Slidev Ultracharger

A doc/demo for the ultracharger set of Sli.dev addons.

Each feature is illustrated in its own part.

Some guidance is also given about usage/limitations/evolutions/improvements/TODO/etc.


NEXT FEATURE
a/z keys for slide browsing

a/z keys for slide browsing

Use `a` and `z` to navigate from slide to slide, showing the slides as they look after all clicks (animations) contrary to up/down arrows.

- defined in `./setup/shortcuts.ts`
- NB: the two next slides are just for you to try these `a` and `z` and how they differ from the default 4 arrow keys

Trying a/z: Slide 1

- Step 1.1
- Step 1.2
- Step 1.3
- Reminder:
  - `→` step forward
  - `←` step backward
  - `z` jump to next slide's end
  - `a` jump to previous slide's end
  - `↓` jump to next slide's beginning
  - `↑` jump to previous slide's beginning

Trying a/z: Slide 2

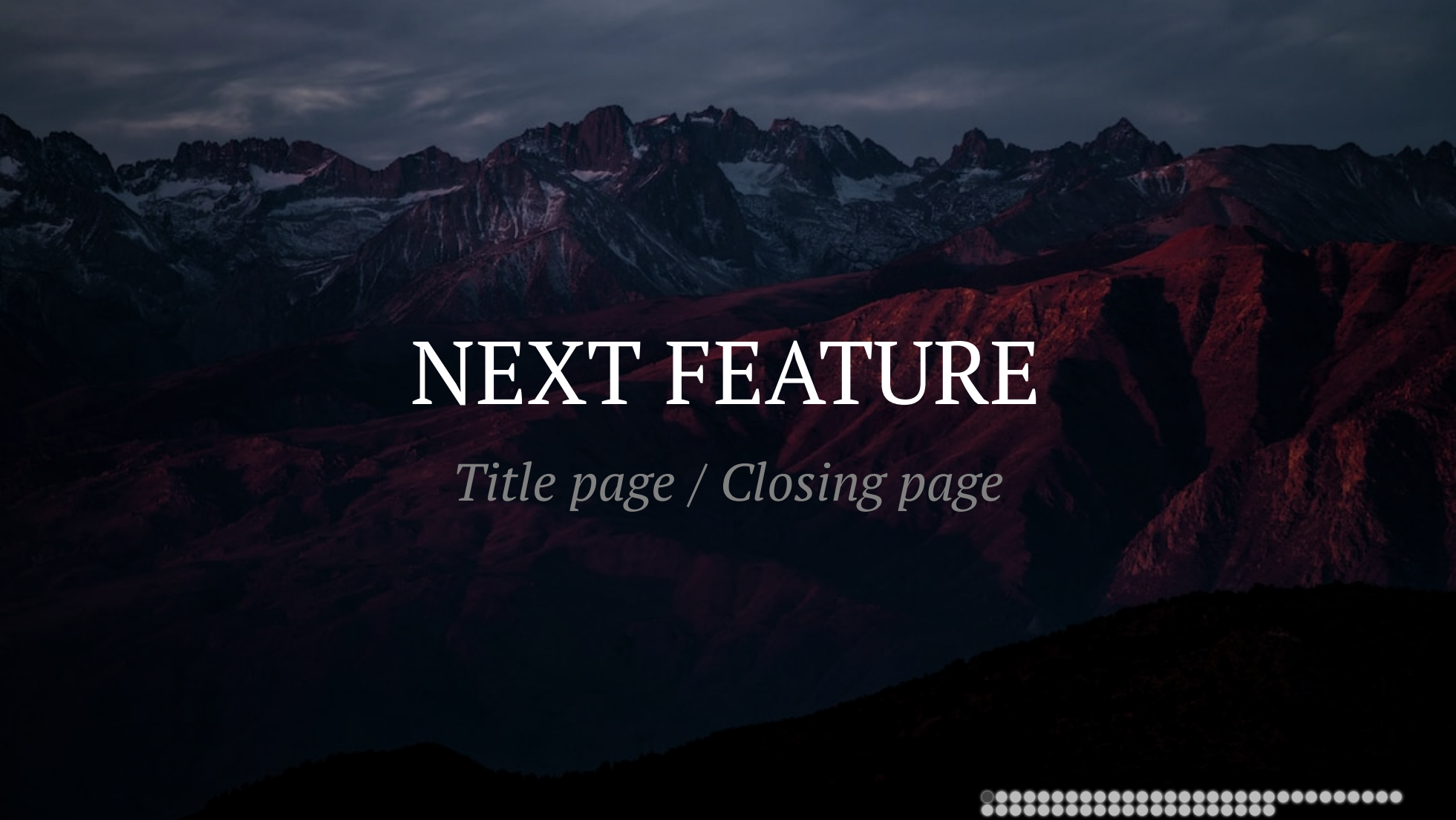
- Step 2.1
- Step 2.2
- Step 2.3
- Reminder:
  - `→` step forward
  - `←` step backward
  - `z` jump to next slide's end
  - `a` jump to previous slide's end
  - `↓` jump to next slide's beginning
  - `↑` jump to previous slide's beginning


NEXT FEATURE
b key to blackout

b key to blackout

Use `b` to blackout remote clients' views (from presenter).

