which can draw a circle easily. Because a string is a line, a circle is approximated by lines. Because 3D printers cannot typically draw exact arcs, this approximated shape represents both a limitation of string-based expression and a limitation of current 3D printers. The number of strings that forms a circle can also be given by a parameter. However, if a circle consists of many and excessively short strings, the printing speed becomes lower than specified, and the printing work may stop or become unstable.

The library also includes a method called *spiral*, according to which single spiral can be drawn (See Figure 4(a)). A more complicated part generation method, *helix*, is also included in the library. This method generates unnested helix, that is, an empty cylinder without a bottom. It can generate a thin cylinder of any height, with no layers and hence no seams between layers.

In contrast to a single cylider generated by method *helix*, method *cylinder* generates a filled cylinder. However, seams cannot be avoided completely when printing a filled cylinder.

All the methods explained above prints objects helically or spirally from the bottom to the top. By printing helically, new filament can be supported by the filament below, which can eliminate the inter-layer seams which are often generated by 3D printing [13]. Printing a helix (or a spiral) is, therefore, considered to be the most important function in procedural 3D printing.

3.3 Parts assembly

When creating a product or a prototype using machine processing, the normal process is to generate the parts first and then to assemble them. In subtractive manufacturing, parts are assembled after the cutting process; however, when using AM, multiple parts are often generated at once. In-

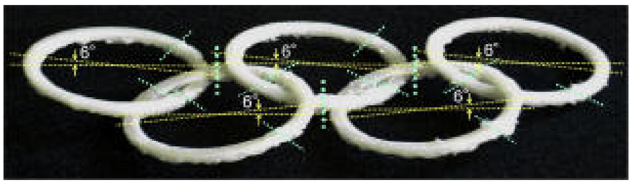


Fig. 5 Olympic symbol printed as a collection of splitted and assembled rings

stead of printing part by part, all preassembled parts are printed concurrently. Such combinations are enabled by conventional 3D design and printing methods. The proposed library may also be used for printing assembled parts, although there are currently many limitations on the combinations.

When using the proposed library, if the parts satisfy the following two conditions, they can be assembled by printing them sequentially.

- The print head is not disturbed by previously printed filaments.
- Printed filaments are supported by the print bed or previously printed filament.

Note that the processes for testing these conditions are not yet automated.

If the above conditions cannot be satisfied even by changing the printing order, the conditions are to be tested whether they are satisfied or not by dividing a part and the printing order [10]. For example, a chain of rings cannot be printed procedurally unless the rings are divided. Thus, in a previous paper [10], a 3D-shaped (chained) Olympic symbol was attempted by dividing the ring manually (because not yet automated) (Figure 5). However, because an Olympic symbol cannot be represented by a combination of parts introduced in Section 3.2, specialized parts (generation functions) were required. In addition, this Olympic symbol had to be printed without contact with the print bed, so support material was required.

By introducing functions with part-assembly procedures, the functions will become generation functions of complex parts. They represent procedurally abstracted modular structures (of programs and printed parts).

3.4 Deformation of objects

In the library *draw3dp.py*, a part generated by 3D- or 2D-generation functions can be deformed before it is printed. Deformation is introduced because only a limited number of simple shapes are manually registered in this library, and thus it is difficult to generate various shapes through assembly alone. If a part or a combination of parts can be freely deformed, especially by applying nonlinear transformations, various shapes can be relatively easily created. In conventional CAD tools, linear transformations such as translation, rotation, enlargement, and reduction, can be applied to models. The OpenSCAD lauguage, which was mentioned in Section 3.1, can also apply these transformations. However, they can be applied only to declaratively defined parts. In contrast, in *draw3dp.py*, procedurally defined parts, which are defined as a sequence of strings, can be deformed. Because a part can be regarded as a program as described before, this deformation can be regarded as a program transfomation. In computer graphics, a freer deformation is important [21][2], but deformations operate only on declaratively defined shapes. Deformation operation not only has the ability to create various shapes, but can also preserve 3D-printable [12] parts, and the printability can be preserved by using these parts.

