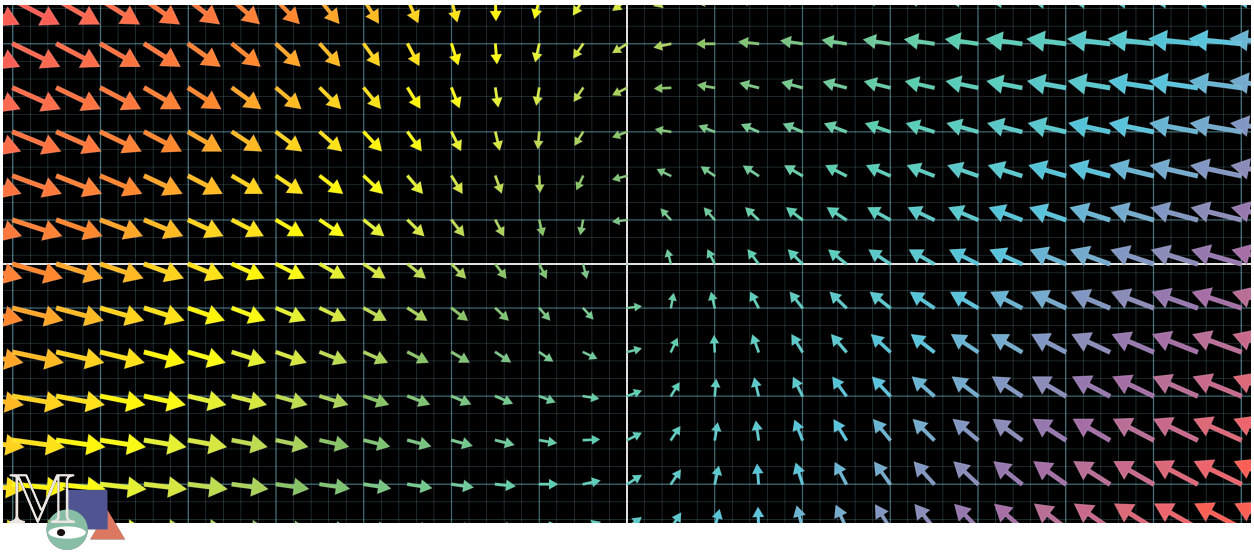


ManimGL Setup Guide Mac

by Jonathan Waldorf

What is Manim?

Manim is an open-source python library by Grant Sanderson to create math animations for his YouTube channel 3Blue1Brown. Manim is divided into the user-friendly community version ManimCE and the more powerful but less documented ManimGL. In this tutorial you will learn how to configure and effectively use ManimGL.

What is ManimGL?

ManimGL is the version of version of Manim used by Grant Sanderson, which has features based on OpenGL making it much faster at rendering than the community version. It includes jupyter notebook style functionality, where checkpoints can be inserted into your code so that the frame at that specific point is saved and you can run code from this frame until your cursor line to prototype small changes rapidly.

Packages Used

Python 3.13.4
manimpango 0.6.0
manimgl 1.7.2
audioop-lts (long term support)

Python Versioning

Before we begin, we'd like to make sure to be on the newest python version, so that the commands I use on this tutorial work on your console as well. I am using `Python 3.13.4` although newer versions should work fine as well. ManimGL requires at least `Python 3.8` to function. To update your python:

1. Go to the Official Python Page
2. Click the big download button that says "Latest Version"
3. Open the PKG and go through the installation instructions, then you should be good to go!

Package Installation

1. Open your spotlight by pressing `Command + SPACE` and open terminal
2. Check which python version you're running by typing `python -V` and pressing `ENTER`
3. paste the following command into the window that pops up, which will install Homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

4. Next use the command `brew install ffmpeg mactex` which will required packages
5. Make sure you're in the directory where you want to install manim, for me this is my OneDrive folder so I would type `cd OneDrive\ -\ QuEra\ Computing` which puts me

into my OneDrive folder

6. Create a folder to hold all your Manim stuff by running `mkdir Manim_Stuff`
7. Enter this folder with `cd Manim_Stuff`
8. Create a virtual environment by typing `python -m venv venv` . This will be where we install all our remaining packages and our python version.
9. Activate the virtual environment by running `source venv/bin/activate`
10. Run `pip install manimgl manimpango audioop-lts setuptools`
11. To test that your installation worked, run `manimgl` . If this is working properly, a blank popup should appear that you can close.

