Fundamental Concepts of 3D Turtle Geometry

Manuel Riel, *manuel.riel@fu-berlin.de*
Didactics of Computer Science, University of Potsdam, Germany

Ralf Romeike, *ralf.romeike@fu-berlin.de*
Computing Education Research Group, Free University Berlin, Germany

Abstract

Seymour Papert's turtle in Logo has launched a revolution in programming enabling students, even at a young age, to create their own art using elementary geometry. Recent developments allow for creating turtle art in 3D: Visual, block-based programming environments, such as Beetle Blocks, in explicit tradition of Papert's turtle graphics, help students to design their own 3D models and convert them from the virtual to the physical world using 3D printing technology. However, reported experiences from previous teaching attempts involving 3D turtle geometry faced a significant challenge: in order to "draw" such beautiful three-dimensional objects, complicated advanced mathematical methods seemed inevitable, even for creating basic figures. This encouraged us to explore in detail, how artistic figures can be created in 3D using intuitive geometric knowledge suitable for novice programmers. Therefore, as a first step to understand what makes creating 3D turtle art complicated, we identify the underlying conceptual difference between the two-dimensional and three-dimensional space for turtle geometry: the existence of two entirely different coordinate systems. Building on this, we discuss in depth, how traditional turtle geometry can be applied for creating 3D objects, and explain a universal approach to creating a wide range of shapes in 3D turtle geometry. Subsequently, we present our experiences from the classroom gained in our teaching series based on the discussed concepts. In conclusion, when taught in an intuitive way, 3D turtle geometry offers another motivating setting for fostering creativity.

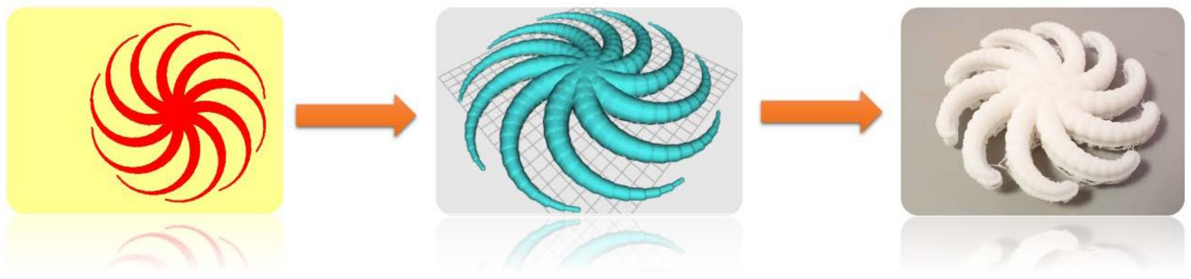


Figure 1. Classic 2D turtle art transformed from 2D to 3D and printed out as physical object. [contains image from turtleart.org[1]]

Keywords

3D printing, 3D model, computing education, programming introduction, Beetle Blocks, turtle art

[1] https://turtleart.org/gallery/imagepage.html?50

1. Introduction

Within the last years, fab labs and makerspaces have shown that modern fabrication processes, which reach from 3D modeling objects in CAD environments to manufacturing them using additive or subtractive technologies, motivate individuals to create small items for personal use. An aspect of this fascinating maker culture can be brought to the classroom in CS education when using 3D printers to build previously self-designed 3D models.

However, professional CAD environments used in fab labs are quite hard to understand for students. Instead, this technique can be applied in the classroom with one of the following approaches:

1) An older, more "traditional" way which utilizes easy-to-use CAD environments, e.g. TinkerCAD (Buehler et al., 2014) or Google SketchUp (Lutz, 2013). Students use pre-defined 2D and 3D shapes, such as circles, cubes, and spheres, to create new objects by combining the given shapes.

2) Constructionist driven *programming* environments which directly follow the ideas of Logo – but add the third dimension: In Beetle Blocks, a beetle – like Logo's turtle – is controlled by students with simple programming commands. On instruction, it extrudes shapes along the travelled path, hence creating Turtle Art in 3D (Romagosa et al., 2016).

Most educators around the world prefer the latter approach, because it inherently supports the constructionist geometric idea for novice programmers. Furthermore, it seems more intuitive than the usual abstract (de-)composition of objects in conventional CAD environments.

Offering great potential for constructionist approaches, 3D printing technology allows students to convert their 3D models, which have been created in environments like Beetle Blocks, from the virtual to the physical world. However, in the light of experiences with existing teaching attempts for 3D printing in CS education, described by Kastl et al. (2017), one major challenge becomes apparent: Even in higher school classes students faced difficulties due to complicated mathematical methods used for creating even simple figures.

Encouraged by these reports, in this paper we explore in depth, how the intuitive concepts of traditional turtle geometry can be brought to 3D. Therefore, the paper is structured as follows: In section 2 we analyze the underlying conceptual difference between the two-dimensional and three-dimensional space for turtle geometry, which causes the challenge mentioned above: The existence of two different, three-dimensional coordinate systems with two different reference points. We explain further, how this affects the process of constructing 3D objects. Building on this, we discuss in section 3, which concepts of traditional turtle geometry are already appropriate for creating 3D objects or how strategies can be adapted in 3D – always considering novice programmers keeping the mathematical background intuitive. Additionally, we present an easily accessible approach, which supports creating 3D turtle geometry and can be used in a wide range of projects. Based on the concepts discussed, we designed a teaching series for novice programmers and share our experiences from the classroom presenting students' final projects in section 4: Supported by fundamental concepts in 3D turtle geometry students can create their own 3D objects.

