Interpolating (and extrapolating) 3D turtle programs in Beetle Blocks

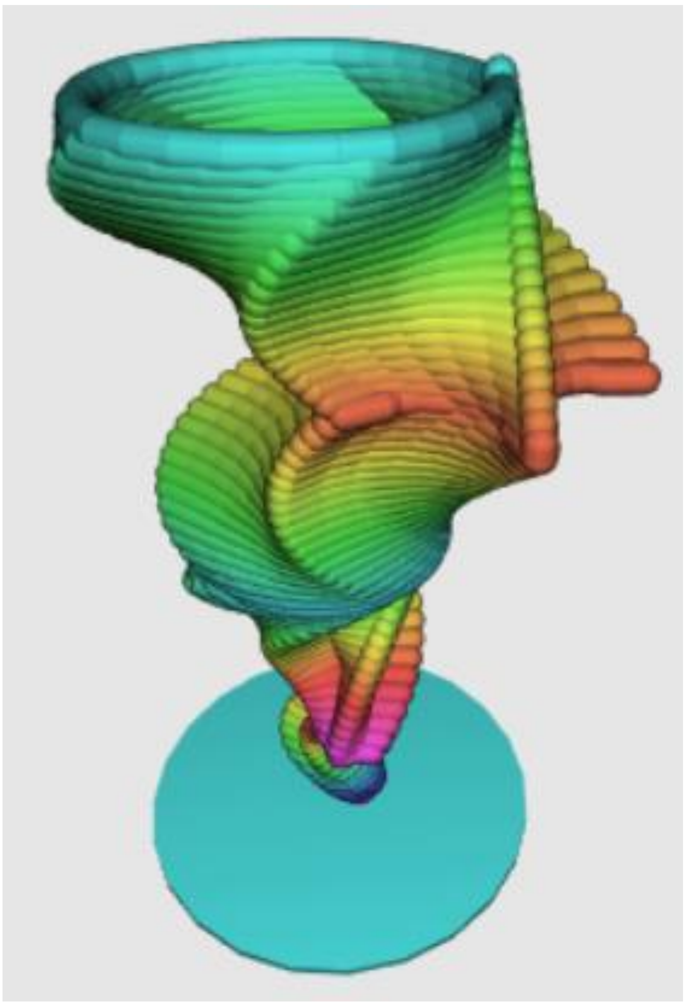
Ken Kahn, *toontalk@gmail.com*
Department of Education, University of Oxford, 15 Norham Gardens, Oxford, OX2 6PY

Abstract (Demonstration)

Turtle programs can be treated as objects to manipulate. In this demo a program takes two turtle programs as input and creates a new program that is the interpolation between the input programs. An input of .25, for example, will behave like one-fourth of the first program and three-fourths of the second. An input greater than 1 will extrapolate beyond the second program in the direction from the first program. This idea was explored in (Kahn 2007) for two-dimensional turtle programs. Here we generalise it for Beetle Blocks (Romagosa et al 2016), a 3D version of Snap! (Harvey & Mönig 2010).

Keywords

Program interpolation; Snap!; Turtle programming; 3D Turtles; Beetle Blocks; Codification;



Interpolating and extrapolating between circles, stars, and pentagons

Interpolating 2D Logo turtle programs

(Kahn 2007) describes a program interpolator that can create a new program from two Logo programs that are defined using FORWARD, RIGHT, REPEAT, SETPENCOLOR, PENUP, and PENDOWN. As the simplest example consider two programs that draw different length lines:

```
to short
forward 40
end
```

```
to long
forward 100
end
```

The interpolated program is:

```
to short_to_long :x
forward interpolate :x 40 100
end
```

```
to interpolate :x :a :b
output :a + :x * (:b - :a)
end
```

SHORT_TO_LONG 0 behaves just as SHORT, SHORT_TO_LONG 1 behaves as LONG, SHORT_TO_LONG .5 averages them, and SHORT_TO_LONG 2 extrapolates beyond LONG starting from SHORT.

