

Chapter 12 - Are We Living in a Computer
Simulation

Matthew-Stephen Wileman, Bianca-Gabriela Leoveanu

Contents

1	Abstract	2
2	Introduction	2
2.1	Aim	2
2.2	Scope	2
2.3	Methodology	3
3	Theoretical Part	3
4	Practical Part	4
4.1	Implementation	4
4.1.1	The Pond Simulation	4
4.1.2	Implications of the Pond Simulation	5
4.1.3	Measuring the Cost of Crossing the Lattice	6
4.2	Results	8
5	Summary	10

1 Abstract

This project explores the simulation hypothesis by constructing a virtual pond using Python's turtle module [2, 6]. A turtle named Yertle navigates this environment, revealing computational artifacts such as anisotropy and grid-based limitations [4, 3]. By comparing straight and angled movements, we simulate how scientists might detect signs of a simulated reality.

2 Introduction

This project explores the simulation hypothesis proposed by philosopher Nick Bostrom, which suggests that our universe might be a computer simulation [2]. Using Python's turtle module, we built a simple simulated environment and performed an experiment to search for computational artifacts, simulating how scientists might detect evidence of a simulated reality [6].

2.1 Aim

The aim of this project is to develop a Python-based simulation that models a virtual world (Yertle's pond) and performs an experiment comparing computational costs of movement along and across a simulated grid, to explore potential signatures of an underlying simulation structure.

2.2 Scope

- Creating a simulated environment (a pond, a log, and a turtle) using turtle graphics [6].
- Programming a turtle (Yertle) to move along straight and angled paths [6].
- Measuring and comparing the time taken to draw straight vs. angled lines across a simulated grid [6].
- Analyzing the results as an analogy for detecting a simulation.

2.3 Methodology

To investigate the simulation hypothesis through a computational lens, we constructed a simplified virtual environment using Python’s turtle graphics module [6]. This environment mimics a pond where a turtle (Yertle) can move along both horizontal and diagonal paths. We then executed timed drawing operations to simulate the computational “cost” of different types of movement within this space [6].

Our project consists of:

1. Designing a rectangular grid to represent a virtual pond using turtle graphics [6].
2. Positioning Yertle the Turtle on a log and programmed movement sequences in horizontal and diagonal directions.
3. Using timing functions to measure the duration taken for each type of movement.
4. Recording and compared the timings to determine if directional bias in computation time exists.

3 Theoretical Part

In 2003, Nick Bostrom, a Swedish philosopher advanced The Simulation Argument which proposes that individuals are artificial constructs of future societies existing solely as silicon chips inside their data repositories [2].

Electronic simulations function upon distinct grids where the area becomes separated into pixels. The grid structure yields two primary consequences anisotropic behavior where motion varies according to orientation and granularity signifying the minimal quantifiable element[7].

The movement of the turtle demonstrated within this undertaking becomes influenced by these aforementioned artifacts. Linear trajectories extending from left to right appear fluid and non-linear trajectories seems uneven due to the display’s pixelated structure. This represents a consequence stemming from digital depiction affect[5].

The simulation hypothesis proposes one significant assertion namely that should our universe be simulated similar artifacts are potentially detectable[2]. Although the simulation within this undertaking lacks significant complexity it nonetheless mirrors this concept idea. Variations based on direction observed in both the animation process and the time required for execution illustrates how these limitations arise organically within models founded upon grids[4].

A prevalent pattern observed across simulations is that the orientation of movement may influences the computational resources necessary and this phenomenon is widely documented[3].

4 Practical Part

4.1 Implementation

4.1.1 The Pond Simulation

This simulation models a simplified pond environment using Python's turtle graphics library [6]. It features a pond, a mud island, a floating log, a knot-hole, and a turtle named Yertle who explores the environment. The program is visually driven and designed to serve as a metaphor for detecting signs of artificiality in a simulated world [2].