2. Conceptual Differences in 2D vs. 3D Turtle Geometry

Before we can discuss fundamental concepts of turtle art in 3D, further investigation of the geometrical situation in 3D is needed, which differs more from 2D than expected: Not only a third axis is added to the 2D coordinate system, but also *two* different three-dimensional coordinate systems become apparent (see fig. 2).

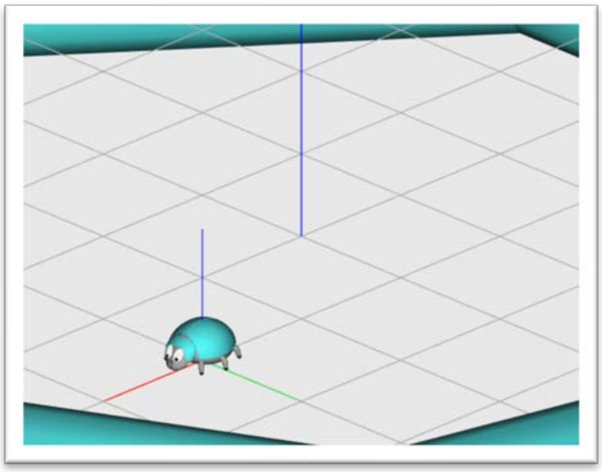


Figure 2. The two coordinate systems in 3D turtle geometry, easily recognizable by the particular blue z-axis.

In this section we explain, why two different coordinate systems are necessary for 3D turtle geometry and how they affect strategies for creating three-dimensional objects. Furthermore, we discuss strategies for helping students to overcome issues when entering the third dimension.

2.1 The Need for Two Coordinate Systems

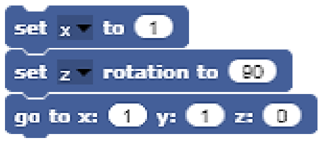
The existence of two different reference systems is a result of two intuitive and complementary requirements for a 3D design environment. As a consequence, both coordinate systems differ in their reference point as well as in their movability:

1) The world's absolute coordinate system starts in the world's origin and thus is fixed there. It allows to name the position of the beetle or other objects in the "traditional" way; e. g. (1, 4, 3) for the point in the world with coordinates x=1, y=4 and z=3.

2) The beetle's coordinate system arises in the beetle itself and hence is dynamic: It moves together with the beetle's position and supports programming from the beetle's point of view – which makes 3D turtle graphics possible. This coordinate system is primarily important for turning and moving the beetle within the three-dimensional space.

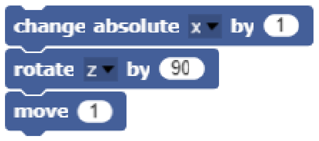
By using these two coordinate systems, which both seem intuitive at a first glance, inconsistencies can result from mixing up their specific conceptions, which will be addressed in the following section. An additional challenge arises from the fact that Beetle Blocks lacks an explicit separation of their dedicated instructions. Since there is no extra category (like the existing "Motion" or "Control") it is important to keep the different coordinate systems in mind while programming (see fig. 3).

commands from the view of the world's coordinate system



```
set x to 1
set z rotation to 90
go to x: 1 y: 1 z: 0
```

equivalent instructions from the beetle's point of view



```
change absolute x by 1
rotate z by 90
move 1
```

Figure 3. Exemplary comparison of instructions from different point of views.

2.2 Two Coordinate Systems – Two Ways for Creating 3D Objects

The two coordinate systems allow 3D objects to be constructed in two common, but quite different ways, which we will explain in the following: One possibility is to move the beetle like it is "going up stairs" by rotating the beetle around of one of its horizontal axes, i.e. the x- or y-axis. The other option is to create two-dimensional base areas and layering them on top of each other (see fig. 4).

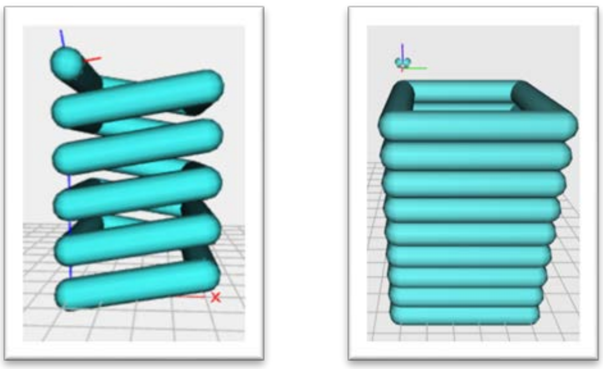
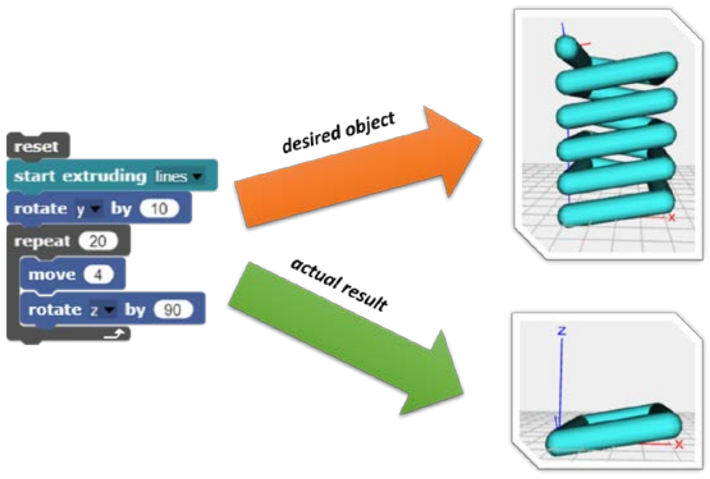


Figure 4. The two ways of creating 3D objects:
"Going up stairs" on the left, layering base areas on the right.

