Manim

May 29, 2019

Contents:

These docs are generated from the master branch of the Manim repo. You can contribute by submitting a pull request there.

CHAPTER 1

About

Animating technical concepts is traditionally pretty tedious, since it can be difficult to make the animations precise enough to convey them accurately. `Manim` uses Python to generate animations programmatically, which makes it possible to specify exactly how each one should run.

This project is still very much a work in progress, but I hope that the information here will make it easier for newcomers to get started using `Manim`.

CHAPTER 2

Installation

Instructions on installing Manim

2.1 Linux

2.1.1 Ubuntu

Install system libraries:

```
# apt install sox ffmpeg libcairo2 libcairo2-dev
```

Install Latex distribution:

```
# apt install texlive-full
```

Install manim via pypi:

```
# pip3 install manimlib
```

OR Install manim via the git repository with venv:

```
$ git clone https://github.com/3b1b/manim
$ cd manim
$ python3 -m venv ./
$ source bin/activate
$ pip3 install -r requirement.txt
```

To use manim in virtual environment you need to activate the environment with the `activate` binary by doing `source bin/activate`, to exit use the `deactivate` command.

Note: The git repository is updated first before the one on pypi. The git repository also includes project files used to produce 3b1b videos. Some of the old projects might not work as due to api changes.

Note: The required latex packages are dictated by `manimlib/tex_template.tex` which `texlive-full` will satisfy. The download size can be quite large. If you wish to install only the packages required to use manim, substitude `texlive-full` with:

```
texlive texlive-latex-extra texlive-fonts-extra
texlive-latex-recommended texlive-science texlive-fonts-extra tipa
```

2.2 Mac

A stub for mac installation

2.3 Windows

2.3.1 Install System Libraries

Make sure you have *Python 3* for Windows installed first:

https://www.python.org/downloads/windows/

Install ffmpeg:

https://ffmpeg.org/download.html#build-windows

Install sox:

http://sox.sourceforge.net/Main/HomePage

Install a latex distribution. On Windows MikTex is commonly used:

https://miktex.org/howto/install-miktex

2.3.2 Path configuration

To invoke commandline without supplying path to the binary the PATH environment needs to be configured. Below are template examples, please change the path according to your username and specific python version. Assuming all the softwares are installed with no alteration to the installation paths:

```
C:\Users\$username\AppData\local\Programs\Python\Python$version\
C:\Users\$username\AppData\local\Programs\Python\Python$version\Scripts\
C:\MikTex\miktex\bin\x64\
C:\ffmpeg\bin\
```

The path entries should be separated by semicolon.

2.3.3 Installing python packages and manim

Make sure you can start pip using `pip` in your commandline. Then do `pip install pyreadline` for the `readline` package.

Grab the pycairo wheel binary `pycairo-1.18.0-cp37-cp37m-win32.whl` from https://www.lfd.uci.edu/~gohlke/pythonlibs/#pycairo and install it via `pip C:\absolute\path\to\the\whl\file`

clone the manim repository if you have git `git clone https://github.com/3b1b/manim` or download the zip file from the repository page with `Clone or download` button and unzip it.

Open the commandline within the manim directory with `Shift + Right click` on an empty space in the folder and select `open command window here`

Install manim python dependencies with `pip install -r requirement.txt`

2.3.4 Test the installation

Type in `python -m manim -h` and if nothing went wrong during the installtion process you should see the help text.

Use `python -m manim example_scene.py SquareToCircle -pl` to render the example scene and the file should play after rendering. The movie file should be in `media/videos/example_scenes/480p15`

CHAPTER 3

Getting Started

Todd Zimmerman put together a very nice tutorial on getting started with `manim`, but is unfortunately outdated. It's still useful for understanding how `manim` is used, but the examples won't run on the latest version of `manim`.

