RESEARCH ARTICLE OPEN ACCESS

"3D Turtle Graphics" by using a 3D Printer

Yasusi Kanada
Dasyn.com, Japan

ABSTRACT
When creating shapes by using a 3D printer, usually, a static (declarative) model designed by using a 3D CAD system is translated to a CAM program and it is sent to the printer. However, widely-used FDM-type 3D printers input a dynamical (procedural) program that describes control of motions of the print head and extrusion of the filament. If the program is expressed by using a programming language or a library in a straight manner, solids can be created by a method similar to turtle graphics. An open-source library that enables "turtle 3D printing" method was described by Python and tested. Although this method currently has a problem that it cannot print in the air; however, if this problem is solved by an appropriate method, shapes drawn by 3D turtle graphics freely can be embodied by this method.
Keywords **-** 3D printer, Turtle graphics, Fused Deposition Modeling, FDM

I. INTRODUCTION

When creating solids by using a 3D printer, usually, a model designed by a 3D CAD system is horizontally sliced by using a program called a "slicer" and the resulting file is sent to the printer. Although there are various output formats for 3D design data outputted by CAD systems, the slicer usually accepts a file described by STL (Standard Triangulation Language or Stereo-Lithography), which is a declarative language. STL approximates the surface shape of the model by a collection of triangles. (It cannot express the inner structure of 3D shapes.)

Although the model outputted from a CAD system is static (declarative), CAM programs for 3D printers are dynamic (procedural) because they create products operationally. There are various types of 3D printers; however, most of cheaper printers belong to the FDM (fused deposition modeling) type. FDM-type printers extrude melted filament (plastic) from a tip of a nozzle and solidify it. (**Figure 1**). When using an FDM-type printer, the object to be printed is sliced horizontally and represented by G-code [3], which is a language for computer-aided manufacturing (CAM) and originally used for conventional machining tools such as milling machines. The model outputted from a CAD system in STL or other format is static (declarative); however, because G-code originally expresses motion of machine tools, it is intrinsically dynamic (procedural/operational). The motion of a print head and the velocity of plastic extrusion can be specified by G-code.

Two examples of G-code commands are described. First, G0 command orders simple tool motion. For example, the following command specifies motion to coordinate (0, 0, 0) by speed 3600 mm/min.

G0 X0 Y0 Z0 F3600

Second, for carving machine tools, G1 command means a motion with carving, but it means a motion with printing (i.e., with extruding filament) for 3D printers. For example, by executing the following command, the head of a 3D printer extrudes amount of filament specified by E100 while moving to (0, 0, 0).

G0 X0 Y0 Z0 F3600 E100

(The amount of filament may be a relative or absolute value according to specified printer mode.)

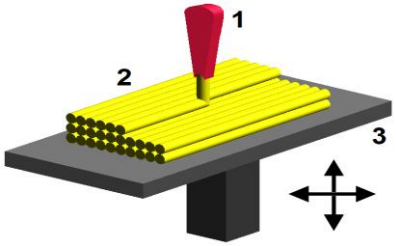


Figure 1. Principle of FDM-type 3D printers ("FDM by Zureks" by Zureks - Wikimedia Commons)

The print head of an FDM-type printer only moves to restricted directions in normal situations; however, it can actually move freely. Because an FDM-type printer usually prints sliced object layer by layer, the print head usually does not move vertically except when transition between layers. However, by using G-code directly, it can be moved to arbitrary direction. For example, if the following command is executed when the original coordinate is (*x*0, *y*0, ***z*0**), the head moves to (*x*1, *y*1, ***z*1**).

G0 X*x*1 Y*y*1 Z***z*1**

Although many 3D printers are not designed to move toward vertical direction quickly, Delta-type printers, such as Rostock MAX, are suited for this purpose.

II. RELATED WORK

Although recent researches and practices on machining are mostly based on CAD, the history of computer numerical control (CNC) started with procedural language based approach. 3D printers are considered to be machine tools for additive manufacturing (AM), and the history of computerized numerical machine tools began in subtractive manufacturing, such as lathes or milling machines. In 1950s and 1960s, programming languages for subtractive tools were widely studied. A representative language is APT [1], which was developed in MIT.