Font Installation

If you're reading this tutorial I'm sure you're a fan of 3Blue1Brown and his iconic Computer Modern font (which is also the default font of LaTeX!), which does not come installed on Mac by default. If you want to use this font in your videos, here are the instructions to get it running on your computer.

1. Go to this link and click download font for free
2. Open the .zip file that downloads to your computer
3. Press `Command + SPACE` and type `Font Book`
4. Open Font Book and drag the .ttf files inside the `cmu-serif` folder into the Font Book window

Once you're making text objects in your Manim code, make sure to specify the font being used:

```
helloworld = Text("Hello world", font="CMU Serif")
```

Or if you want text to be in the Computer Modern font by Default, you can use the snippet:

```
class CMText(Text):
    def __init__(self, text, **kwargs):
        kwargs.setdefault("font", "CMU Serif")
        super().__init__(text, **kwargs)
```

And then use the `CMText` object instead of `Text`.

QuEra Fonts

In addition we can download the fonts used for QuEra templates, which are variants of Radion B.

1. Go to this link and scroll down to click `download` on `Radion B Regular`, `Radion B Italic`, `Radion B Book`, `Radion B Book Italic`, `Radion B Demi`, and `Radion B Demi Italic`
2. Open all the .zip files, each one should give you a `.otf` file
3. Press `Command + SPACE` and type `Font Book`
4. Open Font Book and drag the `.otf` files into the Font Book window

Just like with Computer Modern, you will be able to choose these fonts for your `Text()` objects.

Sublime Text Editor

Grant's Manim setup uses Sublime, a lightweight code editor that he's configured for an optimized workflow. In this tutorial we will go through how to setup grant's workflow which will allow you to render snippets and edits to your manim scene in real time using checkpoints, which is far more efficient than remaking the scene for every little change you make.

Grant's workflow and keybinds can be found here

Sublime Setup

1. Install Sublime by going to this link and clicking “Install for Mac”, open the .DMG file and add to applications folder
2. Open a Sublime window
3. Press `Command + Shift + P` and then type `Package Control: Install Package` and click on it
4. Type `Terminus` and click on it
5. Press `Command + Shift + P` and then type `Package Control: Enable Package` and click on it
6. Type `Terminus` and click on it
7. Quit and Reopen Sublime
8. Open a new finder window, and press `Command + Shift + G`
9. Type in `~/Library/Application Support/Sublime Text/Packages/User` and hit enter
10. Go to this github link, download and copy the files in this folder into the finder folder you just opened
11. Go back to your Sublime window and go to the menu bar at the top of the screen and click `Sublime Text > Settings > Key Bindings`
12. Click on the tab on the right side, and replace the empty brackets with this:

```
   // Grant Sanderson Commands
[
   { "keys": ["shift+super+r"], "command": "manim_run_scene" },
   { "keys": ["super+r"], "command": "manim_checkpoint_paste" },
   { "keys": ["super+alt+r"], "command": "manim_recorded_checkpoint_past
e" },
   { "keys": ["super+ctrl+r"], "command": "manim_skipped_checkpoint_paste"
},
   { "keys": ["super+e"], "command": "manim_exit" },
   { "keys": ["super+option+/"], "command": "comment_fold"}
]
```

12. Save with `Command + S` and close out of this window

Shortcut Keybinds

Keybind	Action
CMD + Shift + R	Run scene at current cursor position
CMD + R	Paste a checkpoint command into the Terminus shell
CMD + Option + R	Paste a **recorded** checkpoint (record=true)
CMD + CTRL + R	Paste a **skipped** checkpoint (skip=True)
CMD + E	Quit the Manim/IPython session in Terminus
CMD + Option + /	Fold selected comment blocks for readability

Checkpoints

A checkpoint is a custom command like `checkpoint_paste()` sent to the Terminus terminal. It serves as a visual and semantic marker to document which lines were just run, often labeled with comments and the number of lines selected.

