Manim

Feb 19, 2020

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

The simplest way to install the system dependencies on Mac OS X is with Homebrew. Mac come preinstalled with python2, but to use manim, python3 is required

1. Install python3 https://docs.python.org/3/using/mac.html
2. Install Cairo: `brew install cairo`
3. Install Sox: `brew install sox`
4. Install ffmpeg: `brew install ffmpeg`
5. Install latex (MiKTeX): https://miktex.org/howto/install-miktex-mac
6. Install manimlib `pip install manimlib` (or `pip install --user manimlib` to just yourself)

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 `python -m pip install 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_scenes.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`, which has been updated to run on python 3.7. Note that you'll want to change *from big_ol_pile_of_manim_imports import ** to *from manimlib.imports import ** to work with the current codebase.

3.1 Learning by Example

3.1.1 SquareToCircle

`example_scenes.py` contains simple examples that we can use to learn about manim.

Go ahead and try out the `SquareToCircle` scene by running it with `$ manim example_scenes.py SquareToCircle -p` in manim directory.

```
from manimlib.imports import *

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))
```

Note: The flag `-p` plays the rendered video with default video player.

Other frequently used flags are:

- `-l` for rendering video in lower resolution (which renders faster)

- `-s` to show the last frame of the video.

Run `manim -h` all the available flags (`python -m manim -h` if you installed it to a venv)

Let's step through each line of `SquareToCircle`

```
3 class SquareToCircle(Scene):
```

You create videos in manim by writing `Scene` classes.

Each `Scene` in manim is self-contained. That means everything you created under this scene does not exist outside the class.

```
4 def construct(self):
```

`construct()` specifies what is displayed on the screen when the `Scene` is rendered to video.

```
5 circle = Circle()
6 square = Square()
```

`Circle()` and `Square()` create `Circle` and `Square`.

Both of these are instances of `Mobject` subclasses, 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.

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

`flip()` `rotate()` `set_fill()` apply various modifications to the mobjects before animating them. The call to `flip()` flips the `Square` across the RIGHT vector. This is equivalent to a refection across the x-axis.

The call to `rotate()` rotates the `Square` 3/8ths of a full rotation counterclockwise.

The call to `set_fill()` sets the fill color for the `Circle` to pink, and its opacity to 0.5.

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

To generated animation, `Animation` classes are used.

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()`.

`ShowCreation` draws a `Mobject` to the screen.

`Transform` morphs one `Mobject` into another.

`FadeOut` fades a `Mobject` out of the `Scene`.

Note: Only the first argument to `Transform` is modified, the second is not added to the `Scene`. `Transform` only changes the appearance but not the underlying properties.

After the call to `transform()` `square` is still a `Square` instance but with the shape of `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

Coordinate

By default, the scene in manim is made up by 8 x 14 grid. The grid is addressed using a numpy array in the form of [x, y, z]. For 2D animations only the x and y axes are used.

```
class DotMap(Scene):
    def construct(self):
        dots = dict()
        annos = dict()
        var_index = 0
        for x in range(-7, 8):
            for y in range(-4, 5):
                annos[f"{x}{y}"] = TexMobject(f"({x}, {y})")
                dots[f"{var_index}"] = Dot(np.array([x, y, 0]))
                var_index = var_index + 1
        for anno, dot in zip(annos.values(), dots.values()):
            self.add(anno)
            self.add(dot)
            self.wait(0.2)
            self.remove(anno)
```

Note: You can place objects outside this boundary, but it won't show up in the render.

4.1 Using Coordinates

Coordinates are used for creating geometries (*VMobject* in manim) and animations.

