MoviePy v2.0 have introduced breaking changes, see Updating from v1.X to v2.X for more info.

Compositing multiple clips

Video composition, also known as non-linear editing, is the fact of mixing and playing several clips together in a new clip. This video is a good example of what compositing you can do with MoviePy:

Skip to main content

Note

Before starting, note that video clips generally carry an audio track and a mask, which are also clips. When you compose these clips together, the soundtrack and mask of the final clip are automatically generated by putting together the soundtracks and masks of the clips. So most of the time you don't need to worry about mixing the audio and masks.

Juxtaposing and concatenating clips

Two simple ways of putting clips together is to concatenate them (to play them one after the other in a single long clip) or to juxtapose them (to put them side by side in a single larger clip).

Concatenating multiple clips

Concatenation can be done very easily with the function `concatenate_videoclips()`.

```
"""Let's concatenate (play one after the other) three video clips."""

from moviepy import VideoFileClip, concatenate_videoclips

# We load all the clips we want to concatenate
clip1 = VideoFileClip("example.mp4")
clip2 = VideoFileClip("example2.mp4").subclipped(0, 1)
clip3 = VideoFileClip("example3.mp4")

# We concatenate them and write the result
final_clip = concatenate_videoclips([clip1, clip2, clip3])
final_clip.write_videofile("final_clip.mp4")
```

The `final_clip` is a clip that plays the clips 1, 2, and 3 one after the other.

Note

The clips do not need to be the same size. If they aren't, they will all appear centered in a clip large enough to contain the biggest of them, with optionally a color of your choosing to fill the background.

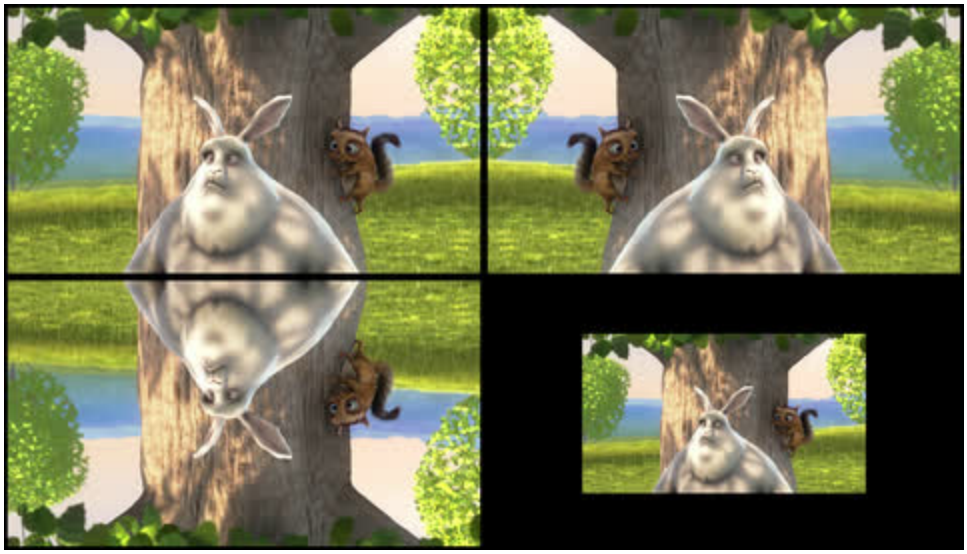
Skip to main content

Juxtaposing multiple clips

Putting multiple clip side by side is done with `clip_array()`:

```
"""Let's juxtapose four video clips in a 2x2 grid."""

from moviepy import VideoFileClip, clips_array, vfx

# We will use the same clip and transform it in 3 ways
clip1 = VideoFileClip("example.mp4").with_effects([vfx.Margin(10)])  # add 10px co
clip2 = clip1.with_effects([vfx.MirrorX()])  # Flip horizontaly
clip3 = clip1.with_effects([vfx.MirrorY()])  # Flip verticaly
clip4 = clip1.resized(0.6)  # downsize to 60% of original

# The form of the final clip will depend of the shape of the array
# We want our clip to be our 4 videos, 2x2, so we make an array of 2x2
array = [
    [clip1, clip2],
    [clip3, clip4],
]
final_clip = clips_array(array)
# let's resize the final clip so it has 480px of width
final_clip = final_clip.resized(width=480)

final_clip.write_videofile("final_clip.mp4")
```

You obtain a clip which looks like this:



For more info, see `clip_array()`.