Topolabs [7] develops software for non-horizontal FDM 3D printing. They demonstrate a video on their printing process of a sloped but smooth object in YouTube [8] and show several horizontally and vertically curved "line arts" [7], which are similar to 3D turtle graphics in this paper. However, their line arts are 2D (i.e., single-layer) objects. No 3D objects, such as shown in this paper, are shown.

Turtle 3D printing is close to APT in describing tool motions and process procedurally. However, it is different in their objectives, i.e., the purpose of turtle 3D printing is to develop AM, and in the coordinates used for description. In addition, the library developed for turtle 3D printing is for a general-purpose language, i.e., Python, but APT was a language specialized for CNC (and maybe close to Logo, which was a special-purpose language). However, it can be said that turtle 3D printing is a revival of machine processing by using a procedural language, which has been neglected as a forgotten background of CAD-based technologies since 1960's.

III. TURTLE GRAPHICS

This section describes (2D) turtle graphics and 3D extensions of them.

A. 2D turtle graphics

Turtle graphics was introduced by Seymore Papert in 1960s. Papert proposed a programming language called Logo, which was designed for children. By using Logo, 2D line-art can be drawn by a trajectory of a "turtle". This is called "turtle graphics".

The basic drawing commands of turtle graphics are the following three.

- Forward *d*. This command moves the turtle forward by distance *d*.
- Turn left *a*. This command turns the turtle to the left by angle *a*° (degrees).
- Turn right *a*. This command turns the turtle to the right by angle *a*°.

By using these commands, the turtle can be moved to any location in the 2D space, and the trajectory can be displayed as shown in **Figure 2**.

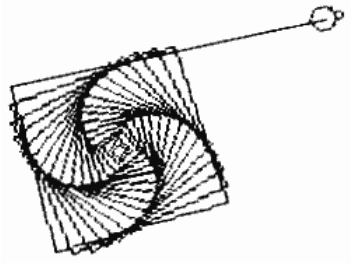


Figure 2. 2D turtle graphics

B. 3D turtle graphics

As described in the previous subsection, turtle graphics is originally two-dimensional; however, similar methods called "3D turtle graphics" were developed to draw 3D shapes (e.g., [9][6]). To extend turtle graphics to 3D, commands for moving up/down or for turning up/down must be added. Moreover, Bernd Paysan proposed "Dragon Graphics" [5], which is an extended 3D turtle graphics that can generate complex 3D shapes easily. However, all of them are graphics for displaying shapes by a 2D display. They cannot show the trajectory of turtle by 3D display.

IV. "TURTLE GRAPHICS" BY 3D PRINTING

This section discusses on a method of 3D printing, which is based on 3D turtle graphics and describes a design and implementation of this method. This method is called the *turtle 3D printing* method.

A. Outline

The semantics of 3D drawing commands and G-code are similar, so the former can be translated to the latter. Because G-code is procedural, it can execute commands similar to turtle-graphics commands. However, human beings do not usually write G-code directly, and it is not suited for human because it is similar to assembly languages. Fortunately, it is easy to translate 3D drawing commands to G-code; it is required only to translate forward command to G1 command. It can thus execute commands similar to turtle graphics.

However, because the coordinate of a print head is usually described by using a Descartes coordinate, they must be translated to turtle-direction-based coordinates. The printing procedure can be translated by a combination of the following three. First, the direction of the turtle is memorized by the G-code generation program. Second, when translating the forward command, the next coordinate is calculated by using the current coordinate and direction and set to the arguments of the G1 command. Third, when translating a "turn left" or "turn right" command, the direction in the memory is to be updated. An up/down motion can be performed in the same way,

but a coordinate system must be selected before describing the method.