However, even though the first way seems to be the intuitive and turtle-like approach, it poses a significant challenge to the students due to a likely confusion of the coordinate systems, which will be discussed in detail in the next section. In order to overcome such issues, we suggest initially using the second approach, which we will illustrate afterwards.

2.2.1 "Going Up Stairs"

We explain, why "going up stairs" in 3D is complicated, by illustrating an example from the class-room: A student intended to program a spiral, which consists of "shifted" squares, but the attempt failed due to a misconception regarding the two coordinate systems. It is not sufficient to "rotate the beetle upwards" ("rotate y by 10"), as figure 4 illustrates.

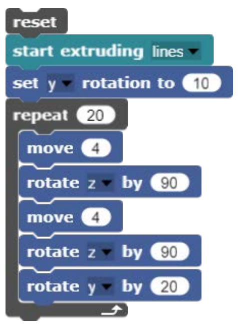


```
reset
start extruding lines
rotate y by 10
repeat 20
    move 4
    rotate z by 90
```

Figure 5. Misconception resulting from the existence of two reference systems

This misconception, which can quickly be adopted by novice programmers, is that the rotational situation of the beetle is not only changed by the block "rotate y by 10", but also by "rotate z by 90" - in relation to its own dynamic coordinate system, not the world’s fixed reference system.

Recurring to the spiral in figure 4: When implemented correctly, a *repeated* "rotation" around the y-axis (from beetle's perspective) is necessary[2] in order to achieve the desired result (see fig. 5). This makes the program quite complicated.



```
reset
start extruding lines
set y rotation to 10
repeat 20
    move 4
    rotate z by 90
    move 4
    rotate z by 90
    rotate y by 20
```

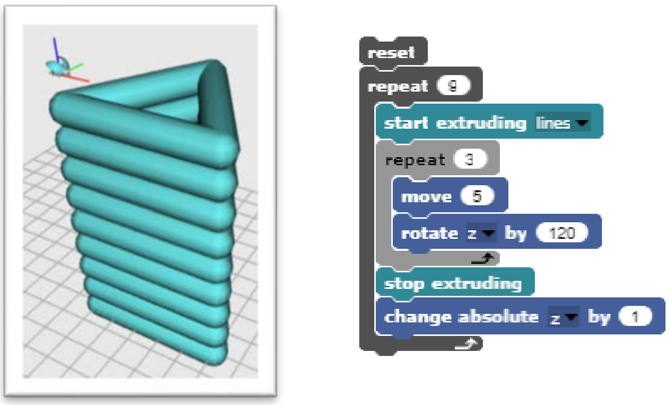
Figure 6. Correct implementation of the spiral.

Generally speaking, rotations around different axes are not even commutative – which makes the geometrical concept even harder to understand. On top of that, horizontal rotations as a strategy for creating three-dimensional objects can hardly be applied systematically to a variety of objects. In conclusion, “going up stairs” as well as horizontal rotations in general most likely should be avoided in CS introductory lessons.

2.2.2 Layering Base Areas

In contrast to “going up stairs”, layering up base areas on top of each other seems to be an all intuitive approach for creating three-dimensional objects. At a first glance, it looks like traditional turtle art in 2D can just be layered on top of each other for creating three dimensional figures (see fig. 7).

[2] Note that Beetle Blocks developer, Bernat Romagosa, has made an optional library for movements relative to the beetle’s orientation available in Beetle Blocks’ forums: This library allows the spiral in fig. 5 to be implemented quickly, but still shares the other issues from two-axes-rotations.



```
reset
repeat 9
    start extruding lines
    repeat 3
        move 5
        rotate z by 120
    stop extruding
    change absolute z by 1
```

Figure 7. Triangles layered up to create a tower.

However, the situation is more complicated than that: It is essential to keep the vertical axis of the desired 3D object in mind, which goes through the starting point of the beetle when generating the base area. So, while in 2D the turtle's starting point for figures usually does not matter, in 3D the beetle's starting point – correspondingly the starting point for its dynamic coordinate system – will define the vertical axis of the resulting 3D object. We illustrate this phenomenon by using the example of a vase consisting of equilateral triangles in increasing sizes (see fig. 8), similar to the vases published by Kastl et al. (2017).