In *draw3dp.py*, two methods, *deform_xyz* and *deform_cylinder*, are supplied for deformation, and method *draw* is used for fixing the shape of the part (to generate G-code) (see Figure 3). Two of the deformation methods have the same function; however, both methods are prepared in order to facilitate the used of Cartesian coordinates in some cases and the use of cylinder coordinates in others [12].

Method *part.deform_xyz*(*fd*, *fc*, *fv*) deforms *part* based on Cartesian coordinates. Function *fd*(x, y, z) (the first argument) maps a location (x, y, z) before the deformation to a location after the deformation. Function *fd*, thus, returns three values. Function *fc*(c, x, y, z) (the second argument) maps a cross section c before the deformation to a cross section after the deformation. Function *fv*(v, x, y, z) (the third argument) maps a printing speed (head-motion speed) v to a printing speed after the deformation. Because there is currently no way to preserve 3D printability automatically, the part designer must define appropriate functions *fc* and *fv* to preserve it.

Method *part.deform_cylinder*(*fd*, *fc*, *fv*) deforms *part* according to the cylinder coordinates. The function of this method is the same as *deform_xyz*, with the exception that the coordinates are different. Most of the currently defined parts are printed helically, so this method, which is based on cylinder coordinates, is more useful than that based on Cartesian coordinates.

These deformation methods transform the coordinates of the start and end points of strings. These transformations map straight lines, that is, strings, to straight lines, so the errors of midpoints of the strings vary. To preserve printability, the transformation function should be continuous, and enlargement and reduction should be suppressed by these transformations because enlargement or reduction causes errors, which may spoil printability (i.e., disable printing).

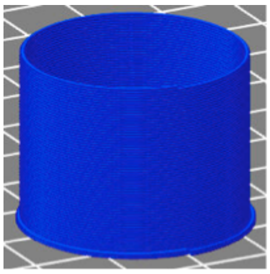
Examples of deformation are shown in Figures 6 and 7. A 3D-printing tool called Repetier Host was used to visualize the models in these figures.

Figure 6 shows a cup, which consists of a helix and thin cylinder (bottom), and a shape that is a deformation of the cup. The cup in Figure 6(a) becomes the plate shown in Figure 6(b) by applying the following deformation:

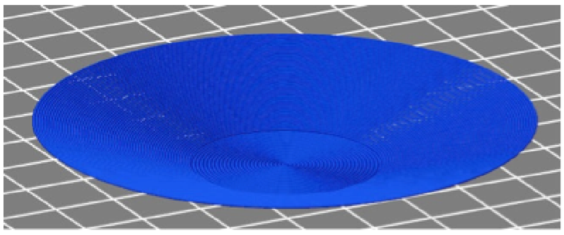
deform_cylinder(*fdd*, *fcd*, *fvd*),
where *fdd*(r, θ, z) $= (r + 1.05z, \theta, 0.3z)$,
fcd(c, r, θ, z) $= 0.96\ c$, and *fvd*(v, r, θ, z) $= v$.

Note that the size of the deformed bottom must fit the shape, that is, deformed helix.

Figure 7 shows a helix and a sphere; the latter is generated

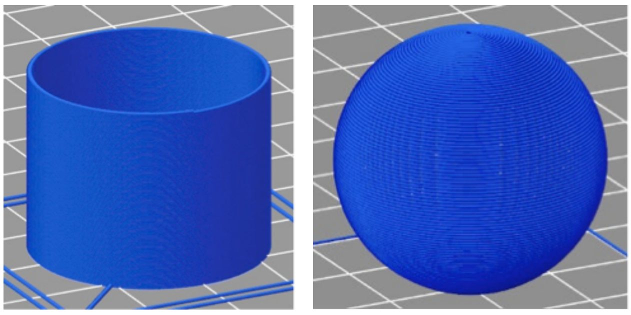


(a) Cup before deformation



(b) Plate after deformation

Fig. 6 Example of deformation from a cup



(a) Helix before deformation (b) Sphere after deromation

Fig. 7 Example of deformation from helix

by deforming the helix. Figure 7(a) shows the original helix, from which the sphere shown in Figure 7(b) is obtained by deformation. This deformation is based on the following expression, where the pitch of the filament is preserved (that is, this transformation does not enlarge or reduce the helix vertically, but it just twist around a sphere).

deform_cylinder(*fds*, *fcs*, *fvs*),
where $fds(r, \theta, z) = (Radius * sin(\pi\ z/cylinderHeight),$
$\theta, r — Radius * cos(\pi\ z/cylinderHeight)),$
$fcs(c, r, \theta, z) = 1.2\ c$, and
$fvs(v, r, \theta, z) = 1.2 * ((fr(r, \theta, z)/Radius)**2 + 0.1)\ v$,
where parameter *cylinderHeight* denotes the height of the helix before the deformation, and the length is equal to the half of the meridian length after the transformation. Several other examples were shown in a previous paper [12].