Skip to main content

The `CompositeVideoClip` class is the base of all video compositing. For example, internally, both `concatenate_videoclips()` and `clip_array()` create a `CompositeVideoClip`.

It provides a very flexible way to compose clips, by playing multiple clip *on top of* of each other, in the order they have been passed to `CompositeVideoClip`, here's an example :

```
"""Let's stack three video clips on top of each other with
CompositeVideoClip."""

from moviepy import VideoFileClip, CompositeVideoClip

# We load all the clips we want to compose
clip1 = VideoFileClip("example.mp4")
clip2 = VideoFileClip("example2.mp4").subclipped(0, 1)
clip3 = VideoFileClip("example.mp4")

# We concatenate them and write theme stacked on top of each other,
# with clip3 over clip2 over clip1
final_clip = CompositeVideoClip([clip1, clip2, clip3])
final_clip.write_videofile("final_clip.mp4")
```

Now `final_clip` plays all clips at the same time, with `clip3` over `clip2` over `clip1`. It means that, if all clips have the same size, then only `clip3`, which is on top, will be visible in the video... Unless `clip3` and/or `clip2` have masks which hide parts of them.

> **Note**
>
> Note that by default the composition has the size of its first clip (as it is generally a *background*). But sometimes you will want to make your clips *float* in a bigger composition. To do so, just pass the size of the final composition as `size` parameter of `CompositeVideoClip`.

For now we have stacked multiple clip on top of each others, but this is obviously not enough for doing real video compositing. For that, we will need to change when some clip starts and stops playing, as well as define the x:y, position of these clips in the final video.

For more info, see `CompositeVideoClip`.

Changing starting and stopping times of clips

Skip to main content

In a CompositionClip, each clip start to play at a time that is specified by his `clip.start` attribute, and will play until `clip.end`.

So, considering that you would want to play `clip1` for the first 6 seconds, `clip2` 5 seconds after the start of the video, and finally `clip3` at the end of `clip2`, you would do as follows:

```
from moviepy import VideoFileClip, CompositeVideoClip

# We load all the clips we want to compose
clip1 = VideoFileClip("example.mp4")
clip2 = VideoFileClip("example2.mp4").subclipped(0, 1)
clip3 = VideoFileClip("example3.mp4")

# We want to stop clip1 after 1s
clip1 = clip1.with_end(1)

# We want to play clip2 after 1.5s
clip2 = clip2.with_start(1.5)

# We want to play clip3 at the end of clip2, and so for 3 seconds only
# Some times its more practical to modify the duration of a clip instead
# of his end
clip3 = clip3.with_start(clip2.end).with_duration(1)

# We write the result
final_clip = CompositeVideoClip([clip1, clip2, clip3])
final_clip.write_videofile("final_clip.mp4")
```

Note

When working with timing of your clip, you will frequently want to keep only parts of the original clip. To do so, you should take a look at **`subclipped()`** and **`with_section_cut_out()`**.

Positioning clips

Frequently, you will want a smaller clip to appear on top of a larger one, and decide where it will appear in the composition by setting their position.

You can do so by using the **`with_position()`** method. The position is always defined from the top left corner, but you can define it in many ways :