3.1 Learning by Example

You create videos in manim by writing `Scene` instances. `example_scenes.py` contains a few simple ones that we can use to learn about manim. For instance, take `SquareToCircle`.

```
class SquareToCircle(Scene):
    def construct(self):
        circle = Circle()
        square = Square()
        square.flip(RIGHT)
        square.rotate(-3 * TAU / 8)
        circle.set_fill(PINK, opacity=0.5)

        self.play(ShowCreation(square))
        self.play(Transform(square, circle))
        self.play(FadeOut(square))
```

`construct()` specifies what is displayed on the screen when the `Scene` is rendered to video. You can render a `Scene` by running `extract_scene.py`. Run `python extract_scene.py -h` to see how it's used.

```
> python extract_scene.py -h
usage: extract_scene.py [-h] [-p] [-w] [-s] [-l] [-m] [-g] [-f] [-t] [-q] [-a]
                [-o OUTPUT_NAME] [-n START_AT_ANIMATION_NUMBER]
                [-r RESOLUTION] [-c COLOR] [-d OUTPUT_DIRECTORY]
                file [scene_name]

positional arguments:
  file                  path to file holding the python code for the scene
  scene_name            Name of the Scene class you want to see
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -p, --preview
  -w, --write_to_movie
  -s, --show_last_frame
  -l, --low_quality
  -m, --medium_quality
  -g, --save_pngs
  -f, --show_file_in_finder
  -t, --transparent
  -q, --quiet
  -a, --write_all
  -o OUTPUT_NAME, --output_name OUTPUT_NAME
  -n START_AT_ANIMATION_NUMBER, --start_at_animation_number START_AT_ANIMATION_NUMBER
  -r RESOLUTION, --resolution RESOLUTION
  -c COLOR, --color COLOR
  -d OUTPUT_DIRECTORY, --output_directory OUTPUT_DIRECTORY
```

The most common flags are `-p`, to automatically play the generated video, `-l`, to render in lower quality in favor of speed, and `-s`, to show the last frame of the `Scene` for faster development. Run `python extract_scene.py example_scenes.py SquareToCircle -pl` to produce a file called SquareToCircle.mp4 in the media directory that you have configured, and automatically play it.

Let's step through each line of the `Scene`. Lines 3 and 4 instantiate a `Circle` and `Square`, respectively. Both of these subclass `Mobject`, the base class for objects in manim. Note that instantiating a `Mobject` does not add it to the `Scene`, so you wouldn't see anything if you were to render the `Scene` at this point.

```
3 circle = Circle()
4 square = Square()
```

Lines 5, 6, and 7 apply various modifications to the mobjects before animating them. The call to `flip()` on line 5 flips the `Square` across the RIGHT vector. This is equivalent to a refection across the x-axis. Then the call to `rotate()` on line 6 rotates the `Square` 3/8ths of a full rotation counterclockwise. Finally, the call to `set_fill()` on line 7 sets the fill color for the `Circle` to pink, and its opacity to 0.5.

```
5 square.flip(RIGHT)
6 square.rotate(-3 * TAU / 8)
7 circle.set_fill(PINK, opacity=0.5)
```

Line 9 is the first to generate video. `ShowCreation`, `Transform`, and `FadeOut` are `Animation` instances. Each `Animation` takes one or more `Mobject` instances as arguments, which it animates when passed to `play()`. This is how video is typically created in manim. `Mobject` instances are automatically added to the `Scene` when they are animated. You can add a `Mobject` to the `Scene` manually by passing it as an argument to `add()`.

```
9  self.play(ShowCreation(square))
10 self.play(Transform(square, circle))
11 self.play(FadeOut(square))
```

`ShowCreation` draws a `Mobject` to the screen, `Transform` morphs one `Mobject` into another, and `FadeOut` fades a `Mobject` out of the `Scene`. Note that only the first argument to `Transform` is modified, and the second is not added to the `Scene`. After line 10 is executed `square` is a `Square` instance with the shape of a `Circle`.

3.2 Mathematical Objects

Everything that appears on screen in a manim video is a `Mobject`, or Mathematical Object. A `Mobject`'s appearance is determined by 3 factors:

- `m.points`, an Nx3 `numpy.array` specifying how to draw `m`
- `m`'s style attributes, such as `m.color`, `m.stroke_width`, and `m.fill_opacity`
- `m.submobjects`, a list of `Mobject` instances that are considered part of `m`

3.3 Animating Mobjects

Learn about animations.

3.4 Making a Scene

Talk about Scenes and organization, bring it all together.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search