3.5 Texture map by modulation of printing

The proposed 3D-printing method can be relatively easily extended by drawing (mapping) characters, images, or textures on the surface of objects to be printed by controlling the printing process. This method generates asperity by changing the cross section of filament [14]. An application of this method is called *modulation* of printing [14]. Although changing the cross section while printing cannot generate deep asperity, it can generate shallow asperity.

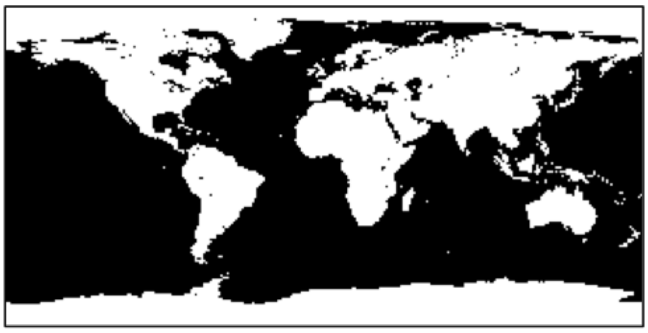


Fig. 8 Binary-valued bitmapped map

There are two ways to change the cross section of filament.

- Changing the extrusion speed of filament.
- Changing the motion speed of the print head.

The first way is more direct; however, the second way is used in the present study because it has better responsiveness. Concerning 3D printers, delay of extrusion, that is, the time from motion change of the extruder, which extrudes the filament, to the change of filament extrusion, is lengthy. It may be several seconds. The first method is thus limited by slow responsiveness. Although the print head of a 3D printer has a high level of inertia, it responds much more quickly to a motion-speed change. The second method is thus suited for generating shallow asperity. The proposed library thus uses this method. Method *modulate_cylinder* is defined for modulation (See Figure 3).

The example of modulation by a map is described below. If a plain surface is modulated by a map, the map is printed as is. In contrast, if a sphere is modulated, a globe can be printed. Figure 8 shows a binary-valued world map generated from a world map by equidistant cylindrical projection based on data from NASA, which can be obtained from the Celsia Motherload (http://www.celestiamotherlode.net/catalog/earth.php). This site contains various maps with various processing and various bitmap sizes. The size of the map shown in Figure 8 is 300 × 150. Therefore, each dot in a bitmap should be mapped to an area with 1.2° of longitude and 1.2° of latitude. Each circle of the globe should consist of 300 strings; a sphere is generated with 150 circles, and each dot on the map is mapped to a string on the globe. That is, the printing speed of each part of the string is selected from two values.