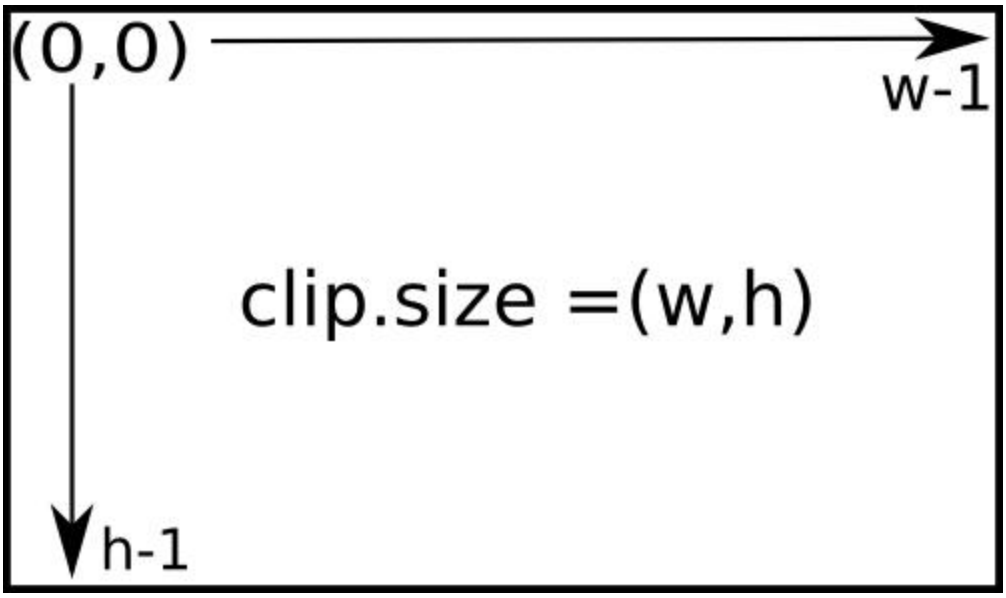
Skip to main content

```
"""Let's position some text and images on a video."""

from moviepy import TextClip, VideoFileClip, CompositeVideoClip, ImageClip

# We load all the clips we want to compose
background = VideoFileClip("example2.mp4").subclipped(0, 2)
title = TextClip(
    "./example.ttf",
    text="Big Buck Bunny",
    font_size=80,
    color="#fff",
    text_align="center",
    duration=1,
)
author = TextClip(
    "./example.ttf",
    text="Blender Foundation",
    font_size=40,
    color="#fff",
    text_align="center",
    duration=1,
)
copyright = TextClip(
    "./example.ttf",
    text="© CC BY 3.0",
    font_size=20,
    color="#fff",
    text_align="center",
    duration=1,
)
logo = ImageClip("./example2.png", duration=1).resized(height=50)

# We want our title to be at the center horizontaly and start at 25%
# of the video verticaly. We can set as "center", "left", "right",
# "top" and "bottom", and % relative from the clip size
title = title.with_position(("center", 0.25), relative=True)

# We want the author to be in the center, 30px under the title
# We can set as pixels
top = background.h * 0.25 + title.h + 30
left = (background.w - author.w) / 2
author = author.with_position((left, top))

# We want the copyright to be 30px before bottom
copyright = copyright.with_position(("center", background.h - copyright.h - 30))

# Finally, we want the logo to be in the center, but to drop as time pass
# We can do so by setting position as a function that take time as argument,
# a lot like frame_function
top = (background.h - logo.h) / 2
logo = logo.with_position(lambda t: ("center", top + t * 30))

# We write the result
```

Skip to main content

```
final_clip = CompositeVideoClip([background, title, author, copyright, logo])
final_clip.write_videofile("final_clip.mp4")
```

When indicating the position keep in mind that the `y` coordinate has its zero at the top of the picture:



Adding transitions effects

The last part of composition is adding transition effects. For example, when a clip start while another is still playing, it would be nice to make the new one fade-in instead of showing abruptly.

To do so, we can use the transitions offered by MoviePy in **`transitions`**, like **`crossfadein()`** :

```
"""In this example, we will concatenate two clips with a 1-second
crossfadein of the second clip."""

from moviepy import VideoFileClip, CompositeVideoClip, vfx

# We load all the clips we want to compose
clip1 = VideoFileClip("example.mp4")
clip2 = VideoFileClip("example2.mp4")

clips = [
    clip1.with_end(2),
    clip2.with_start(1).with_effects([vfx.CrossFadeIn(1)]),
]
final_clip = CompositeVideoClip(clips)
final_clip.write_videofile("final_clip.mp4")
```

Skip to main content

MoviePy offer only few transitions in `transitions`. But technically, transitions are mostly effects applied to the mask of a clip! That means you can actually use any of the already existing effects, and use them as transitions by applying them on the mask of your clip (see .

For more info, see `transitions` and `moviepy.video.fx`.

Compositing audio clips

When you mix video clips together, MoviePy will automatically compose their respective audio tracks to form the audio track of the final clip, so you don't need to worry about compositing these tracks yourself.

If you want to make a custom audio track from several audio sources, audio clips can be mixed together like video clips, with `CompositeAudioClip` and `concatenate_audioclips()`:

```
"""Let's first concatenate (one after the other) then composite
(on top of each other) three audio clips."""

from moviepy import AudioFileClip, CompositeAudioClip, concatenate_audioclips

# We load all the clips we want to compose
clip1 = AudioFileClip("example.wav")
clip2 = AudioFileClip("example2.wav")
clip3 = AudioFileClip("example3.wav")

# All clip will play one after the other
concat = concatenate_audioclips([clip1, clip2, clip3])

# We will play clip1, then on top of it clip2 starting at t=5s,
# and clip3 on top of both starting t=9s
compo = CompositeAudioClip(
```