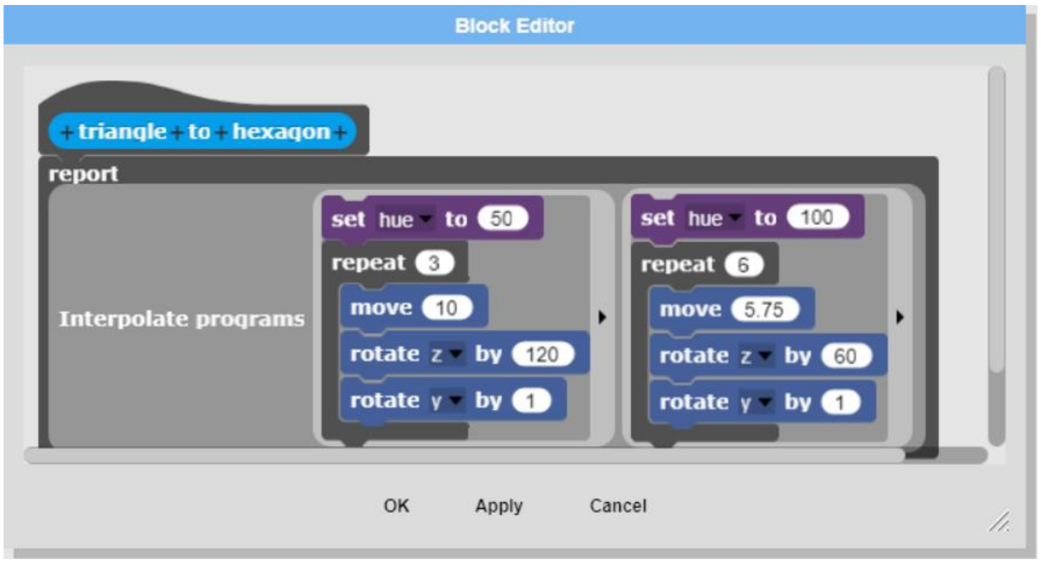
The difficult step is canonicalising input programs that repeat a sequence of turtle commands a different number of times. (Kahn 2007) explains this in detail.

Moving to 3D

Beetle Blocks (Romagosa et al 2016) is a well-designed 3D version of Snap! (Harvey & Mönig 2010). Before one can begin to create interpolations of Snap! or Beetle Blocks programs we need a way to treat a block program as a data structure that can be manipulated programmatically. Fortunately, Snap! has a “codification” feature (Ball et all 2015; Harvey & Mönig 2018). Codification has been used to define how Snap! blocks can be translated to Python, C, Smalltalk, JavaScript, or other languages. We used this feature to translate Snap! blocks into JSON strings that are then converted into Snap! lists.

The problem of programming a Snap! program that constructs another program (the interpolation program) was resolved by defining the constructed program as a list of closures that can be run to execute a sequence of commands. The values or expressions in corresponding commands in the two input programs are handled as variables closed over by functions.

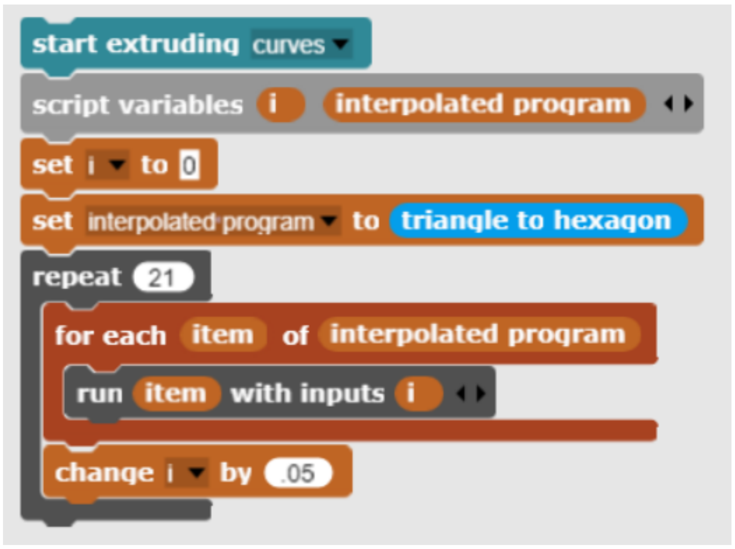
FORWARD (or MOVE as it is called in Beetle Blocks) is treated in a similar manner to how the 2D turtle interpolator worked. RIGHT (or ROTATE as it is called in Beetle Blocks) has an additional argument that specifies whether the rotation is around the x, y, or z axes. As long as both programs rotate in the same dimension in the corresponding program locations it is straight-forward to generalise to 3D. To make a more generic interpolator other Beetle Block commands are also supported including “go to x: y: z:” and a list of “set” commands that change coordinates, hue, saturation, lightness, and opacity. Beetle Block command for extruding curves and lines are also handled. Furthermore, arithmetic expressions involving addition, subtraction, multiplication, and division are supported. Support for user reporters can be easily added.



```
+triangle+to+hexagon+
report
  Interpolate programs
    set hue to 50
    repeat 3
      move 10
      rotate z by 120
      rotate y by 1

    set hue to 100
    repeat 6
      move 5.75
      rotate z by 60
      rotate y by 1
```