3.6 Program example

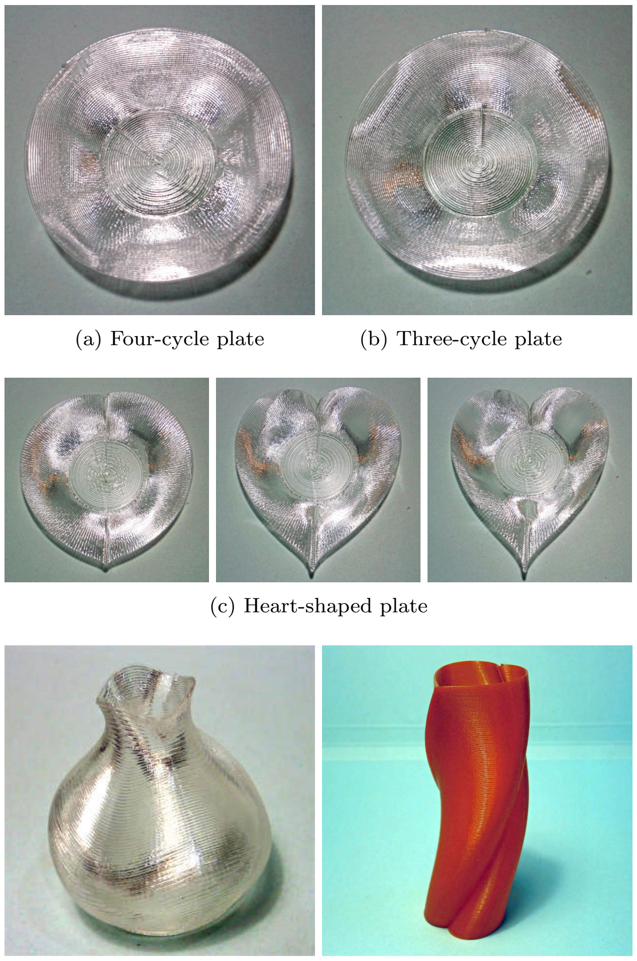
Because it may be difficult to grasp the entire model generation and printing process, the process is explained using the program below, which prints a sphere. After this program configures constants and parameters, it calls method *init* for overall initialization (but especially for initialization of the 3D printer). The program prints a so-called "skirt". A skirt is extra filament around the printing area, which is generated before the part is printed. It is printed for the sake of stabilizing the state of the print head and filament. The

```
import draw3dp
from math import sin, cos
## Constant ##
PI = 3.14159265359
## Printer parameters ##
IsABS = False     # Using PLA as the material
DefaultVelocity = 40     # mm/sec
## Printing parameters ##
x0 = 0; y0 = 0; z0 = 0.4
## Extrusion parameters ##
defaultCrossSection = 0.196 # mm^2 (Radius 0.5 mm)
FilamentDiameter = 1.75 # mm (Normally 1.75 mm or 3 mm)
## Temperature patameters ##
if IsABS:
  HeadTemperature = 235
      # ABS requires slightly highter temperature
  BedTemperature = 90     # ABS requieres heating printbed
else: # PLA
  HeadTemperature = 220
  BedTemperature = 35
      # Close to room temperature for PLA

## Initialize ##
draw3dp.init(FilamentDiameter, HeadTemperature,
      BedTemperature, DefaultVelocity)
## Generate and print skirt ##
sk = draw3dp.Trace(defaultCrossSection, 0, 0, 0.4)
skirt2(sk) # Definition if skirt2 is omitted
sk.draw(0.4)

## Generate object to be printed ##
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) # Initial printing speed configuration
obj.helix(radius, helixHeight, vpitch, x0, y0, 0)
     # Generate helix
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) # Deformation 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)     # Deformation 2
## Print ##
obj.draw()
```

Fig. 9 Program for printing empty sphere


(a) Four-cycle plate (b) Three-cycle plate

(c) Heart-shaped plate

(d) Vase

Fig. 10 Plates and pods

part to be printed is named "obj". "Obj" is initially empty but becomes a helix (by the addition of strings that form a helix by method *helix*), and it is deformed to a sphere by applying method *deform_cyliner* twice. The main deformation is performed by the first application, but it is slightly moved to z direction and the printing speed and filament-extrusion speed are adjusted by the second application. Finally, the strings are converted to G-code using method *draw*.

4. Practice of Procedural 3D Printing

By using the method described in the previous section, small plates, pods, spheres, and globes, which can be printed in 10 to 20 minutes, were manufactured. Although the products shown in this section are not intended for real-world use, readers can obtain all of these samples (http://bit.ly/1EZ4SZI or http://store.shopping.yahoo.co.jp/dasyn/).