Setting Up the Environment The screen is initialized using the turtle module to create a 600x400 pixel pond. The background is colored light blue to mimic water. A mud island is created using a turtle object with a circular base, which is stretched into an oval shape and colored tan to resemble land [6].

```
import turtle
pond = turtle.Screen()
pond.setup(600, 400)
pond.bgcolor('light blue')
pond.title("Yertle's Pond")
mud = turtle.Turtle('circle')
mud.shapesize(stretch_wid=5, stretch_len=5, outline=None)
mud.pencolor('tan')
mud.fillcolor('tan')
```

Drawing the Log and Knothole A rectangular log is drawn using another turtle object. The log is rotated and placed diagonally across the screen. A small circular knothole is added for detail, using the circle method with a radius of 5 pixels [6].

```

SIDE = 80
ANGLE = 90
log = turtle.Turtle()
log.hideturtle()
log.pencolor('peru')
log.fillcolor('peru')
log.speed(0)
log.penup()
log.setpos(215, -30)
log.left(45)
log.begin_fill()
for _ in range(2):
    log.fd(SIDE)
    log.left(ANGLE)
    log.fd(SIDE / 4)
    log.left(ANGLE)
log.end_fill()
knot = turtle.Turtle()
knot.hideturtle()
knot.speed(0)
knot.penup()
knot.setpos(245, 5)
knot.begin_fill()
knot.circle(5)
knot.end_fill()

```

Animating Yertle the Turtle Yertle is a green turtle object that simulates movement across the pond. He swims from the island to the log and back, then swims out again at a different angle, representing deviation due to faulty memory—a symbolic nod to the unpredictability in simulated systems [6].

```

yertle = turtle.Turtle('turtle')
yertle.color('green')
yertle.speed(1)
yertle.fd(200)
yertle.left(180)
yertle.fd(200)
yertle.right(176)
yertle.fd(200)

```

4.1.2 Implications of the Pond Simulation

The pond simulation does more than display movement; it models how digital worlds operate under computational constraints. It reveals how simulations rely on discrete frameworks, such as pixel grids, to position and render all elements [6, 3].

Movement Limitations and Grid Resolution Since movement within this simulation occurs in whole pixels, the smallest object that can be displayed is one pixel in size. This highlights the discrete nature of simulated environments. If our universe were simulated in a similar manner, this would suggest the presence of a minimal observable unit—implying a resolution so fine that it can accommodate subatomic particles [3, 1].

Anisotropy and Path Bias Simulated grids may favor specific directions due to their rectangular structure. This directional bias, known as anisotropy, can affect how smoothly objects move depending on their angle. For instance, straight horizontal or vertical paths appear clean and uninterrupted, while diagonal paths tend to zigzag as the simulation compensates for the lack of true continuous movement across pixels [4, 3].

Energy Cost of Angled Movement From a computational standpoint, moving directly along the x or y axis involves simple arithmetic. In contrast, diagonal or angled movement requires trigonometric calculations to resolve components along both axes. This increased computational cost means that even if two movements cover equal distances, the diagonal one is inherently more expensive in terms of processing [4].

4.1.3 Measuring the Cost of Crossing the Lattice

This simulation investigates if rendering at an angle requires more duration compared to drawing horizontally. The objective involves observing whether displacement along a diagonal path versus movement parallel to an axis generates a quantifiable computational disparity. The findings derived from this investigation lend credence to the theoretical concept of anisotropy as it manifests within simulations constructed upon a pixel foundation [1].

The experimental procedure is executed utilizing Python’s turtle module. Specifically, two linear segments possessing identical lengths are rendered, one aligned horizontally and the other slightly diagonally at an angle approximating 3.7° . The duration necessary to render each segment is quantified employing high-resolution temporal measurement. To mitigate the impact of random variations, this procedure is reiterated 20 times for each angular orientation and the mean duration is subsequently calculated.

Importing the Essential Libraries and Configuration of the Drawing Environment The necessary modules are imported: **turtle** for drawing, **perf_counter** for precise time measurement, and **statistics** to calculate average performance results. The turtle canvas is set up large enough to accommodate both lines [6].

```
from time import perf_counter
import statistics
```

```
import turtle

turtle.setup(1200, 600)
screen = turtle.Screen()
```

Specification of Experimental Variables Two angles undergo evaluation: 0° for the horizontal segment and 3.695° for the slightly diagonal segment. These values are selected to possess identical actual lengths through reference to a Pythagorean triple. The turtle's drawing speed is set to maximum to eliminate animation delay [6].

```
ANGLES = (0, 3.695220532) # In degrees.
NUM_RUNS = 20
SPEED = 0
```

