

# No Three In Line: Learning Manim

Noah Morris  
CSCI 270 Final Project  
Hendrix College

My project taught me a lot about two topics that have long fascinated me. Below is a summarization of my findings.

I learned to animate in python, because code-based animation has always seemed like a potential great product. A friend of mine who studies art with emphasis on animation has long bemoaned the counter-intuitive, overly complex, and prohibitively expensive currently available animation software. I wanted to learn how to make 2d animations with python, because doing so (perhaps in a more general, AI-powered way) sounds like a great future project. Step one was to learn about what exists, which (among the free options) is manim.

I learned about an unsolved math problem, called "No Three In Line," which has enchanted me since I found it on Wikipedia a couple years ago. I turned it into a mild abstract-strategy game, and have given the scenario a good deal of non-rigorous thought.

## THE PROBLEM:

Suppose you've got an  $n \times n$  board, and  $2 \cdot n$  markers. You want to arrange them in such a way that no three form a straight line at any angle. This problem is identical to another famous unsolved geometry problem, involving maximizing the area of the smallest triangle formable from an arrangement of  $n$  points. It's easy to bump up against the unknown fooling around with it through code. Consider the filled grid as a list of tuples. Thus

```
[ (2,1), (0,0), (1,1), (3,0),      X.X.
  (2,3), (0,2), (3,2), (1,3) ] -> .X.X
                                   .X.X
                                   X.X.
```

To assess that no three points form a line, you must check that each group of three points satisfies  $(x_3 - x_2) / (x_2 - x_1) = (y_3 - y_2) / (y_2 - y_1)$ . These can be rewritten as multiplication, to avoid division by zero errors, though they must still be dealt with beforehand.

After writing a basic method, I needed to learn how to screen out solutions that were symmetric to ones that were already found. Below is a table of transformations on tuples that rotate and reflect squares in all possible ways.

$(x, y)$	$0^\circ$	$\leftrightarrow$	$(y, x)$	$0^\circ$ REFLECTED
$(y, -x)$	$90^\circ$	$\leftrightarrow$	$(x, -y)$	$90^\circ$ REFLECTED
$(-x, -y)$	$180^\circ$	$\leftrightarrow$	$(-y, -x)$	$180^\circ$ REFLECTED
$(-y, x)$	$270^\circ$	$\leftrightarrow$	$(-x, y)$	$270^\circ$ REFLECTED

Even with this, a brute force approach takes  $n^3$  time to assess a single grid, and the amount of possible grids is  $(n^2 \text{ choose } 2n)$ , which is horrendously exponential. Thus, my algorithm took 45 seconds to find all solutions for the 5x5, which are pictured below.

Sym.	Abt.	Diag		No Symmetry		
XX...	X.X..			XX...	X.X..	.XX..
X..X.	..X.X			.X.X.	.X..X	X...X
...XX	XX...			X...X	...XX	...XX
.XX..	...XX			..X.X	XX...	XX...
..X.X	.X.X.			..XX.	..XX.	..XX.

The 4x4s each have a different symmetry group.

$180^\circ$	4-way Rotational	$180^\circ$ and $180^\circ$ REFL	8-Fold
XX..	X.X.	X.X.	.XX.
..XX	.X.X	..XX	X..X
XX..	.X.X	XX..	X..X
..XX	X.X.	.X.X	.XX.

General solutions are far easier to find when assuming some degree of symmetry, as this limits the possible degrees of freedom. One unsolved problem is whether there exists a solution larger than the 10x10 which has 8-fold symmetry. Additionally, no solution has been found with 2-fold reflectional symmetry but no rotational symmetry - if this was found, I'd replace the "8-fold" above with "16-fold."

There exist algorithms for placing less than  $2n$  non-tricollinear points in a grid of size  $n$ . The first, and most famous, is due to Erdos, who used

$$[(i, i*i \% n) \text{ for } i \text{ in range}(n)]^1$$

This pattern has an appealing reflectional symmetry, and large instances look pleasantly like aliens. I tried in vain to prove that this always holds. I managed to word the problem in pure number-theoretic terms, but got stuck, and didn't include any of the incomplete derivations in the video.

Let  $i, j, p \mid \text{prime } p, i < j, 2i-j < p$ .

Show that  $2(j^2 \% p) \neq (i^2 \% p) + ((2i+j)^2 \% p)$

MANIM:

I first learned about Manim from YouTube. In times of leisure I often watch math videos, and modern ones are almost always rendered using this software. I've wanted to learn it, as it seems like a great tool for presentations, especially on the Math and Theoretical CS I plan on continuing pursuit of.