B. Selection of a coordinate system

There are two alternatives for turtle coordinate: *polar coordinate* and *cylindrical coordinate*. Polar coordinate is used for flight simulators, and the direction of turtle is decided independent of the gravity direction. Unfortunately, because it is inevitable to take gravity direction into account in 3D printing, this coordinate makes guaranteeing printability difficult. That is, when using a flight simulator (and probably when flying by an airplane), it is easy to crash the airplane because it is difficult to grasp the gravity direction. A similar situation occurs in 3D printing.

When using cylindrical coordinate, the turtle always assumed to be directed horizontally. A vertical motion is described by specifying the vertical displacement, but the direction of the turtle is unchanged. Because the gravity direction is constant for the turtle, it is easier to design objects to be printed than using polar coordinate.

C. Difference between turtle 3D printing and 3D turtle graphics

As described above, by using a 3D printer, solids can be generated in a similar way as 3D turtle graphics. However, there are two differences between turtle 3D printing and 3D turtle graphics.

The first difference is that a line can be drawn at any location in the 3D virtual space by turtle graphics but a printed material (or a print bed) to support extruded filament is usually required and it is difficult to print in the air when using 3D printing.

The second difference is that the printing speed and the amount of filament extrusion, which are 3D-printing-specific parameters, must (can) be controlled in turtle 3D printing. Although thickness or some other properties of lines can be specified in turtle graphics, they can be freely chosen. In contrast in turtle 3D printing, printing speed and/or extrusion speed, which determine the thickness, must be selected properly for printing exactly and beautifully.

V. ALTERNATIVES AND LIBRARY DESIGN

Alternatives for supporting turtle 3D printing functions are described and the Python library that implements the selected functions is explained in this section.

A. Alternatives for turtle 3D printing

Turtle 3D printing function can be naturally described procedurally, so this function can naturally be part of a programming language in the same method as Logo or other turtle-graphics language-functions. In consequence, two alternative methods exist.

- To develop a special-purpose language such as Logo.
- To develop a library for an existing language.

The author selected the second alternative. The reason is as follows. When Logo was designed in 1960's, there were not many programming languages and it was not easy to extend an existing language to include turtle graphics, probably therefore, an alternative to develop a new language was selected. However, there are many extensible languages today, so there is no reason for introducing a new language for this purpose; that is, this function can more easily be used if it is introduced into existing languages. This is the reason why this function is provided as a library for an existing language.

Although it is better to support this function for various languages, Python is selected for the first language because Python is one of the most widely used language in modern languages.

B. Library for Python

The author developed turtle.py, which is a Python library for turtle 3D printing, and provided it as open-source software (http://bit.ly/1rVknxD or http://www.kanadas.com/program-e/2014/08/-a_python_library_for_3d_turtle.html). However, it is still under development; that is, it is only for a specific 3D printer (Rostock MAX).

By using functions, such as forward, left, or right, which is defined in this library, a G-code program for turtle 3D printing can be generated. This means, the library enables programming in the following way. A cylinder coordinate is used and the moving direction of the turtle, i.e., the print head, is always the front direction of the coordinate. Function forward(*dr*, *dz*) generates G-code command that proceeds the turtle forward by *dr* and lifts it up by *dz*, and function left(*da*) generates G-code command that turns the turtle left by *da* degrees.

For example, a program that stacks filaments by drawing a helix can be described as follows.

```
turtle.init(FilamentDiameter,
    HeadTemperature, BedTemperature,
    CrossSection, x0, y0, 0.4)
dz = 0.4 / 72
for j in range(0, 16):
  for i in range(0, 72):
    forward(1, dz)
    left(5)
```

turtle.init is a function for initialization and explained later. This program is similar to a 2D turtle-graphics program that draws an approximated circle by repeating advancing and turning 5° 72 times. What is different is that, instead of advancing straightforward, the turtle lifts up by dz (mm). The vertical pitch of extruded filament is assumed to be 0.4 mm. It lifts up the turtle by 0.4 mm by 72 times motions, thus the filaments are stacked well. The

diameter of the extruded filament must be slightly larger than 0.4 mm (that means the amount of extrusion must be adjusted so). It makes bonding adjacent filaments.