4.1 Plates and pods generated from cup

Plates and pods with various shapes generated using the method described in Section 3.4 are shown in Figure 10. Figures 10(a) and (b) show plates [3], which are deformed from a helix as shown in Figure 6(b). Even when winding filaments mostly horizontally, no support material, which may make the printed object cloudy, is required. Figures 10(a) shows a

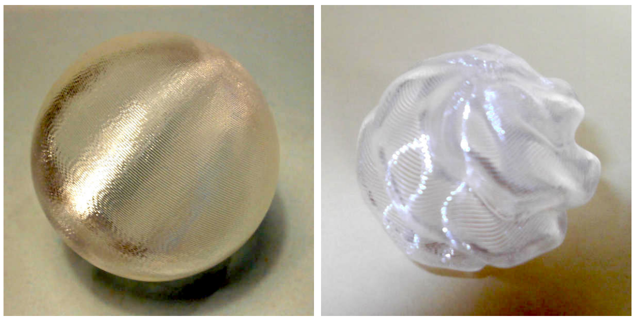


Fig. 11 Sphere and deformed sphere

four-cycle plate (i.e., with four trigonometric-function-based cyclic motions), and Figures 10(b) shows a three-cycle plate. Both specify a trigonometric function in function fd, which is an argument of method *deform_cylinder.* Because the angle of plate surface and filament density varies from location to location, the reflection of light also varies [12]. Such reflection or brightness is generated using pure and transparent polylactic acid (PLA) and by eliminating support material. No such reflection is generated by colored plastics.

Figure 10(c) shows a plate generated by using a transformation from a circular helix to a heart-shaped helix [4]. The following function, which converts a circle to a heart shape, is used.

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

This shape is based on the equation of a heart-shaped curve [26]. An appropriate range of b is 0−1.2. This transformation becomes an identity function if $b\ z = 0$ holds, and it generates a sharper heart shape if the value of $b - z$ becomes large. The helix is deformed using this function and *deform_xyz.* The shape of the horizontal cross section is a circle at the bottom, and it becomes sharper toward the top because the value of $b - z$ increases monotonically. By changing the maximum value of b, the various shapes shown in Figure 10(c) are generated. Moreover, as Figure 10(c) shows, the gradient and reflection of plate parts are varied by adding small verticall oscillations using a trigonometric function. Figure 10(d) shows two types of vases. The left vase [14] is generated by radius-direction and vertical-direction deformations using trigonometric functions. The right vase is generated by twisting the same heart-shaped helix (that is, the direction of the heart changes according to the height).

4.2 Helix-based spheres and other objects

As explained in Section 3.4, a sphere can be generated by deforming a helix. The left photo in Figure 11 shows a simple sphere generated by this method. Because it is not possible to support it at single point (i.e., the south pole) when printing it, a special technique for support is required. However, no support in the conventional meaning is used [12]. The right photo in Figure 11 shows an object generated by further deforming a sphere using a trigonometric function in the same way for the objects shown in Figures 10(a) and (b).

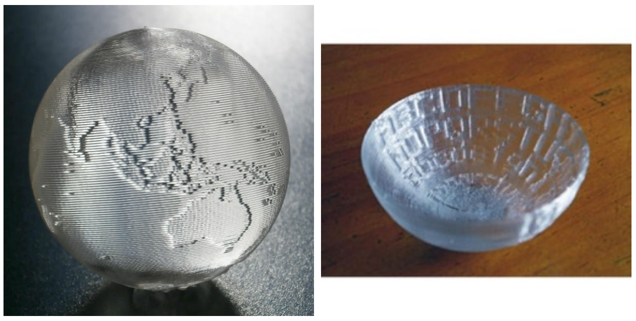


Fig. 12 Modulated sphere (globe) and bowl

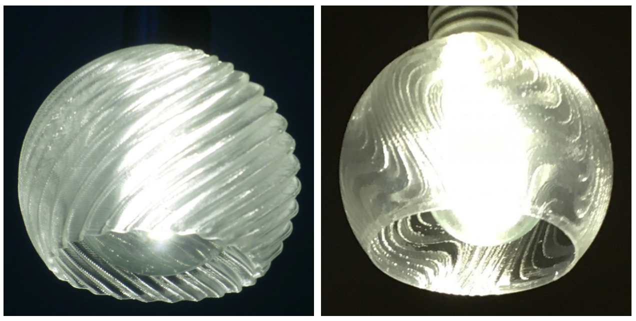


Fig. 13 Shade for small lamp

Figure 12 shows objects generated with the modulation technique that uses a bitmap, which is described in Section 3.5. The left photo shows a globe [5], which was generated by modulating a sphere by a map. The whole sphere is illuminated by only one LED placed below the sphere [15]. The right photo shows a half-sphere-shaped bowl (more precisely, a half sphere and a stand) modulated by alphabets. (This bowl was printed upside down.)