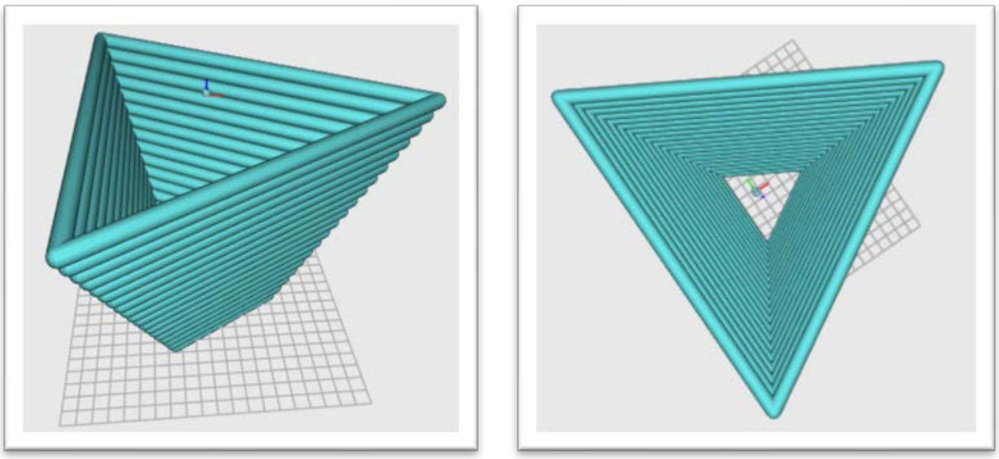
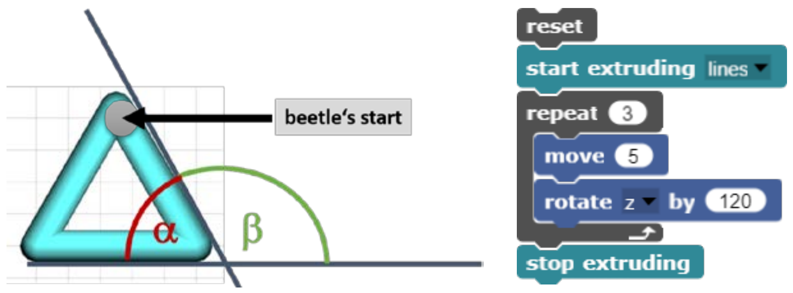


Figure 8. Vase-like object consisting of equilateral triangles.

First, we create – in the typical Logo way – an equilateral triangle as base area for the vase (see fig. 9). The beetle's start position is – as usual in Logo – in one corner point of the polygon.



```
reset
start extruding lines
repeat 3
    move 5
    rotate z by 120
stop extruding
```

Figure 9. Creating a triangle with side length 5 by rotating the beetle around β = 120°: Just like in two-dimensional Logo.

In the next step, we layer the triangles without varying their size – this will result in the tower already shown in figure 7 – looking good so far.

However, when modifying the program in the final step to continuously increase the triangles' side lengths, a leaning tower will emerge instead of the intended vase (see fig. 10).

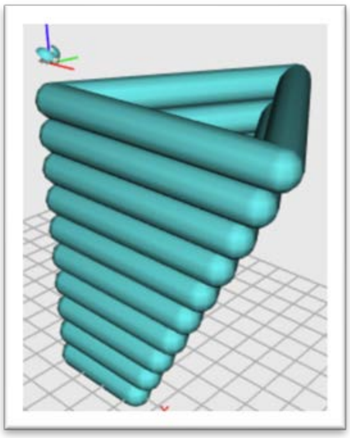


Figure 10. Resulting leaning tower by layering traditional constructed triangles in increasing size.

As explained, this is another effect of the two coordinate systems, particularly the vertical z-axis of the beetle's dynamic coordinate system, and leads us to *the* main challenge in 3D turtle geometry: In order to create the desired vase – or generally speaking: in order to create any 3D object symmetrical to its vertical axis – it is essential to start the creation process of their base area in its center point. In existing teaching attempts this is achieved by advanced mathematical methods, like calculating the coordinates of corners using the Pythagorean theorem (Kastl et al., 2017).

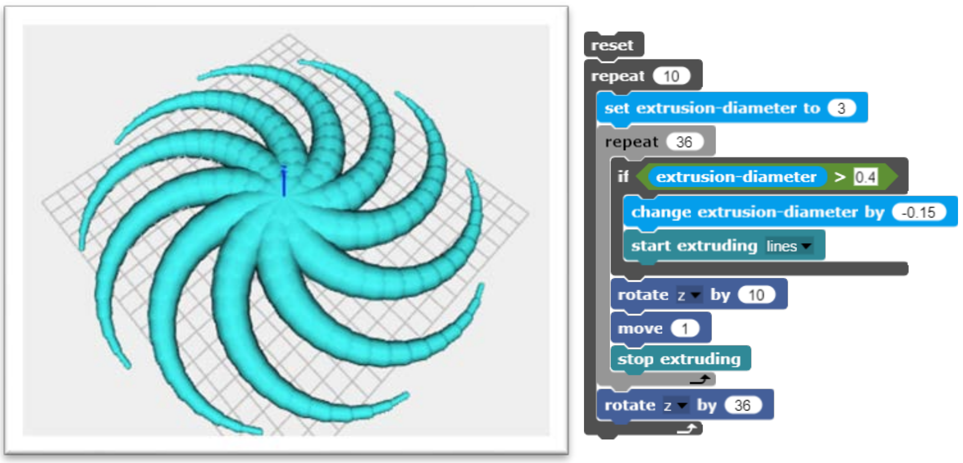
Following the proposal of Kastl et al. (2017) to find simpler mathematical methods for 3D objects like the vase above, we explore possible solutions in the subsequent section.

