Kino

0.1.0 2025-9-12 MIT

https://github.com/aualbert/kino

aualbert

This document contains the documentation for the Typst package `kino`. It also provides instructions on how to use the companion Python script `kino.py`.

1. Structure of an animation

This section describes the basic structure of an animation. It focuses on the `animation` show rule and the functions `a`, `cut` and `finish`.

An animation is a typst file with the following structure:

```typst
#show: animation
// animation primitives & content
#finish()
```

Variables can be animated using animation primitives (e.g., `animate`), initialized using `init`, and their value accessed using `a`. The type of an animation variable cannot change during an animation. Supported types are `int`, `float`, `ratio`, `angle`, `array` of `function`. The size of an array and the types of its elements must be fixed. The functions must be defined at 0, and the type of its image cannot change, e.g.

```typst
#init(a: 0)
#init(r: (45%, .0))
#init(f: x => (y => x*y))
```

Animation variables can then be evaluated using `a`, passing the variable name as argument. Context must also be provided, e.g.

```typst
#context {
    a("a") + a("f")(.4)(.3)
}
```

To animate a variable from its current value to a new one, use an animation primitive (see Section 2). For example, the following primitives generate 1 second of frames at a given framerate. In this animation, `a("a")` successively evaluates to 0, 1 and 2. Meanwhile, `a(r)` interpolates continuously from `(45%, .0)` to `(60%, 1)`. Note that given the initial value of `r`, the argument `(60%, 1)` is interpreted as `(60%, 1.0)`.

```typst
#animate( a: 2)
#meanwhile( r: (60%, 1))
```

If you animate a non-initialized variable, the system infers an initial value of the correct type, e.g.

```typst
// the initial value
// of g is _ => (0.,)
#animate(g: t => (t,))
```

If a variable `x` is neither initialized nor animated, `a("x")` evaluates to 0%. Finally, the function `cut` splits the output into several segments. The exact semantics depends on the output format (see Section 4).

- `a`

Evaluates an animation variable in context.

Parameters

`a(name: str)`

- `animation`

The main show rule. Must be applied before any animation primitive is used. The body must contain a call to `finish()`.

Parameters

```
animation(
  body: content,
  fps: int
)
```

fps `int`

Frames per second of animation. Overrides command line parameters.

Default: `-1`

- `cut`

Add a cut at the end of the current block.

Parameters

```
cut(loop: bool)
```

loop `bool`

Whether the pre-cut segment should loop (revealjs only)

Default: `false`

- `finish`

Terminates the animation. Mandatory.

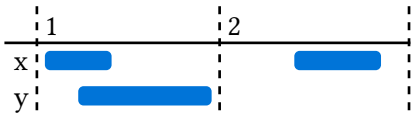
Parameters

```
finish()
```

2. Animation primitives

This section describes the different animation primitives. They roughly share the same parameters, so we describe only the `animate` primitive in detail.

Behind the scenes, the system converts each call to an animation primitive into a timeline. This timeline describes the value of animation variables at each time step of the animation. You can visualize the current timeline using `showtimeline()`:



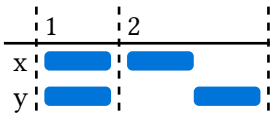
As seen above, a timeline is divided into blocks. Blocks become very useful when coordinating several animation variables. Any call to `animate` creates a new block, but you can also specify a block as a parameter. By default, the system inserts a cut between each block (see Section 4).

Finally, when calling an animation primitive, you can specify how variables are interpolated using the `animation` parameter (see Section 3 for details). You can also specify the duration of the animation in seconds.

- `animate`

Animate variables in a new block, or in the specified block. Changes the current block.

```typst
#animate(x:50%, y:3cm)
#animate(x:20%)
#animate(block:2, y:4cm)
```



Parameters

```
animate(
  block: int,
  hold: second,
  duration: second,
  dwell: second,
  transition: transition str,
  ..args
)
```

block `int`

A block identifier to start animation at.

Default: `-1`

hold `second`

Waiting time before animation.

Default: `0`

duration `second`

Duration of the animation.