Language Server Protocol for Sublime

Sublime is a rather lightweight bare-bones code editor and given ManimGL's lack of documentation I find it useful to be able to easily find the definitions of different classes and functions within Manim. This can be most easily done through the use of LSP (Language Server Protocol). Here is how you get LSP running on Sublime:

1. Open a Sublime window
2. Press `Command + Shift + P` and then type `Package Control: Install Package` and click on it
3. Type `LSP` and click on it
4. Repeat steps 2-3 for `LSP-pyright` (Python) and `LSP-TexLab` (LaTeX)
5. In your virtual environment run `python -c "import sys; print(sys.prefix)"` , this should give the full path to your virtual environment which for me is: `/Users/jonathanwaldorf/Library/CloudStorage/OneDrive-QuEraComputing/Manim_Stuff/venv`
6. Press `Command + Shift + P` and then type `Preferences: LSP-pyright Settings` and click on it
7. Replace the brackets with:

```
{
  "settings": {
  "python.analysis.typeCheckingMode": "basic",
    "python.analysis.extraPaths": [
      "[VENV LOCATION]/lib/[PYTHON VERSION]/site-packages"
    ],
    "python.pythonPath": "[VENV LOCATION]/bin/python"
  }
}
```

Where `[VENV LOCATION]` is the virtual environment location you got in step 5 and `[PYTHON VERSION]` is your python version in the virtual environment. If you can't remember the python version, go back in your virtual environment and type `python3 --version`

8. Save this file by pressing `COMMAND + S` and close out of this window
9. Quit and restart Sublime.

Running your first Scene

Once all of the above configuration is installed, you can finally start making Manimations!

1. Open a Sublime window
2. Go to the menu bar at the top of the screen and click `File > Open` or use `Command + O`
3. Go to your manim animations folder, (which should be inside a larger folder alongside `venv` and `manim`) for me it's at `OneDrive > Manim_Stuff > Animation Projects` and click `Open`
4. Press `Command + N` to make a new file, then click on `Plain Text` in the bottom-right corner and replace it with `Python`

5. Create your Manim animation, or use this code snippet from the quickstart guide instead:

```
from manimlib import *

class CreateCircle(Scene):
    def construct(self):
        circle = Circle()
        circle.set_fill(BLUE, opacity=0.5)
        circle.set_stroke(BLUE_E, width=4)

        self.add(circle)
        self.play(circle.animate.shift(RIGHT), run_time=1) # Move circle to right
```

6. Save your file with `Command + S` , name the file `circletest.py` , make sure that you're in your manim animations folder, which for me is `OneDrive > Manim_Stuff > Animation Projects` and then hit save.
6. Go to the menu bar at the top of the screen and click `File > New Window`
7. Click on this new window and then use `Command + Shift + P` and then type `Terminus: Open Default Shell in Panel`
8. You should currently be in your main folder, denoted by `~` . Use the commands `cd <FOLDERNAME>` (enters FOLDERNAME) and `ls` (shows what folders are inside your current directory) to get into the folder containing your virtual environment. For me this is `OneDrive\ -\ QuEra\ Computing > Manim_Stuff`
9. Activate your virtual environment using `source venv/bin/activate`
10. Go into your manim animations folder, which for me would be `cd Animation_Projects`
11. Type `manimgl -pql circletest.py CreateCircle` and press enter
12. Click on the window that pops up and hit the spacebar
13. Congratulations, you've made your first animation!

Interacting with your scene

As it turns out, you can actually play with the scene you just made! One nice advantage of ManimGL is that the scenes you make are interactive, rather than a static video file. After clicking on the window with the circle:

- If you press `f` and move your mouse, you can pan around the scene you made
- If you press `d` and move your mouse, you can rotate the camera angle of your scene. While our manim code inherits from the `Scene` class (which constrains all Manim objects to be in the 2D plane the camera faces) rather than `ThreeDScene` , scenes rendered in ManimGL are all rendered in a 3D environment that we can explore to see the scene from multiple angles
- If you press `z` and use your scroll wheel, you can zoom in and out in your scene. (When I click on the scene and use my scroll wheel it seems to zoom in and out anyway so I'm not sure this is necessary)
- Pressing `r` resets the camera to the standard camera position