3. Exploring Concepts of Turtle Geometry in 3D

Our aim is to systematize fundamental concepts of 3D turtle geometry, which support students in creating their own 3D objects: These concepts should be easily accessible even for novice programmers and be (re-)usable in a large variety of different projects. Thus, we explore in this section, which proven concepts of traditional turtle art in 2D can also be applied in 3D, or how they can be adapted to 3D. Subsequently, we put forward a universal approach addressing the vertical axis problem.

3.1 Traditional Turtle Art for flat objects in 3D

While new approaches to base areas are required when creating actual three-dimensional objects, traditional turtle geometry can be applied in 3D without further modifications and achieve appealing results: For example, the swirl from the logo-inspired Turtle Art Gallery[3] can also be implemented in an appealing three-dimensional way (see fig. 11) and even looks good printed out (see fig. 1).



```
reset
repeat 10
    set extrusion-diameter to 3
    repeat 36
        if extrusion-diameter > 0.4
            change extrusion-diameter by -0.15
            start extruding lines
        rotate z by 10
        move 1
        stop extruding
    rotate z by 36
```

Figure 11. Screenshot and code of Turtle Art Gallery's swirl brought to 3D.

While the swirl is a specific example, a broad variety of flat objects can be created by just rotating polygons (see fig. 12), which have been constructed the traditional way (as seen in fig. 9).

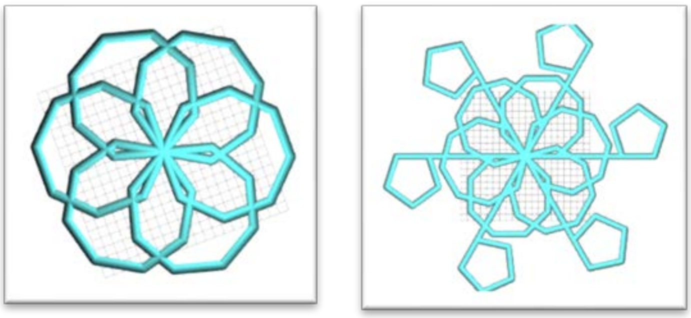


Figure 12. 2D turtle art recreated in Beetle Blocks.

In conclusion, Logo's two-dimensional turtle art can be recreated in 3D programming environments like Beetle Blocks and thus its concepts can also be taught in introductory lessons – as an approach which has been proven effective for decades.

[3] https://turtleart.org/gallery/index.html

3.2 Regular Polygons as base areas for creating actual three-dimensional Objects

What is fascinating about 3D printing, however, are not flat structures consisting of base areas, but more the creation of real three-dimensional objects. We focus on regular polygons as base area for actual 3D objects, because they represent symmetrical structures, which are considered aesthetic bearing in mind Papert's "poly pictures" (1972). In 3D turtle geometry the main challenge in creating base areas is to start in their center point to obtain symmetrical objects (as explained in section 2.2.2). In the following we address that challenge regarding polygons constructed the traditional way and subsequently suggest a universal approach to constructing base areas.

3.2.1 Tricks for Polygons constructed traditionally

Regular polygons constructed the usual way will result in "leaning towers": For example, a simple layering (and continuous shrinking) of squares leads to a leaning tower (see fig. 13), because the vertical axis runs through the beetle's starting point, i.e. a corner, when creating the base area.

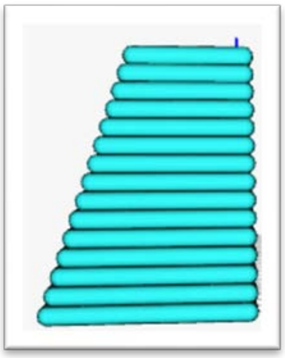


Figure 13. Leaning tower of piled squares with vertical axis through corner of the base area.

However, two intuitive tricks are available for *some* regular polygons, which have been constructed the traditional way: The first trick finds the center point of squares and hexagons, the second one turns leaning towers with a triangular or squared base area into symmetrical vases.

1st trick: Moving the beetle to the center point: For squares and hexagons it is possible to determine the base area's center with elementary geometry – using the (half) side length and the known inner angle of 90° or 60° respectively. The resulting tower meets the expectations (see fig. 14).

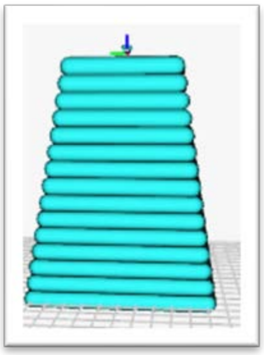
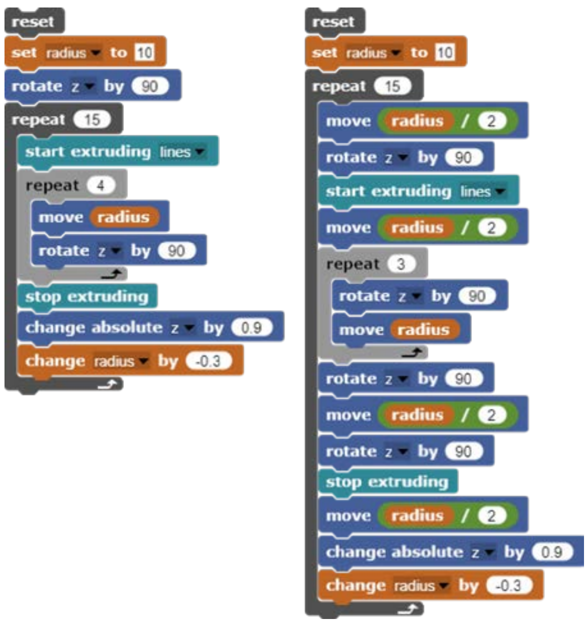