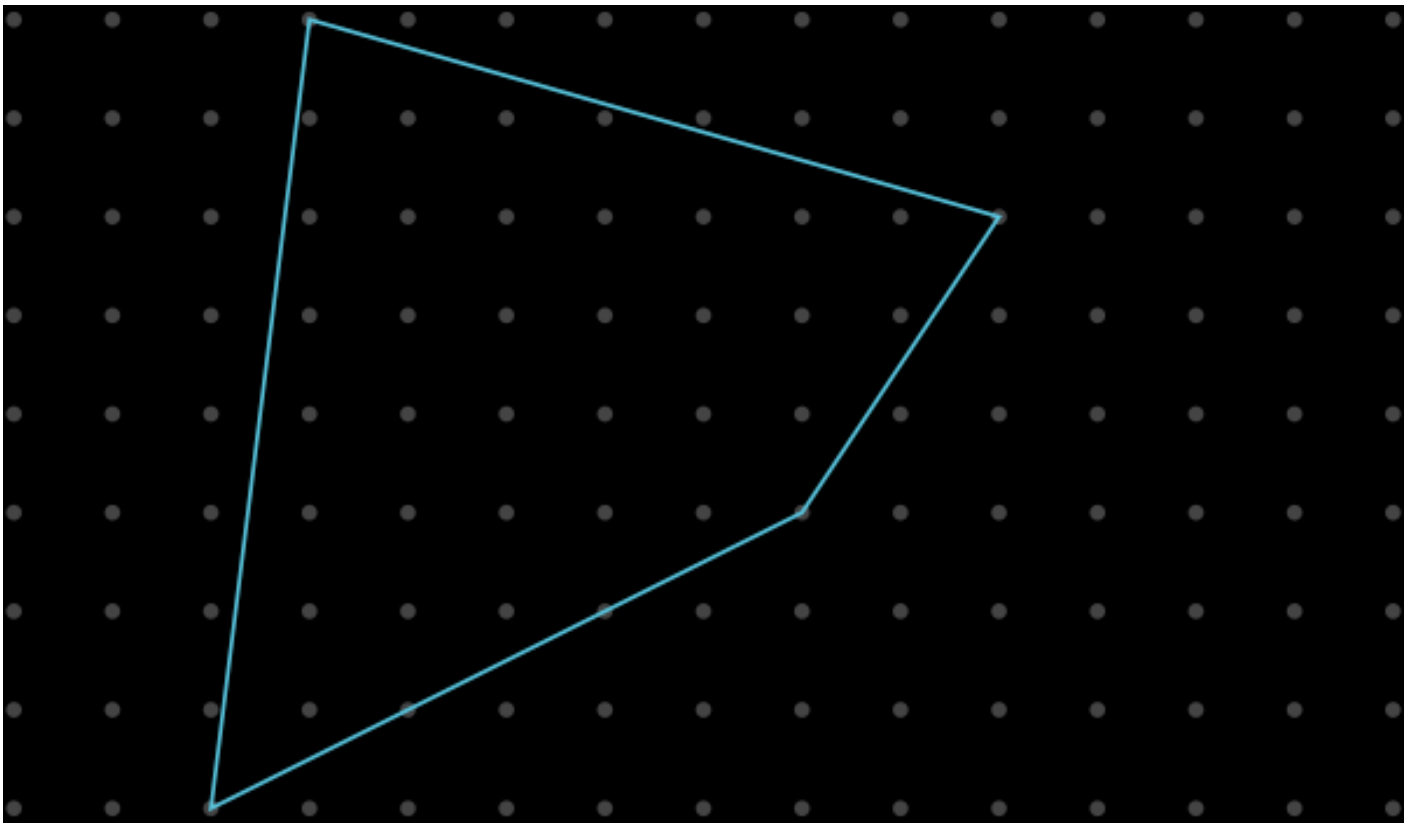
Here coordinates are used to create this Polygon

```
class CoorPolygon(Scene):
    def construct(self):
        for x in range(-7, 8):
```

(continues on next page)

(continued from previous page)

```
            for y in range(-4, 5):
                self.add(Dot(np.array([x, y, 0]), color=DARK_GREY))
        polygon = Polygon(
            np.array([3, 2, 0]),
            np.array([1, -1, 0]),
            np.array([-5, -4, 0]),
            np.array([-4, 4, 0]))
        self.add(polygon)
```



4.2 Coordinate Aliasing

From some animations typing a `np.array` everytime you need a coordinate can be tedious. Manim provides aliases to the most common coordinates:

```
UP == np.array([0, 1, 0])
DOWN == np.array([0, -1, 0])
LEFT ==  np.array([-1, 0, 0])
RIGHT == np.array([1, 0, 0])
UL == np.array([-1, 1, 0])
DL == np.array([-1, -1, 0])
UR == np.array([1, 1, 0])
DR == np.array([1, -1, 0])
```