Default: `1`

dwell `second`

Waiting time after animation.

Default: `0`

transition `transition` or `str`

A transition name or custom transition.

Default: `"linear"`

- `init`

Initialize one or several animation variables.

Parameters

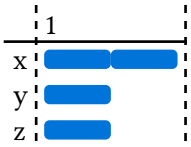
```
init(..args)
```

- `meanwhile`

Animate variables at the start of the current block, *if there is no collision.*

```
#animate(x:1)
#then(x:2)
#meanwhile(y:1)
#meanwhile(z:3%)
//#meanwhile(y:2) raises an error
```

typst



Parameters

```
meanwhile(
  hold: second,
  duration: second,
  dwell: second,
  transition: transition str,
  ..args
)
```

- `then`

Animate variables in the current block.

```
#animate(x:1)
#then(x:2)
#then(y:1)
```

typst

Parameters

```
then(
  hold: second,
  duration: second,
  dwell: second,
  transition: transition str,
  ..args
)
```

- `wait`

Add waiting time in the current or specified block.

Parameters

```
wait(
  block: int,
  duration: second
)
```

3. Transitions

This section describes the built-in transitions used by the animation primitives. A transition is a mathematical function $[0, 1] \to [0, 1]$. Wherever a built-in transition name is expected, you can provide a custom transition instead. In addition, you can concatenate transitions using `concat`.

- `circ`

Circular transition (square root)

Parameters

```
circ(t)
```

- `concat`

Concatenates two transitions

Parameters

```
concat(
  trans1: transition,
  trans2: transition
)
```

- `cubic`

Cubic transition (power of three)

Parameters

```
cubic(t)
```

- `linear`

Linear transition

Parameters

`linear(t)`

- `quad`

Quadratic transition (power of two)

Parameters

`quad(t)`

- `quart`

Quartic transition (power of four)

Parameters

`quart(t)`

- `sin`

Sine transition

Parameters

`sin(t)`

4. Export tool

You can use this package alongside the Python script kino.py, found at https://github.com/aualbert/kino. The requirements are:

- `python3`
- `pypdf`
- `ffmpeg`
- `typst`

A nix flake is also provided for convenience. The script exports animations to static slides, videos, or revealjs presentations. Refer to Figure 1 for the syntax of the program kino.py. This section describes the different arguments.

- `INPUT` Input file, either a single scene (a typst file) or a list of scenes to be played in order (a toml file with the following syntax):

```toml
scenes = ["scene1.typ", ...]
```

- `h` Displays help.
- `ROOT` Root of the project, passed to `typst`.
- `TIMEOUT` Timeout for each operation (`typst` or `ffmpeg`)

The `OUTPUT` can be one of `video`, `revealjs`, or `slides`. Using the `slides` option produces a PDF without animations. Using `video` supports any common video format. Using `revealjs` outputs a reveal.js presentation as a single HTML file with embedded videos. A valid reveal.js installation is still required. The following sections describe option-specific arguments.

Video and reveal.js export

The following options are exclusive to video and reveal.js export.

- `CUT` When to cut.

Cut animations produce multiple videos or steps in revealjs presentations. Manual `cut()` calls are never overridden. Can be `none` (no additional cuts besides `cut()`), `scene` (between each scene when using a `.toml` as input), or `all` (between each scene and each block).

- `FPS` Frame per seconds.

Does not override the parameters of the `animation` show rule.

- `PPI` Pixel per inches.

Video export

- `FORMAT` Format of the output video.

Reveal.js export

- `TITLE` Title displayed in browser.
- `--progress` Enables a progress bar.
- `TEMPLATE` A custom reveal.js template.

See for example the default template at `bin/revealjs.html`.

```
kino.py [-h]                INPUT {video|revealjs|slides} [–cut {none|scene|all}]
        [–root ROOT]                                      [–fps FPS]
        [–timeout TIMEOUT]                                [–ppi PPI]
                                                          [–format FORMAT]
                                                          [–title TITLE]
                                                          [–progress]
                                                          [–template TEMPLATE]
```

Figure 1: Command-line syntax of kino.py