Figure 14. Tower with vertical axis through circumferential center of the quadratic footprint

We compare the source code of the leaning tower and its symmetrical counterpart achieved by this trick in figure 15: The simple layering on top of each other, which is basically realized with the command "change absolute z by 1", is lengthened with further instructions. This makes the formerly easy code much more complicated to read and modify.



```
reset
set radius to 10
rotate z by 90
repeat 15
    start extruding lines
    repeat 4
        move radius
        rotate z by 90
    stop extruding
    change absolute z by 0.9
    change radius by -0.3
```

```
reset
set radius to 10
repeat 15
    move (radius / 2)
    rotate z by 90
    start extruding lines
    move (radius / 2)
    repeat 3
        rotate z by 90
        move radius
    rotate z by 90
    move (radius / 2)
    rotate z by 90
    stop extruding
    move (radius / 2)
    change absolute z by 0.9
    change radius by -0.3
```

Figure 15. Source code of the leaning tower (left) and its symmetrical counterpart, which has been constructed finding the center with simple geometry (right)

2nd trick: Rotation of the *whole* object*:* In order to create a symmetrical vase, leaning towers as a whole can be rotated four times by 90° for squared base areas (respectively six times by 60° for triangular base areas) so that a larger tower with inner bars is formed (see fig. 16). Even though only small changes in program code are necessary, these bars, however, can interfere in a lot of objects, e. g. when designing vases.

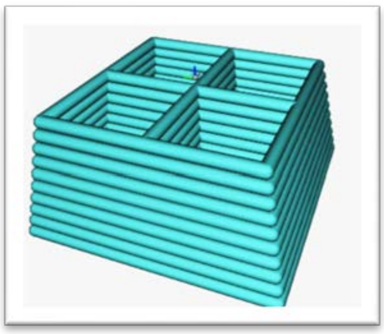
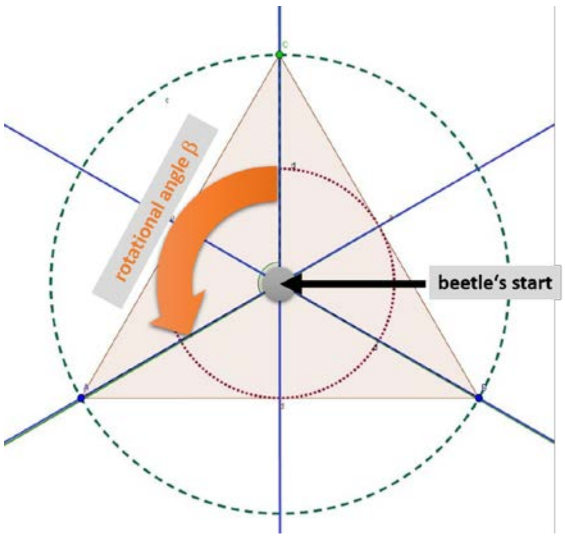


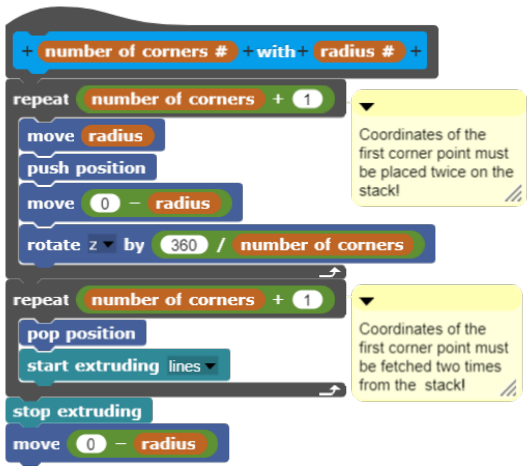
Figure 16. Symmetrical tower created by four-time-rotation of a leaning tower showing inner bars.

3.2.2 Creating Polygons using a Stack Register

For generating base areas with the beetle’s start position in their center point, we suggest the algorithm described in the following: The algorithm’s idea can be understood by novice programmers and supports the creation of many different 3D objects. The original program code and visualization of the geometric situation is as an example shown for a triangle in figure 17 – nevertheless the idea is easily adapted for *all* regular polygons.

Beetle Blocks offers a hidden, not (yet) visualized stack register for the position of the beetle using the instructions "push position" and "pop position" in the "Motion" category. The beetle’s starting point in this alternative method is no longer a corner of the polygon to be implemented, but the (circumferential) center point of the constructed regular polygon. From this point, the beetle moves to a corner and pushes the current position onto the stack. Then it moves backwards to the center of the polygon again, rotates around the angle. The last steps are being repeated for the remaining corners. Eventually, the corner position saved first has to be placed on the stack again. Now the beetle can "beam" itself to the location of a corner by using "pop position" and then connect the corners by extrusion and calling "pop position" again. Finally, the beetle has to stop the extrusion and return to the center of the circle. The latter can be achieved easily, since it is located on the circumferential line and therefore the beetle only has to "go backwards" again.





```
+ number of corners # + with + radius # +
repeat (number of corners + 1)
    move radius
    push position
    move (0 − radius)
    rotate z by (360 / number of corners)
repeat (number of corners + 1)
    pop position
    start extruding lines
stop extruding
move (0 − radius)
```

