Manim Animations

Intro

Visualizations are essential when trying to communicate scientific results. While text is often required to be mathematically precise, giving intuition and an initial understanding is far easier with a good picture. The `manim` package allows us to take this one step further with animations of mathematical concepts. If you have seen the Youtube channel 3blue1brown, these animations will look familiar because that is what he uses. Today we will learn some of the basics.

Installation

The starting code for this lab comes with a `pyproject.toml` file that specifies the `manim` package as a dependency. On top of that, we are also going to use `jax` in this lab, which is also included in the pyproject.toml file.

python environment

If you decided to use `virtualenv` for this lab, that is to create an environment with `python -m venv [ENV_NAME]` and activate it with `source [ENV_NAME]/bin/activate`, then you can install the dependencies with `pip install -r pyproject.toml`.

If you have `uv` installed, you can also use `uv` to install the dependencies with `uv sync` to create the environment.

Extras

There will be some additional dependencies that you need depending on your operation system, check out the instruction on manim's website to make sure you have all the prerequisites installed.

To ensure `manim` is installed correctly, on the command line run `manim --version`. You should see some text like `Manim Community v[Version]`.

Basic Examples

Creating Shapes

Create a file `quickstart.py`. Inside this file, add the following:

```python
import manim as mn

class CreateShapes(mn.Scene):
  def construct(self):
    # your code here
```

Each animation is an `mn.Scene`. Each scene will have a `construct` function which includes all the code to create and animate the objects in the scene.

Now to create shapes. There are many common shapes available in manim, such as `Circle`, `Square`, `Triangle`, `RegularPolygon`, etc. Create a circle with `circle = mn.Circle()`. This just initializes the object in code, so to play the animation, call `self.play(mn.Create(circle))`.

(Alternate ways to create: https://docs.manim.community/en/stable/reference/manim.animation.creation.html)

Manim Animations

Now that the code is written, we need to run it. In your terminal, run `manim -pql quickstart.py CreateShapes`. This should automatically open up a player where you can run the animation. It will also save it in `./media/videos/quickstart/480p15/`.

You can create multiple shapes by declaring multiple shapes, then calling them all at once like `self.play(mn.Create(shape1), mn.Create(shape2), ...)` and so on. Try a couple now, they should be created on top of each other.

Transformations

The `manim` package can be used to automatically transform shapes into other shapes. Create another class `SquareToCircle`, and in its `construct` function create a circle and a square. Animate the square creation as in the previous section. To transform the square into a circle, use the `Transform(old_object, new_object)` function inside a `self.play` call. This will take the attributes of new_object and give them to old_object, modifying old_object.

For a more challenging task, create a new class `CircleConvergence`. The goal is to inscribe a n-gon inside a circle using `RegularPolygon`, then slowly transform it by adding more sides until the n-gon looks indistinguishable from the circle. Use the code block below as a skeleton.

```
class CircleConvergence(mn.Scene):
  def construct(self):
    # initialize the circle and the list of regular n-gons
    # play the animation to create the circle and first n-gon (probably a triangle)

    # loop through the ngons, skipping the first one
    for ngon in sequence_of_ngons[1:]:
        # transform the current ngon to the next ngon
```

Attributes and Set Functions

In addition to different kinds of shapes, each shape also has a number of attributes that can either be defined at construction, or set later with an appropriate set function. Some options include border color, fill color, rotation, or position. For example, suppose we have a `circle` and a `square`. Then we can:

`circle.set_fill(mn.PINK, opacity=0.5) # set the color fill and opacity`

`circle.rotate(mn.PI / 3) # rotate the circle`

`square.next_to(circle, mn.DOWN, buff=0.5) # put square below circle, with 0.5 buffer`

`circle.shift(mn.LEFT) # shift the circle left`

Construct a few shapes and play around with all of these. All these are static attributes of the shapes, but we can also animate them. For any of these calls, we can prepend `.animate` to return an animation that will go in a self.play call. For example,
`self.play(circle.animate.rotate(mn.PI / 3))` will animate the rotation of circle.

Construct 4 different shapes in grid with different colors and rotations. Make some of the attributes initial, and some animated.

Update the `CircleConvergence` script to display the area of the n-gon as it trasnforms, which is an estimate of π. As the number of sides increase, the area should approach π. You can do this with a text

object `label = mn.Text(f'pi estimate{area:0.5f}')` which is created with `mn.Write(label)` inside a play call.

Further Animations

What would you like to animate?

- Something from your project?
- The Nintendo GameCube opening animation?
- Some kind of 3d surface? https://docs.manim.community/en/stable/reference/manim.mobject.three_d.three_dimensions.Surface.html
- etc.