A scene is made by creating a class, with a method "construct," which details what to do and when to do it. Other functions can be defined within the class. To compile to mp4, run "%manim -qm [CLASSNAME]" in the terminal, or a jupyter notebook.

COMMON COMMANDS:






- self.wait() - adds a moment of stillness to the video before moving on to the next thing.

- `self.play(ACTION)` - performs interior action.
- `self.add(MOBJECT)` - places object on screen.

#### ACTIONS:

- `Write(Text(str))` - Creates standard text and prints it to the screen. Useful parameters include `font_size::Int`, `next_to([MOBJECT], [DIRECTION])`, `to_edge(DIRECTION)`.
- `Write(MathTex(r"[str]"))` - Compiles LaTeX and prints it to screen. Has all methods listed above.
- `Create(SHAPE)` - Creates one of many pre-built shapes, such as `Dot`, `Line`, `Star`, `Circle`, `Polygon`, `Arc`, and various operations for combining multiple. Does so with a bit more flourish than `self.add`.
- `Rotate(MOBJECT, ANGLE)` - Rotates given object. There's also `obj.animate.rotate`, which LERPs instead and is a bit more jagged.
- `Write(Code(str))` - Formats text as code, with syntax highlighting for various languages with `'language'` parameter.

The above only scratches the surface. This language has been maintained by a community of creatives for years now, and people have done some really incredible things with it. For example, in his latest series, 3Blue1Brown is using python's GPT-3 plugin to analyze and visualize the model's behavior and underlying algorithms (1). Folks other than the inventor have also had fun with it. Some other nice math explainers that served as inspiration for me are listed below.

- 1  But what is a GPT? Visual intro to transformers | Chapter 5...
- 2  The Fast Fourier Transform (FFT): Most Ingenious Algorithm E...
- 3  The Bubble Sort Curve
- 4  Percolation: a Mathematical Phase Transition
- 5  The Longest Increasing Subsequence

#### MANIM DEPENDENCIES:



```
s += '\n'
print(s)
return
```

FUTURE WORK:

If I expanded the video, one topic I didn't have time to get to was math consensus on the problem, which is contrary to the prevailing trend from computation. They think the pattern eventually breaks, because the amount of possible slopes along which a line can form grows quadratically to the side length. The formula is

$$\sum_{q=1}^{n-3} 4 * \varphi(q).$$

$\varphi$  is a number theory function that works like

$$\varphi n := n - \text{len}([m \in [1..n] \mid \text{gcd}(m,n) == 1])^1$$

n	1	2	6	7	56	57	58	59
$\varphi(n)$	1	1	2	6	24	36	28	58

This makes sense, because it essentially counts the number of unsimplifiable fractions with a given denominator. 6, which has many factors, only introduces 2 new slopes, while 7 introduces 7-1. The n-3 at the top of the summation is adjusting for the sequence doesn't really begin until n=3, and the formula is adjusted so that slopes only count when a line goes through at least three points.

I would also like to learn to animate other things. I want to train an AI to create animation in the modern standard CalArts style, and write a simple, pixel-level algorithm for generating mouth animations from strings (as an excuse to play with the IPA for a while). I think my efforts here could help in both of those, as well as in the future when I need to present something that's visually appealing.

---

<sup>1</sup>\*Math things are often way more intuitive with List Comprehensions. I first came to admire them playing <https://code.golf/>, and I'm grateful that AI and CH have both utilized them prominently (Thank you Dr. Ferrer).

There are two jupyter notebooks included in the github repository. One, "First Steps With Manim," is where I generated the clips for my videos, sort of on top of an existing tutorial. The second involves the problem itself.

#### SOURCES:

1. <http://wwwhomes.uni-bielefeld.de/achim/no3in/readme.html>  
- Most authoritative source online
2. [https://oeis.org/A000755/a000755\\_1.pdf](https://oeis.org/A000755/a000755_1.pdf)  
- Heuristic argument against it always happening, from an author of Winning Ways.
3. <https://oeis.org/A000755>  
[https://en.wikipedia.org/wiki/No-three-in-line\\_problem](https://en.wikipedia.org/wiki/No-three-in-line_problem)  
-The greatest websites of all time
4. <https://homepages.gac.edu/~jsiehler/NoThree/noThree.html>  
-A super nice slideshow that did the best job of explaining the situation to me of all of these. Probably the most influential source.

Github @ <https://github.com/brimstonetrader/n3il>

YouTube @ <https://youtu.be/JRstXzSdzuM>