Figure 1 - A simple example of interpolating a yellow triangle to green hexagon



```
start extruding curves
script variables i interpolated program
set i to 0
set interpolated program to triangle to hexagon
repeat 21
  for each item of interpolated program
    run item with inputs i
  change i by .05
```

Figure 2 - A program to run the triangle to hexagon interpolation program with 21 values from 0 to 1

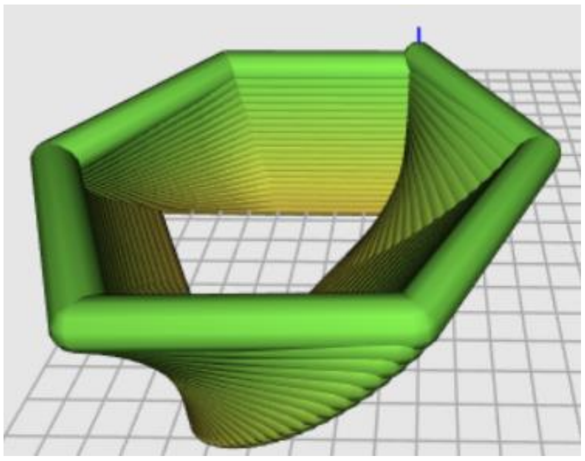


Figure 3 - the result of running 21 interpolations between the triangle and hexagon

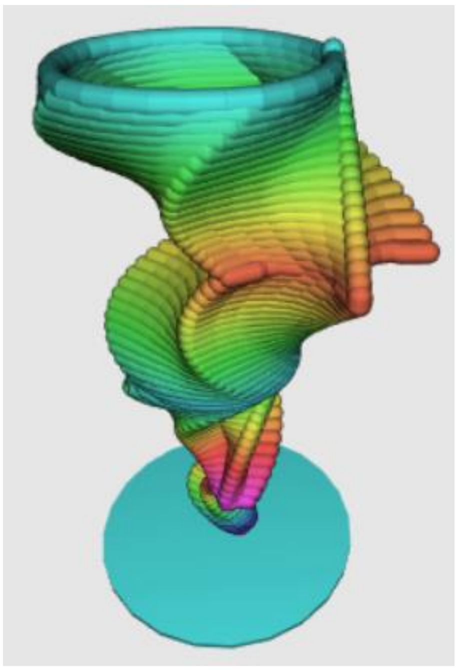


Figure 4 - Interpolating and extrapolating between circles, stars, and pentagons

A very nice feature of Beetle Blocks is that the output of 3D turtle programs can be created on 3D printers. Figure 4, for example, is an attempt to make a vase. This program (and the interpolation program generator) can be found at https://tinyurl.com/circle-star-pentagon.

Conclusion

A turtle program is more than a shape. A circle, for example, can be drawn clockwise, counter-clockwise, or multiple times. The consequences of how a shape is drawn can result in dramatically different interpolations and extrapolations. Students tinkering with program interpolation and extrapolation are entering a mathematically and computationally rich area. And the creative and aesthetic possibilities of interpolating and extrapolating between turtle programs are many.

References

Ball, M., Mock, L., McKinsey, J. Machardy, Z., Garcia, D., Titterton, N., Harvey, B. (2015) Oh, Snap! Enabling and Encouraging Success in CS1. In: SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education. pp 691-691.

Harvey, B., Mönig, J., (2010) Bringing “No Ceiling” to Scratch: Can One Language Serve Kids and Computer Scientists? In Proceedings: Constructionism, Paris, France.

Harvey, B., Mönig, J., (2018) Snap! 4.1. Reference Manual. https://snap.berkeley.edu/SnapManual.pdf.

Kahn, K (2007) A Program to Interpolate (And Extrapolate) Between Turtle Programs. International Journal of Computers for Mathematical Learning. December. Volume 12. Issue 3. pp 255–262.

Romagosa, B., Rosenbaum, E., Koschitz , D. (2016) From the Turtle to the Beetle - The Beetle Blocks programming environment. http://hdl.handle.net/10609/52807. Universitat Oberta de Catalunya.