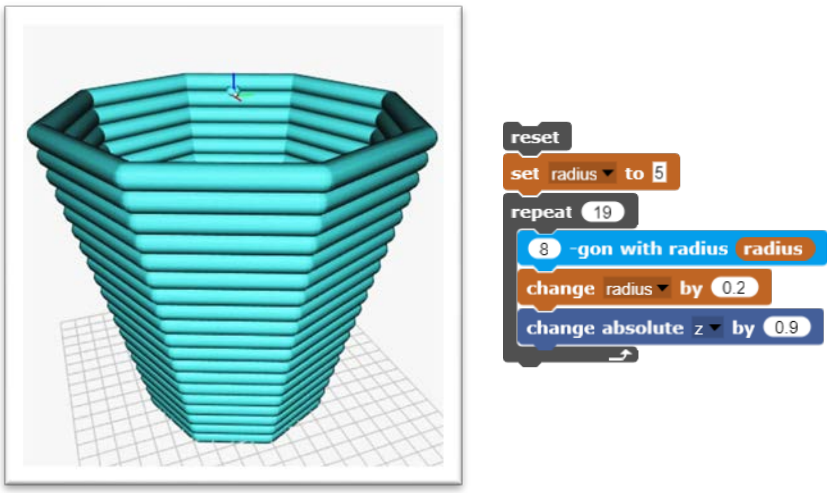
Coordinates of the first corner point must be placed twice on the stack!

Coordinates of the first corner point must be fetched two times from the stack!

Figure 17. Code of stack register based algorithm (on the right), visualization of the geometric situation for a triangle (on the left).

This approach seems, at first, much more demanding from a student's point of view: Some more geometrical “tricks” are needed and an invisible stack register is used.

A closer look, however, levels the alleged disadvantages: The geometrical situation can still be intuitively understood following the beetle movements. Furthermore, the stack register is a fundamental data structure in computer science, therefore makes it both comprehensible for younger students and valuable for CS education (Schwill, 1997). Considering these aspects, this strategy is appropriate for novices.



```
reset
set radius to 5
repeat 19
    8 -gon with radius (radius)
    change radius by 0.2
    change absolute z by 0.9
```

Figure 18. Vase consisting of stacked octagons constructed using the stack register.

In conclusion, in this approach the difficulty of *advanced* school mathematics has been traded in for the need of *fundamental* CS knowledge in comparison to existing teaching attempts. This can be considered as another benefit from an educational perspective as well.

4. Experiences

Based on the concepts identified, we designed a 7 hours long teaching series for twelve-year-old students, who had never programmed before. We present our experiences in the following.

4.1 Introductory Lessons with 2D Turtle Geometry for motivating students

In the two initial lessons we introduced concepts of two-dimensional turtle art to the students and printed – in the lessons itself – their first, mostly flat, objects using a 3D printer in the classroom (see fig. 19). This highly motivated the students, because they could program their first objects quickly, watch how they were built in the 3D printer and immediately take them home after school presenting them to their family members.

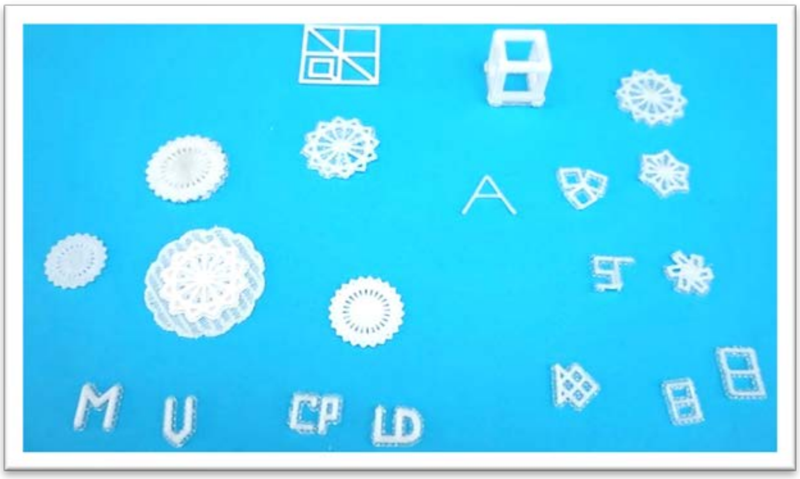


Figure 19. Students' results in the first both hours, printed out in a 3D printer.

While in our experience live printing in the classroom highly increases the students' motivation, it is still practicable only in the first lessons: Printing flat objects is done within minutes using standard 3D printers, but actual three-dimensional objects take a lot longer.

4.2 Long project phase allowing for creative results

In the 3rd and 4th lesson students were introduced to creating 3D objects by layering up base areas and varying their sizes using variables. Furthermore, they could explore the tricks from section 3.2.1, how formerly leaning towers can be turned into symmetrical vases. In the last three hours of the teaching series students could work on their own individual projects. Advanced students were given the opportunity to discover our stack register algorithm.

In general, the students created a broad range of different 3D artifacts: Especially decorative objects, such as vases, jewelry and small architectural prototypes were very popular. In order to get an impression, of what actual students' projects looked like, see figures 20 and 21.