The initialization function, turtle.init, adjusts the following values. The first argument, FilamentDiameter, specifies the diameter of filament inputted to the printer. It is usually 1.75 mm or 3 mm depending on the printer type.

The second and third arguments, HeadTemperature and BedTemberature, specify the temperatures of the print head and the print bed. Materials such as acrylonitrile butadiene styrene (ABS) or polylactic acid (PLA) are used for filament. The head temperature is around 200–220°C for PLA and 220–240°C for ABS. The bed temperature is 25–35°C for PLA and 80–110°C for ABS. A higher bed-temperature is used for ABS because ABS shrinks when the temperature goes down and printed objects unstuck from the print bed.

The fourth argument, CrossSection, specifies the cross section of extruded filament. This argument is given for controlling the amount of filament extrusion. This parameter specifies the amount of extruded filament. An FDM-type 3D printer specifies the amount of extrusion by the length of filament provided for the printer. This specification must be rewritten when changing the diameter of filament. The amount of extrusion is specified by the cross section in the library, so only the value of the first argument of turtle.init must be changed when changing the filament diameter.

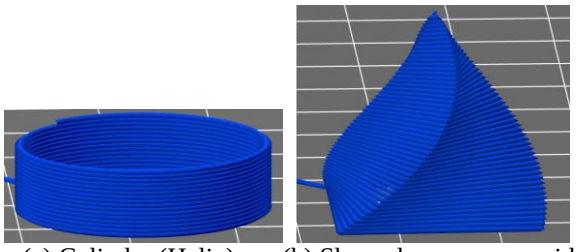
The last three arguments specify the initial location (coordinates) of the turtle. They are specified by Descartes coordinates.

VI. EXPERIMENTS

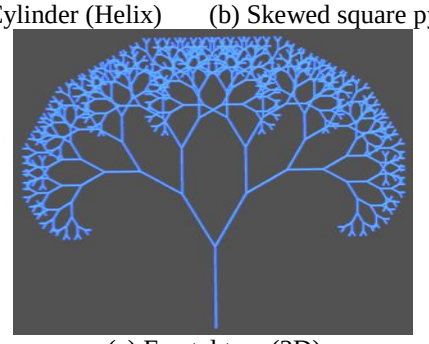
The library, turtle.py, was used for several shapes as examples. This section describes, in addition to the printing results, the development and printing processes using several free tools are described.

A. Examples

Shapes such as a cylinder (helix), a skewed square pyramid, a 2D fractal, were used for the experiments. **Figure 3** shows the shapes displayed by graphics (by a software tool called Repetier Host). Figure 3(a) shows a simplest shape, i.e., a helix or an empty cylinder. Figure 3(b) shows a skewed square pyramid, which is a shrinking pattern. A 2D fractal such as shown in Figure 3(c) does not fully utilize the functions of 3D printers, i.e., 3D shape generation. 3D fractal shapes are better for utilizing them; however, as far as the author knows, 3D fractal shapes require printing in the air, so they cannot be generated by turtle 3D printing; that is, no method for supporting filament in the air is given.



(a) Cylinder (Helix) (b) Skewed square pyramid



(c) Fractal tree (2D)

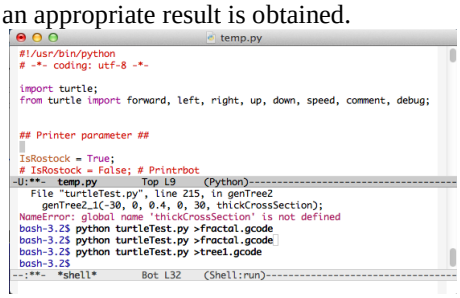
Figure 3. Examples for turtle 3D printing (visualized by Repetier Host)

B. Method

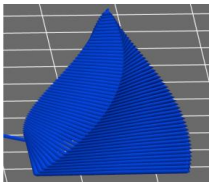
The basic procedure for turtle 3D printing is as follows (**Figure 4**).