Drawing and Timing Each Line For every angular orientation, a distinct turtle instance is generated and appropriately adjusted. The turtle is rotated to the test angle, moved to a fixed starting point, and then used to draw a single line. This specific operation is executed 20 times for every angle.

```
for angle in ANGLES:
    times = []
    for _ in range(NUM_RUNS):
        line = turtle.Turtle()
        line.speed(SPEED)
        line.hideturtle()
        line.penup()
        line.lt(angle)
        line.setpos(-470, 0)
        line.pendown()
        line.showturtle()
```

Measuring Performance and Presentation of Outcomes The time necessary for drawing each individual segment is precisely documented by the **perf_counter()** function. The outcomes are averaged to facilitate comparison between horizontal and diagonal drawing durations [6].

```
start_time = perf_counter()
line.fd(962)
end_time = perf_counter()
times.append(end_time - start_time)

line_ave = statistics.mean(times)
print("Angle {} degrees: average time for {} runs at speed {} = {:.5f}"
      .format(angle, NUM_RUNS, SPEED, line_ave))
```

As explained by Vaughan, diagonal movement necessitates a greater degree of computational processing attributed to trigonometric computations, whereas horizontal segments rely upon simpler arithmetic operations [6].

4.2 Results

This simulation aimed to determine whether the direction of movement in a grid-based environment influences computational performance. By comparing the time it takes to draw two equal-length lines—one horizontal at 0° and one slightly diagonal at approximately 3.7° —the experiment evaluated whether angled motion incurs a higher computational cost.

Across all tests, the angled line consistently took longer to draw than the horizontal one. This confirms the hypothesis that angled movement, which requires more complex calculations such as trigonometry, incurs a greater computational cost than straight movement, which relies on simple addition or subtraction [6].

This difference is visually illustrated in Figures 1 and 4, where the turtle renders a diagonal line and a horizontal line respectively. As observed, both lines are of the same length, yet the diagonal trajectory introduces perceptible irregularities due to angular interpolation, reinforcing the theory of anisotropic effects in grid-based rendering.

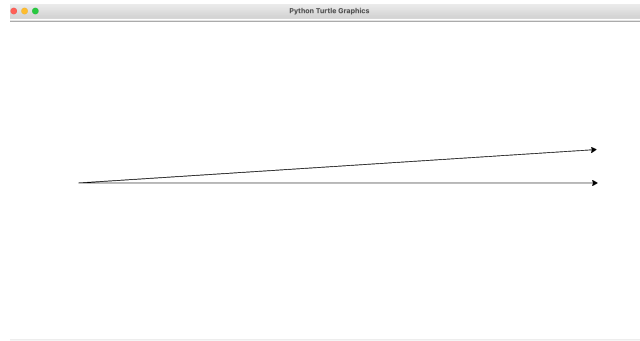


Figure 1: Turtle rendering a line at an angle of approximately 3.7°

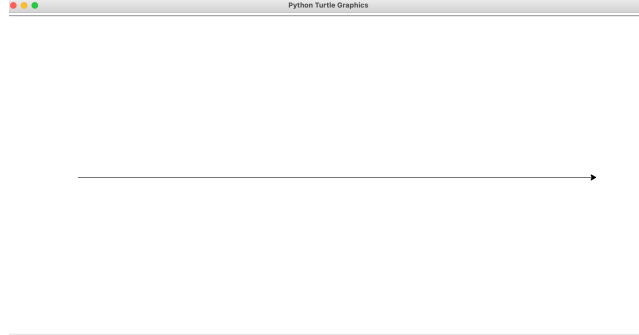


Figure 2: Turtle rendering a line horizontally at 0°

Runs	Speed	Direction	Avg. Time (s)	Ratio = $\frac{\text{Average time (angled)}}{\text{Average time (horizontal)}}$
20	0	0°	0.01585	-
20	0	3.7°	0.01882	1.19 \times
500	0	0°	0.02369	-
500	0	3.7°	0.05293	2.23 \times
1000	0	0°	0.02905	-
1000	0	3.7°	0.08258	2.84 \times
500	6	0°	0.53798	-
500	6	3.7°	0.82748	1.53 \times

Table 1: Results Table

Average Drawing Times and Ratios The results observed in Table 1 are consistent with the findings in *Real-World Python* [6], where the angled line took approximately 2.4 to 2.7 times longer than the horizontal one, depending on the run count and drawing speed.