To exit the scene, press the `ESCAPE` key, or use `COMMAND + E`

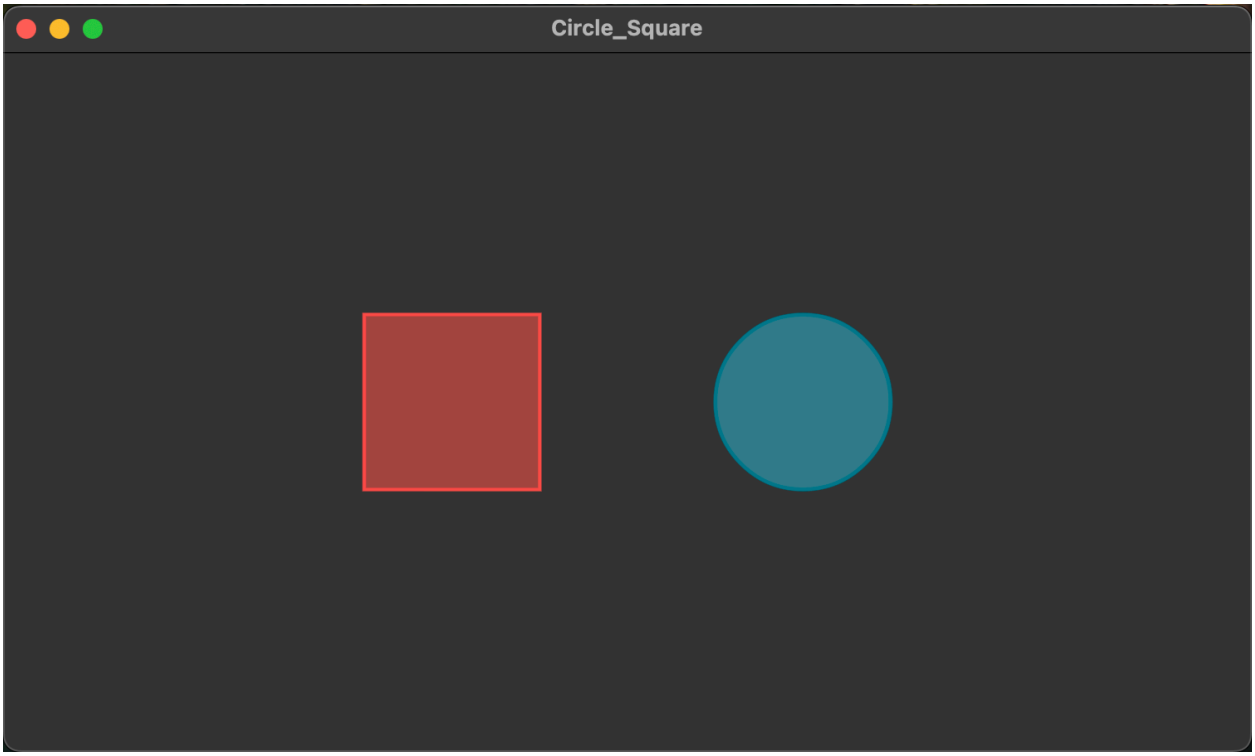
How to run snippets and edit in real time

You can also test small changes to your scene and see them in real time without re-rendering the whole scene! Let's say we have this code:

```
from manimlib import *

class Circle_Square(Scene):
    def construct(self):
        circle = Circle()
        circle.set_fill(BLUE, opacity=0.5)
        circle.set_stroke(BLUE_E, width=4)

        self.add(circle)
        self.play(circle.animate.shift(RIGHT), run_time=1)
        self.wait(1)
```

```
square = Square()
square.set_fill(RED, opacity=0.5)
square.set_stroke(RED_D, width=4)
self.add(square)
self.play(square.animate.shift(LEFT), run_time=1)
```

When we run this scene, we see that a blue circle is created that moves right, and a red square is made that moves left.



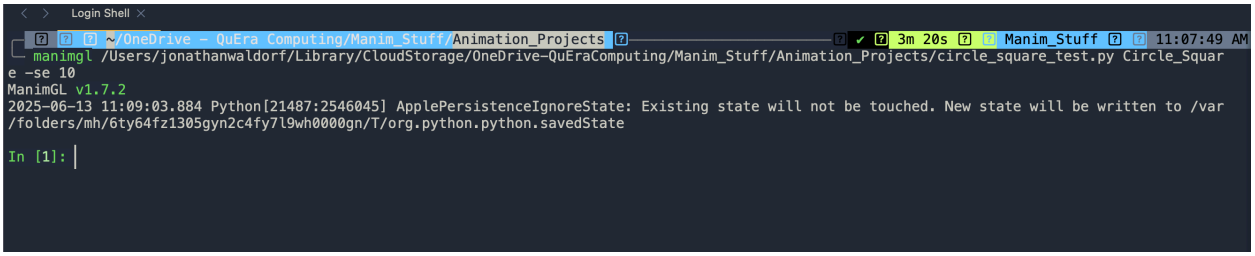
What if we wanted to try having the circle move up instead of to the right, without affecting the rest of the animation? First we can go to line 10 of our program and change the line to read `self.play(circle.animate.shift(2*UP), run_time=1)` . To test this change, we could put our cursor at the end of the modified line:

circle_square_test.py circletest.py

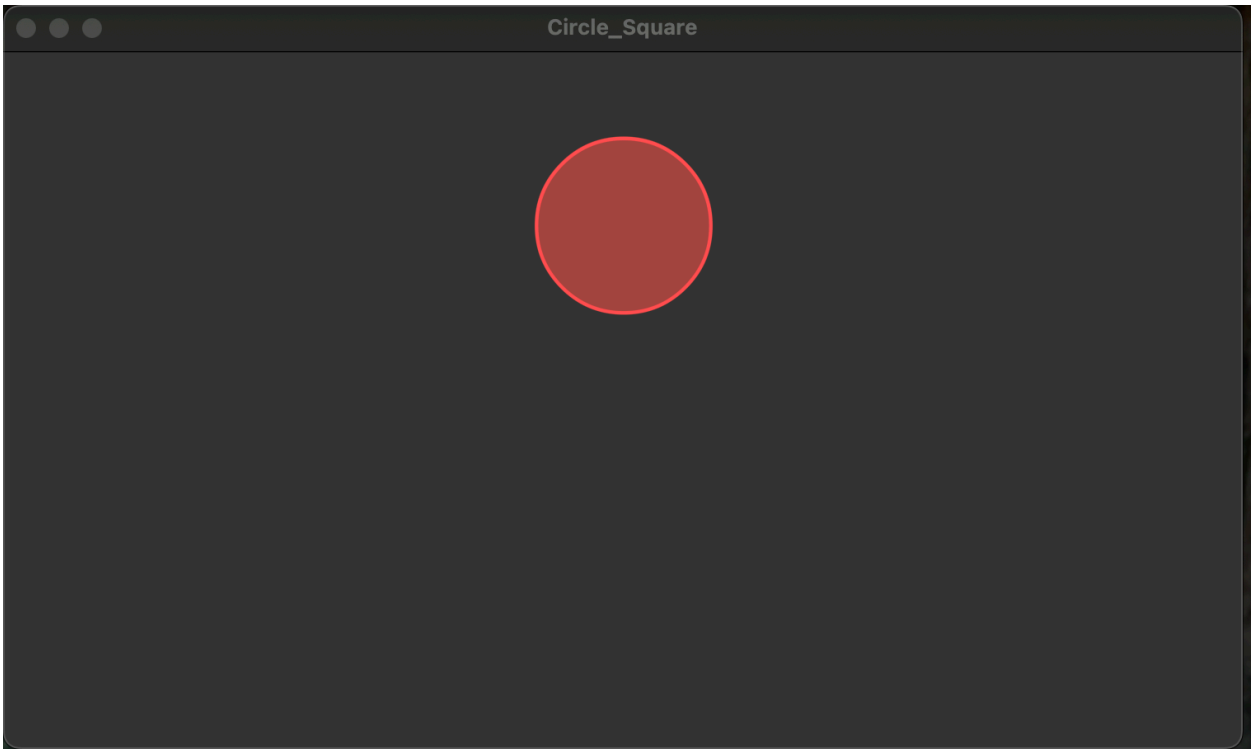
```python
from manimlib import *

class Circle_Square(Scene):
    def construct(self):
        circle = Circle()
        circle.set_fill(BLUE, opacity=0.5)
        circle.set_stroke(BLUE_E, width=4)

        self.add(circle)
        self.play(circle.animate.shift(2*UP), run_time=1)

        square = Square()
        square.set_fill(RED, opacity=0.5)
        square.set_stroke(RED_D, width=4)
        self.add(square)
        self.play(square.animate.shift(2*LEFT), run_time=1)

```

and then press `COMMAND + SHIFT + R` to run the code up to the point on your cursor! All animations up to this point happen instantaneously so we are only shown the final frame of the scene up to this point:

This command also creates a python terminal within our Login Shell which looks like this:



In this terminal, we can type python commands that will be executed on the scene! This is awesome for prototyping, for instance if I want to make the circle green I can type in this command: `circle.set_color(RED)` which will turn the circle on the screen red:



to exit this preview mode, we can press `COMMAND + E` or type `exit` in the python terminal. Generally if we want to enter the interactive shell in a scene, we can add the command `self.embed()` as the last line of our manim scene.

Now that we've setup ManimGL, check out the ManimGL for Beginners guide:

ManimGL for Beginners