(1) Describing the program and generating a G-code program
(2) Verifying the G-code program by graphics
(3) 3D printing (sending the G-code program to a printer)

This means, the developer first describes a Python program using turtle.py, and generates G-code program by executing the Python program. By using a visualization tool, he/she confirms the printed shape represented by the G-code program. When succeeded, he/she tries printing the program. However, the correct shape is not usually obtained by a single iteration of this process. So the process is repeated (i.e., the program is repeatedly modified) until an appropriate result is obtained.



1) Program description



2) Verification by graphics

3) 3D printing

Figure 4. Steps of turtle 3D printing

The reason why a single iteration does not usually generate a correct result is that there are many factors that spoil the result even when the visualization by the tool is succeeded. An easy factor is that, because turtle.init does not initialize the printer completely, printing may fail because of incomplete initialization; however, this problem can easily be solved. Another factor is that optimum temperature may vary by the difference of filament even when the filament material is the same, e.g., it is PLA; however, the temperature can easily be optimized. A difficult factor is that printed filament may go down or up or may cave in because the filament must be supported by previously printed filament but it fails. Examples are shown later. When it is not possible to avoid caving in, the vertical pitch may have to be changed smaller.

The above three steps are explained in detail in the following subsections.

1) Program description and G-code generation

To develop a program and to execute it, any development tools for Python can be used. The author used Emacs for describing and executing Python programs (**Figure 5**). By executing the developed program, a G-code program is generated from the standard output. It is outputted to a file.

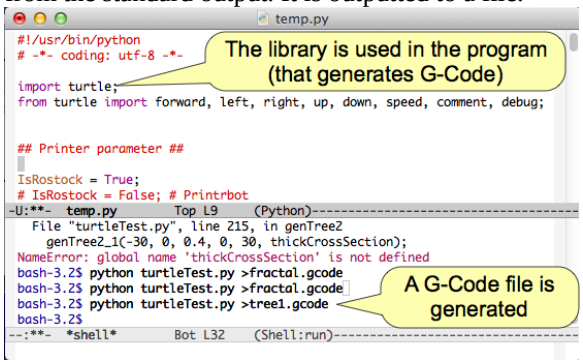


Figure 5. Program description and debugging by using an editor / development environment (Emacs example)

Figure 6 shows the main parts of the programs used for generating objects shown in Figure 3. Figure 6(a), which generates a cylinder shown in Figure 3(a), shows a straightforward program that does not contain any moving- or extrusion-speed control commands. In contrast, Figure 6(b), which generates a skewed square pyramid shown in Figure 3(b), contains motion-speed control, i.e., speed function call. These function calls reduce the motion speed to take time for cooling.

Figure 6(c), which generates a fractal tree shown in Figure 3(c), contains branching control. This means, because the pattern to be generated branches, it is necessary to return the print head explicitly to branching points and to continue printing. In the case of drawing graphics by a programming language, the execution context is automatically recovered when a function call ended, so there is no need to return explicitly. However, in the case of turtle 3D printing (by turtle.py), there is no built-in context saving and recovery mechanism, so it must be done explicitly. No automatic branching control is included in the library because the author believes context saving and recovery should be manually designed by a human programmer. In general (if the printed object is really 3D), it is very difficult to design returning and printing another branch without collision and caving in.

The program in Figure 6(c) contains branching control by using getTurtle and setTurtle functions defined in turtle.py. Function getTurtle gets the location and the azimuth of the turtle, and function setTurtle sets them. These functions must be built-in into the library because the location and the azimuth are usually hidden in the library.

```
def cylinder():
    dz = 0.4 / 72
    for j in range(0, 16):
        for i in range(0, 72):
            forward(2, dz)
            left(5)
    return
```

(a) Cylinder (Figure 3(a))

```
def skewedSquares():
    speed(10)
    dz = 0.4 / 4
    dxy = 0.4 / 4
    linelen = 30
    times = int(linelen / dxy)
    left(45)
    for j in range(0, times):
        speed(linelen + 1)
        forward(linelen, dz)
        left(90.5)
        linelen –= dxy
    return
```