When printing a globe, the printing speed of each string is selected from two alternative values. The printing may fail if the ratio of the cross sections of sea and land is too large. An appropriate ratio is 1 to 0.6 or 1 to 0.7.

As described above, when using a bitmap of 300 × 150, each string cycle is divided into 300 short strings. However, near the poles, the strings are too short, so the number of strings is reduced. It is possible for reducing them when generating the globe model, but, because method *draw* has a function to reduce the number of connected short strings, the model representation does not to be changed.

Figure 13 shows small shades [11] for LED bulbs generated from a helix. The left photo shows a shape generated by deforming a partial sphere using trigonometric functions. The right photo shows a shape generated by modulating the same partial sphere using trigonometric functions. Most of the materials for FDM-type 3D printing become weak in the presense of heat, so they are not suited for filament lamps; however, even PLA can be used for LED shades because LEDs generate less heat. These shades are largest of the objects shown in this section (but their diameter is about 100 mm), but the printing time is around 20 minutes. The printing time is shorter than in conventional 3D printing because the shades are thin (the filament is unnested), but their relatively strong intensity protects them from beging easily broken even if dropped. (This is is the case for other objects as well.)

5. Related Work

The proposed method is characterized by modeling objects to be 3D printed using a combination of procedural parts, which are printed helically or spirally. In the examples shown in the previous section, filament is stacked seamlessly and in an aesthetically pleasing manner. Klein, et al. [16] established a method for the helical 3D printing of transparent glass. The objects printed according to this method are seamless and aesthetically pleasing. The photos in their paper shows aesthetic effects of light reflection and refraction. However, they did not mention the design method used for their work.

6. Conclusion

Current design methods for 3D printing are declarative, and procedural description by designers has not been succeeded in cutting work. However, the author suggests that procedural description is advantageous in additive manufacturing. Thus, libraries for procedural 3D printing are developed, as is a method for 3D printing, in which G-code programs are generated by this method and procedurally abstacted Python programs. Although printable shapes are restricted when using this method, layers and layer seams can be eliminated, as can the support materials required for conventional methods eliminated, and seamless and aesthetically pleasing printing is enabled by this method. The author intends to disseminate this method and the library through this paper.

References

[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] Kanada, Y.: "Three-dimensional Turtle Graphics" by 3D Printers, IPSJ Summer Programming Symposium (2014), in Japanese.

[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] Kanada, Y.: Photos of LED lamps with 3D-printed shades, Blog from Kanada, December 13, 2014 (2014), in Japanese.

[12] Kanada, Y.: Support-less Horizontal Filament-stacking by Layer-less FDM, *International Solid Free-form Fabrication Symposium 2015* (2015).

[13] Kanada, Y.: Making Plates by a 3D Printer without Support, I/O, April 2015, pp. 15–17 (2015), in Japanese.

[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] Yamamoto, N.: Heart-shaped Curves, http://www.geocities.jp/nyjp07/heart/index_heart.html, in Japanese.

m Yasusi Kanada

Yasusi Kanada was born in 1956. He received a B.E. degree in mathematical engineering from University of Tokyo in 1979 and an M.E. degree in information engineering from University of Tokyo in 1981. He has been working for Hitachi, Ltd. (Central Research Laboratory, Systems Development Laboratory, and Technology Innovation Center) since 1981. He stayed in Carnegie Mellon University from 1988 to 1990, and stayed in Tsukuba Laboratory of Real World Computing Partnership (RWCP) from 1992 to 1995. He is a part-time lecturer of Kogakuin University. He received a Ph.D (Engineering) from University of Tokyo in 1992. His work includes developments of Fortran compilers, vectorized symbolic processing and logic language processing, an emergent computation model, information extraction/search/organization, virtual-environment-based communication, policy-based networking, network virtualization, and deep-learning-based computer vision. He is a member of ACM, IEEE (Computer, ComSoc, SMC), Japan Society for Software Science and Technology, Japan Society for Artificial Intelligence, and Japan Society of Mechanical Engineers.