Here coordinates are used for animations

```
class CoorAlias(Scene):
    def construct(self):
        for x in range(-7, 8):
            for y in range(-4, 5):
                self.add(Dot(np.array([x, y, 0]), color=DARK_GREY))

        aliases = {
            "UP": UP,
            "np.array([0,1,0])": np.array([0, 1, 0]),
            "DOWN": DOWN,
            "np.array([0,-1,0])": np.array([0, -1, 0]),
            "LEFT": LEFT,
            "np.array([-1,0,0])": np.array([-1, 0, 0]),
            "RIGHT": RIGHT,
            "np.array([1,0,0])": np.array([1, 0, 0]),
            "UL": UL,
            "np.array([-1,1,0])": np.array([-1, 1, 0]),
            "DL": DL,
            "np.array([-1,-1,0])": np.array([-1, -1, 0]),
            "UR": UR,
            "np.array([1,1,0])": np.array([1, 1, 0]),
            "DR": DR,
            "np.array([1,-1,0])": np.array([1, -1, 0])}
        circle = Circle(color=RED, radius=0.5)
        self.add(circle)
        self.wait(0.5)

        for text, aliase in aliases.items():
            anno = TexMobject(f"\\texttt{{{text}}}")
            self.play(Write(anno, run_time=0.2))
            self.play(ApplyMethod(circle.shift, aliase))
            self.wait(0.2)
            self.play(FadeOut(anno, run_time=0.2))
```

4.3 Coordinate Arithmetic

Numpy array allows arithmetic operations:

```
>>> numpy.array([2,2,0]) + 4
array([6, 6, 4])

>>> np.array([1, -3, 0]) + np.array([-4, 2, 0])
array([-3, -1,  0])

>>> np.array([2, 2, 0]) - np.array([3,6, 0])
array([-1, -4,  0])

>>> numpy.array([2,2,0]) - 3
array([-1, -1, -3])

>>> np.array([1, -3, 0]) * 3
array([ 3, -9,  0])

>>> numpy.array([2,2,0]) / 2
array([1., 1., 0.])
```

(continues on next page)

(continued from previous page)

```
>>> numpy.array([2,2,0]) / numpy.array([1, 4, 0])
__main__:1: RuntimeWarning: invalid value encountered in true_divide
array([2. , 0.5, nan])
```

```
class CoorArithmetic(Scene):
    def construct(self):
        for x in range(-7, 8):
            for y in range(-4, 5):
                self.add(Dot(np.array([x, y, 0]), color=DARK_GREY))

        circle = Circle(color=RED, radius=0.5)
        self.add(circle)
        self.wait(0.5)

        aliases = {
            "LEFT * 3": LEFT * 3,
            "UP + RIGHT / 2": UP + RIGHT / 2,
            "DOWN + LEFT * 2": DOWN + LEFT * 2,
            "RIGHT * 3.75 * DOWN": RIGHT * 3.75 * DOWN,
            # certain arithmetic won't work as you expected
            # In [4]: RIGHT * 3.75 * DOWN
            # Out[4]: array([ 0., -0.,  0.])
            "RIGHT * 3.75 + DOWN": RIGHT * 3.75 + DOWN}

        for text, aliase in aliases.items():
            anno = TexMobject(f"\\texttt{{{text}}}")
            self.play(Write(anno, run_time=0.2))
            self.play(ApplyMethod(circle.shift, aliase))
            self.wait(0.2)
            self.play(FadeOut(anno, run_time=0.2))
```

CHAPTER 5

Animation

The simplest of which is `Scene.add`. The object appears on the first frame without any animation:

```
class NoAnimation(Scene):
    def construct(self):
        square = Square()
        self.add(square))
```

Animation are used in conjunction with `scene.Play`

5.1 Fade

```
class AnimationFadeIn(Scene):
    def construct(self):
        square = Square()

        anno = TextMobject("Fade In")
        anno.shift(2 * DOWN)
        self.add(anno)
        self.play(FadeIn(square))
```

```
class AnimationFadeOut(Scene):
    def construct(self):
        square = Square()

        anno = TextMobject("Fade Out")
        anno.shift(2 * DOWN)
        self.add(anno)
        self.add(square)
        self.play(FadeOut(square))
```

```
class AnimationFadeInFrom(Scene):
    def construct(self):
        square = Square()
        for label, edge in zip(
            ["LEFT", "RIGHT", "UP", "DOWN"], [LEFT, RIGHT, UP, DOWN]
        ):
            anno = TextMobject(f"Fade In from {label}")
            anno.shift(2 * DOWN)
            self.add(anno)

            self.play(FadeInFrom(square, edge))
            self.remove(anno, square)
```

```
class AnimationFadeOutAndShift(Scene):
    def construct(self):
        square = Square()
        for label, edge in zip(
            ["LEFT", "RIGHT", "UP", "DOWN"], [LEFT, RIGHT, UP, DOWN]
        ):
            anno = TextMobject(f"Fade Out and shift {label}")
            anno.shift(2 * DOWN)
            self.add(anno)

            self.play(FadeOutAndShift(square, edge))
            self.remove(anno, square)
```

```
class AnimationFadeInFromLarge(Scene):
    def construct(self):
        square = Square()

        for factor in [0.1, 0.5, 0.8, 1, 2, 5]:
            anno = TextMobject(f"Fade In from large scale\_factor={factor}")
            anno.shift(2 * DOWN)
            self.add(anno)

            self.play(FadeInFromLarge(square, scale_factor=factor))
            self.remove(anno, square)
```

```
class AnimationFadeInFromPoint(Scene):
    def construct(self):
        square = Square()
        for i in range(-6, 7, 2):
            anno = TextMobject(f"Fade In from point {i}")
            anno.shift(2 * DOWN)
            self.add(anno)
            self.play(FadeInFromPoint(square, point=i))
            self.remove(anno, square)
```