(b) Skewed square pyramid (Figure 3(b))

```
def genTree_1(x0, y0, z0, azimuth0, size):
    if size >= 2:
        setTurtle(x0, y0, z0, azimuth0)
        forward(size, 0)
        (X0, Y0, Z0, Azimuth0) = getTurtle()
        size1 = 0.75 * size
        genTree_1(X0, Y0, Z0, Azimuth0 + 30, size1)
        genTree_1(X0, Y0, Z0, Azimuth0 – 30, size1)
    return
def genTree1():
    turtle.speed(20)
    genTree_1(–30, 0, 0.4, 0, 30)
    return
```

(c) Fractal tree (Figure 3(c))

Figure 6. Example programs

2) Confirmation of creation by visualization tool

A visualization tool for 3D printing can be used for confirming the shape generated by the G-code program. A tool called Repetier Host is convenient for this purpose (**Figure 7**). Repetier Host runs on Windows, Macintosh, or Linux. (However, the Macintosh version is quite old and it is not easy to install it on Linux.) Although other similar tools may exist, Repetier Host has various convenient display functions and it can be used for printing generated G-code programs. Figure 7 shows the window of the Macintosh version, which is not good for printing but may be better for visualization. The G-code program is shown at the right-side of the window.

In this step, the developer should examine the object can be printed well by using the visualization tool; however, complete check is difficult by such tools. Only weak checks such as a command syntax check are possible by conventional tools such as Repetier Host. The developer may often fail to judge printability. One reason for this failure is that the print process is dynamic but graphics display is static.

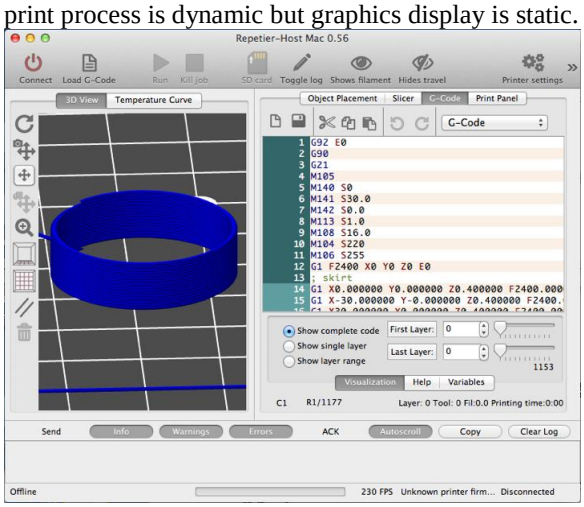


Figure 7. Confirmation of object by visualization

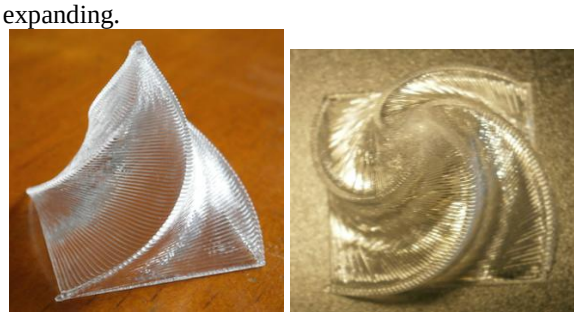
3) Printing object

Most 3D printers can print a solid represented by a G-code program, so the G-code program generated by the developed Python program is outputted to a 3D printer by using a 3D printer driver program. The author uses two driver tools for printing objects, i.e., Repetier Host and Pronterface. Repetier Host is convenient because it can also be used for visualization; however, sometimes it is not very stable, so Pronterface, which is unstable too, may be used. Printing process may sometimes stop in the middle. If it stops, it must be started from the beginning. However, restarting cost is smaller than conventional 3D printing because objects such as shown in Figure 3 takes shorter time, i.e., five minutes or so, for printing than normal printed solids that are filled with filament, which may require hours for printing.