Vaughan explains this phenomenon by noting that “moving along the x or y direction requires only integer addition or subtraction. Moving at an angle requires trigonometry... so we can surmise that moving at an angle takes more energy” [6]. Even at slower drawing speeds, where speed=6 and animation time dominates, the angled path still incurred significantly higher execution time.

These findings support the concept of anisotropy in simulated environments. This mirrors theoretical expectations proposed by Beane et al., who suggest that directional artifacts could arise if our universe were itself a simulation [1].

Therefore, this experiment demonstrates that even in simple digital simulations, directionality has measurable effects on performance, providing computational evidence of structure beneath the visual output.

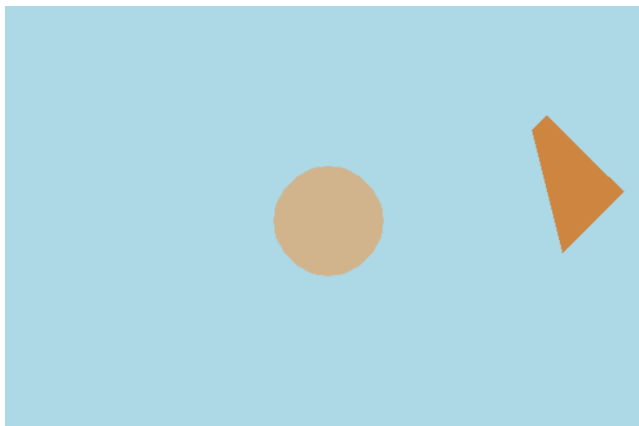


Figure 3: Environment

5 Summary

This project demonstrated how a simple turtle-based simulation can highlight deeper ideas about computational worlds. We explored how fundamental properties—like pixel grids and movement constraints—could offer clues pointing toward a simulated reality [6, 3, 4]. By investigating anisotropy and measuring computational cost, we simulated a strategy used by physicists to explore whether our universe may be running on code [1].

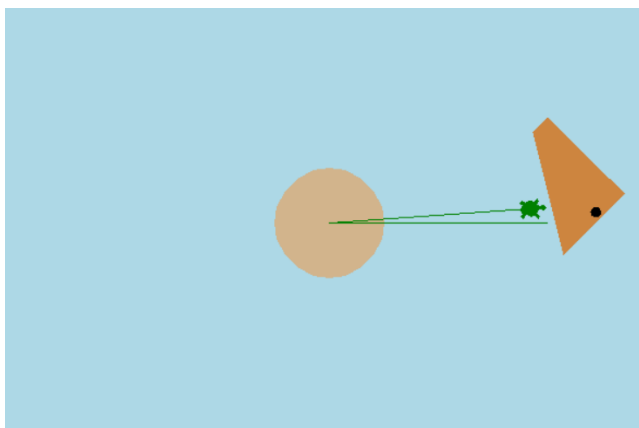


Figure 4: Fully Built Environment + Yertle & Movement

References

- [1] Silas R. Beane, Zohreh Davoudi, and Martin J. Savage. “Constraints on the Universe as a Numerical Simulation”. In: *The European Physical Journal A* 50.9 (2014), pp. 1–9.
- [2] Nick Bostrom. “Are you living in a computer simulation?” In: *Philosophical Quarterly* 53.211 (2003), pp. 243–255.
- [3] Konrad Hinsen. “Grid effects in scientific simulations”. In: *Computers in Physics* 14.6 (2000), pp. 391–397.
- [4] David Markus. “Artifacts in grid-based modeling of motion in digital environments”. In: *Simulation Modelling Practice and Theory* (2009).
- [5] Rachel McDonnell et al. “Perceptual effects in simulated motion over rasterized surfaces”. In: *ACM Transactions on Graphics* (2012).
- [6] Lee Vaughan. *Real-World Python: A Hacker’s Guide to Solving Problems with Code*. No Starch Press, 2020.
- [7] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.