5.2 Grow

```
class AnimationGrowFromEdge(Scene):
    def construct(self):
```

(continues on next page)

(continued from previous page)

```
        for label, edge in zip(
            ["LEFT", "RIGHT", "UP", "DOWN"], [LEFT, RIGHT, UP, DOWN]
        ):
            anno = TextMobject(f"Grow from {label} edge")
            anno.shift(2 * DOWN)
            self.add(anno)
            square = Square()
            self.play(GrowFromEdge(square, edge))
            self.remove(anno, square)
```

```
class AnimationGrowFromCenter(Scene):
    def construct(self):
        square = Square()

        anno = TextMobject("Grow from center")
        anno.shift(2 * DOWN)
        self.add(anno)

        self.play(GrowFromCenter(square))
```

5.3 Diagonal Directions

You can combine cardinal directions to form diagonal animations

```
class AnimationFadeInFromDiagonal(Scene):
    def construct(self):
        square = Square()
        for diag in [UP + LEFT, UP + RIGHT, DOWN + LEFT, DOWN + RIGHT]:
            self.play(FadeInFrom(square, diag))
```

Note: You can also use the abbreviated forms like `UL`, `UR`, `DL`, `DR`. See *Coordinates*.

CHAPTER 6

Manim Constants

The `constants.py` under `manimlib/` contains variables that are used during setup and running manim. Some variables are not documented here as they are only used internally by manim.

6.1 Directories

MEDIA_DIR The directory where `VIDEO_DIR` and `TEX_DIR` will be created, if they aren't specified via flags.

VIDEO_DIR Used to store the scenes rendered by Manim. When a scene is finished rendering, it will be stored under `VIDEO_DIR/module_name/scene_name/quality/scene_name.mp4`. Created under `MEDIA_DIR` by default.

TEX_DIR Files written by Latex are stored here. It also acts as a cache so that the files aren't rewritten each Latex is needed.

Those directories are created if they don't exist.

6.2 Tex

TEX_USE_CTEX A boolean value. Change it to True if you need to use Chinese typesetting.

TEX_TEXT_TO_REPLACE Placeholder text used by manim when generating tex files

TEMPLATE_TEX_FILE By default `manimlib/tex_template.tex` is used. If `TEX_USE_CTEX` is set to True then `manimlib/ctex_template.tex` is used.

6.3 Numerical Constants

PI alias to `numpy.pi`

TAU PI * 2

DEGREES TAU / 360

6.4 Camera Configuration

Render setting presets

PRODUCTION_QUALITY_CAMERA_CONFIG 2560x1440 @ 60fps # This is the default when rendering a scene

HIGH_QUALITY_CAMERA_CONFIG 1920x1080 @ 60fps. # Used when the `-h` or `--high_quality` flag is passed.

MEDIUM_QUALITY_CAMERA_CONFIG 1280x720 @ 30fps. # Used when the `-m` or `--medium_quality` flag is passed.

LOW_QUALITY_CAMERA_CONFIG 854x480 @ 15fps. # Used when the `-l` or `--low_quality` flag is passed.

6.5 Coordinates

Used for 2d/3d animations and placements:

```
ORIGIN
UP
DOWN
RIGHT
LEFT
IN # 3d camera only, away from camera
OUT # 3d camera only, close to camera

UL = UP + LEFT # diagonal abbreviations. You can use either one
UR = UP + RIGHT
DL = DOWN + LEFT
DR = DOWN + RIGHT

TOP
BOTTOM
LEFT_SIDE
RIGHT_SIDE``
```

6.6 Colors

COLOR_MAP A predefined color maps

PALETTE A list of color hex strings, derived from COLOR_MAP

CHAPTER 7

Indices and tables

- genindex
- modindex
- search