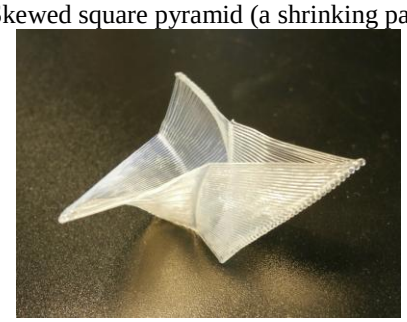
4) Printing processes and results

A printing process was recorded as a video. It can be seen in YouTube (http://youtu.be/7H5-acxQ_RE).

Examples of shapes, such as cylinders, cones, or pyramids, which were created by repeating advance and turning (distance x and angle a), are shown in **Figure 8**. If the same amount of advance and turning is repeated, it generates a cylinder. If the advance parameter, x, is changed, the pattern is shrinking or expanding.



(a) Skewed square pyramid (a shrinking pattern)



(b) Example of expanding pattern

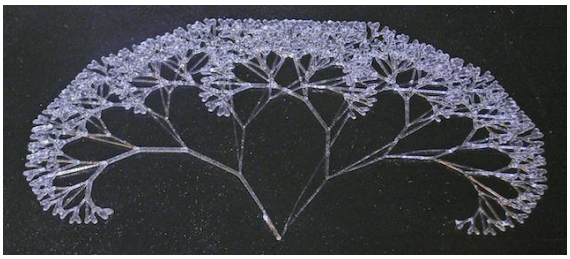
Figure 8. Printed results – patterns with rotation and shrinking/expanding

In the case of skewed square pyramid shown in Figure 8(a), the pattern is shrinking. The right photo in this figure was taken from above. This photo suggests the relationship between this shape and the 2D shape shown in Figure 2. (However, the pattern in this figure is expanding, i.e., time is reversed.)

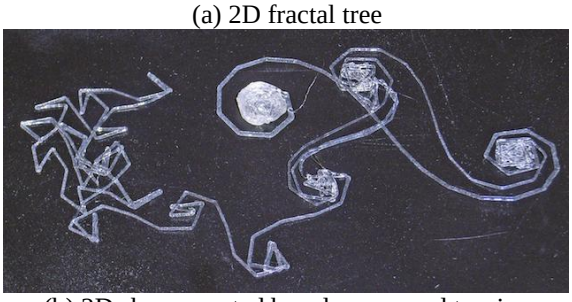
An example of expanding pattern is shown in Figure 8(b). A meticulous care is required when printing such an expanding pattern because the ground contact area is small and it can easily unstuck from the bed.

Figure 9 shows examples of other types of library usages. Figure 9(a) shows a 2D fractal tree, which was described in Section 6.1. This figure shows thin strings are generated when returning the print head. A method for avoiding them should be developed. It is not easy to avoid them automatically.

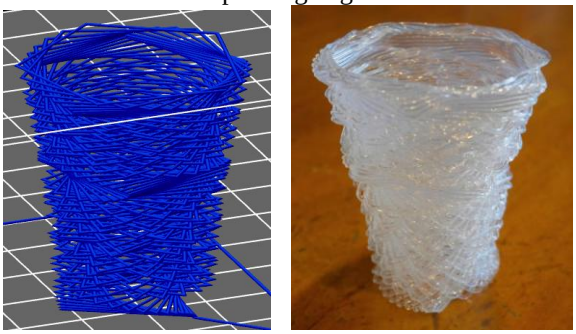
Figure 9(b) shows a pattern generated by increasing the turning angle when repeating advancing and turning. This is also a 2D pattern. Because it is an emergent pattern, it is difficult to design a supported 3D pattern.



(a) 2D fractal tree



(b) 2D shape created by advances and turning expanding angles



(c) A shape with less vertical overlapping of filaments

Figure 9. Other printed examples

Figure 9(c) shows a sparse pattern that the turtle does not stack filament completely. The left figure shows the visualized G-code and the right photo shows the printed result. Originally, the vertical pitch was designed to be 0.4 mm. However, because the pitch was smaller in the printed result, i.e, the pattern was caved in, it was redesigned to be 0.3 mm so that the macroscopic shape becomes closer to the design. However, part of the filament that is not supported by the filament below still sags. If supported area is reduced, the filament sags more; however, in the case of Figure 9(c), the print result is still close to the design.