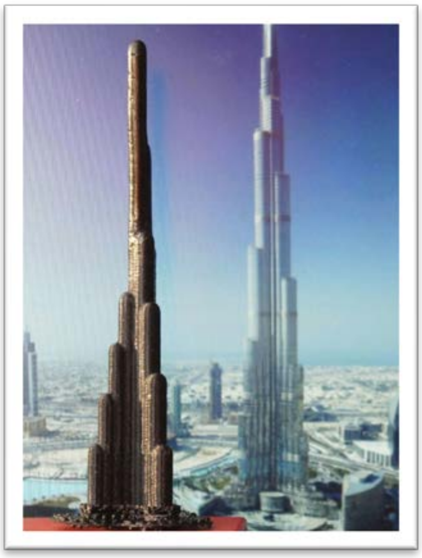


Figure 20. The "Burj Khalifa" model created by a group of students in front of its archetype photo.

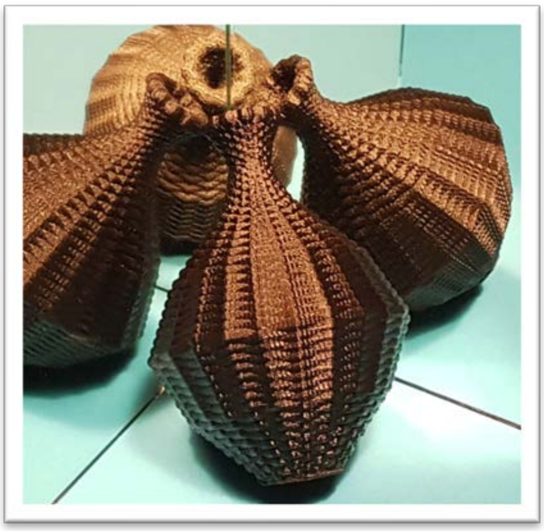


Figure 21. Vase constructed using the stack register in front of a mirror.

5. Conclusion

In this paper, we identified the underlying conceptual difference between 2D and 3D turtle geometry – the existence of *two* different coordinate systems – and explained its effects on how 3D objects can be created. Based on this, the big challenge of previous teaching attempts – the use of complicated advanced mathematical methods for creating base areas around their center point – is overcome with the concepts explored in section 3: We presented strategies, how some traditionally constructed base areas can be enhanced for making symmetric 3D objects possible, and even provided a general solution for all regular polygonal base areas. Furthermore, we showed that two-dimensional turtle art can be successfully recreated in 3D – completing our collection of fundamental concepts in 3D turtle geometry. Following these concepts, we designed a teaching series for programming novices *without* the mathematical challenges mentioned. Our experiences from the classroom show the potential of 3D turtle geometry for fostering the students' creativity.

Even though well-proven teaching concepts for the introduction to algorithms already exist, like creative teaching with Scratch (Romeike, 2008), established approaches regarding turtle geometry, in combination with modern 3D printing technologies offer a special connection to the real world. Never has it been so easy to transform abstract ideas into physical objects – inspiring even more programming novices and their teachers. In conclusion, 3D turtle geometry provides another encouraging setting for students.

Overall, our work can be seen as a first attempt to systematize the fundamental concepts of 3D turtle geometry for computer science classes. More advanced and promising concepts, such as recursion in fractals, are yet to be explored.

References

Buehler, E., Kane, S. K., & Hurst, A. (2014). Abc and 3D: Opportunities and Obstacles to 3D Printing in Special Education Environments. In S. Kurniawan (Ed.), *Proceedings of the 16th International*

SIGACCESS Conference on Computers & Accessability [i.E. Accessibilty], October 20 - 22, 2014, Rochester, New York, USA (pp. 107–114). ACM. https://doi.org/10.1145/2661334.2661365

Kastl, P., Krisch, O., & Romeike, R. (2017). 3D Printing as Medium for Motivation and Creativity in Computer Science Lessons. In V. Dagienė & A. Hellas (Eds.), *Informatics in Schools: Focus on Learning Programming* (pp. 27–36). Springer International Publishing.

Lutz, R. (2013). Enhancing information technology education (ITE) with the use of 3D printer technology. In W. D. Armitage, R. Friedman, & K. Baker (Eds.), *Sigite'13: Proceedings of the 2013 ACM SIGITE Annual Conference on Information Technology Education : October 10-12, 2013, Orlando, Florida, USA* (p. 157). ACM Association for Computing Machinery. https://doi.org/10.1145/2512276.2512327

Papert, S. (1972). On making a theorem for a child. In J. J. Donovan & R. Shields (Eds.), *Proceedings of the ACM annual conference on - ACM'72* (p. 345). ACM Press. https://doi.org/10.1145/800193.569942

Romagosa, B., Rosenbaum, E., & Koschitz, D. (2016). *From the Turtle to the Beetle: The Beetle Blocks programming environment*. http://goo.gl/QKpu8H

Romeike, R. (2008). Workshop: A Creative Introduction to Programming with Scratch. In M. Kendall & B. Samways (Eds.), *Learning to Live in the Knowledge Society* (pp. 341–344). Springer US.

Schwill, A. (1997). Computer science education based on fundamental ideas. *IFIP*. https://link.springer.com/content/pdf/10.1007%2F978-0-387-35081-3_36.pdf

Solomon, C. J., & Papert, S. (1976). A case study of a young child doing turtle graphics in LOGO. In Unknown (Ed.), *Proceedings of the June 7-10, 1976, national computer conference and exposition on - AFIPS '76* (p. 1049). ACM Press. https://doi.org/10.1145/1499799.1499945