An example of typical failure is also shown, because it is important to analyze such cases. Although Figure 9(c) can be regarded as a failed case, it is intentional. In contrast, **Figure 10** shows an example of obvious failure. It was intended to generate a skewed cylinder. The diameter of the cylinder is designed to be increasing. If the amount of increase is small, this process succeeds. However, if it exceeds the limit, it does not stack well. If tension works on the filament, the filament follows a straight line instead of an arc as shown in Figure 10 (a).

(a) Failed result

(b) Succeeded result

Figure 10. An example of failure in turtle 3D printing

VII. DISCUSSION

In the turtle 3D printing method, a cylindrical coordinate or a polar coordinate, which is fixed to the turtle, is used. However, to specify an orbit of a head of a 3D printer, in addition to this method, a method based on a Descartes coordinate, which is fixed to the outside world, can also be used. The turtle-based method is suited for emergent drawing, which is not completely designed. In contrast, the Descartes-coordinate-based method is suited for designing the shape and orbit completely. Although the patterns shown in Figure 7 have approximately continuous side surfaces, such surfaces can be more easily handled by using a Descartes coordinate. Another paper [2] develops such a method, which creates objects by combining predefined parts, deforming them, and printing them. The research directions of these two methods seem to be different. To develop turtle 3D printing further, it is necessary to loosen the constraint that it is impossible to print in the air. Actually, there are some 3D printers, e.g., "Mataerial 3D printer" [4], which can print in the air, although shapes they can create are restricted.

VIII. CONCLUSION

"Turtle 3D printing" method, which is a method for printing solids in a way similar to turtle graphics, was developed. A library for turtle 3D printing was described by Python, and used for printing some examples such as a skewed square pyramid or fractal shapes. By combining this library and G-code tools and a 3D printer, turtle 3D printing based environment for developing 3D shapes can be

realized. Although turtle-based printing process is strictly constrained by the fact that most of current 3D printers cannot print in the air, various shapes, especially emergent artistic shapes, can be generated by this method if this problem is solved by an appropriate method.

Future work includes trial of other shapes, especially shapes with smaller supported area, development of new usage of turtle.py, enhancing turtle.py, trial of polar coordinate.

REFERENCES

[1] Brown, S. A., Drayton, C. E., and Mittman, B., "*A Description of the APT Language*", *Communications of the ACM*, 6(11), 1963, 649–658.

[2] Kanada, Y., "*3D-printing of Generative Art by using Combination and Deformation of Direction-specified 3D Parts*", *4th International Conference on Additive Manufacturing and Bio-Manufacturing* (*ICAM-BM 2014*), November 2014, http://www.kanadas.com/papers-e/2014/11/3dprinting_of_generative_art_b.html.

[3] Kramer, T. R., Proctor, F. M., and Messina, E. "*The NIST RS274NGC Interpreter - Version 3*", NISTIR 6556, August 2000.

[4] Mataerial – A Radically New 3D Printing Method, http://www.mataerial.com/.

[5] Paysan, B., ""*Dragon Graphics", Forth, OpenGL and 3D-Turtle-Graphics*", August 2009, http://bernd-paysan.de/dragongraphics-eng.pdf

[6] Tipping, S., "*Cheloniidae*", December 2010.

[7] Topolabs, "Topolabs", http://www.topolabs.com/.

[8] Topolabs, "*Topolabs Revolutionary New 3D Printing Software Changes How We Think of FDM Printers*", http://www.youtube.com/watch?v=pbGUJt_hcCE.

[9] Verhoeff, T., "*3D Flying Pipe-Laying Turtle*", Wolfram Demonstrations Project, http://demonstrations.wolfram.com/3DFlyingPipeLayingTurtle/