- defined as a component in `./components/Blackout.vue`
- instantiated by default in `./global-top.vue`
- can be disabled with `addonsConfig: { ultracharger: { disable: ['blackout'] } }` (copy)
- registration of the keyboard shortcut in `./setup/shortcuts.ts`
- NB: to test, open a client and a presenter view and press `b` in the presenter view.
- NB: the presenter view is **not available** if you view from a hosted (e.g. github pages) version


NEXT FEATURE
Title page / Closing page

Title page / Closing page

In a slide, one can use metadata (defined in the headmatter).

This allows to regroup some kind of global variables at the beginning of the file, in the headmatter.

These can then be used and reused in the presentation, without needing to replace everywhere.

This can be done either in the native slidev way or using a helper component. (see the links in the next two slides to compare both approaches)

Slidev Ultracharger

...used as both a test page
and a quick documentation

2022-11-23
Rémi Emonet
Online

Here, we access the config variables (headmatter) directly in the slide, see `./example-title-manual.md`

Slidev Ultracharger

...used as both a test page
and a quick documentation

2022-11-23
Rémi Emonet
Online

Here, we some helper component to access the config variables (headmatter), see `./example-title-component.md`


NEXT FEATURE
Default Footer

Default Footer

Which also displays metadata (see any footer)

The addons also comes with a **default footer** with

- a centered footer with some metadata and the slide number
  - can be disabled with `addonsConfig: { ultracharger: { disable: ['metaFooter'] } }` (copy)
- a right footer with the TOC as dots
  - can be disabled with `addonsConfig: { ultracharger: { disable: ['tocFooter'] } }` (copy)
- defined in `./global-top.vue`

The footer can be tuned with

- `addonsConfig: { ultracharger: { fakeEnds: '5 -2' } }` (copy) with a list of slide index to use as fake ends, negative numbers being counted from the end (e.g. -2 if you have 2 extra slides to hide)
- `addonsConfig: { ultracharger: { metaFooter: 'date name' } }` (copy) (space-separated list of metadata)


NEXT FEATURE
Make KaTeX remember definitions

Make KaTeX remember definitions

With proper configuration, one can use gdef in KaTeX/LaTeX code and reuse them in other blocks or even slides.

- defined in `./setup/katex.vue` (might work by default in some contexts?) (but it is still cool)

Also have KaTeX errors inline in color as in $a^2 + 2\ab + b^2$.

KaTeX memory across slides

We can have a KaTeX block with our macro definitions

$$\text{we can define macros and use them in a block: } X, y$$

And use them in our text X, y

or in some block

$$\text{we can use in a block: } X, y$$

KaTeX memory across slides

even in another slide, we can reuse definitions X, y


NEXT FEATURE
SVG Inlining

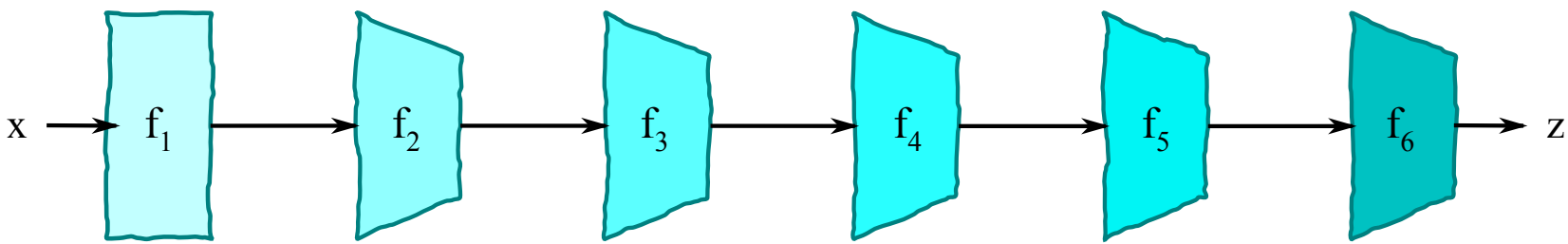
SVG Inlining

Loading SVG into the DOM

Use like an `img` but it is inlined: the SVG ends up in the DOM.

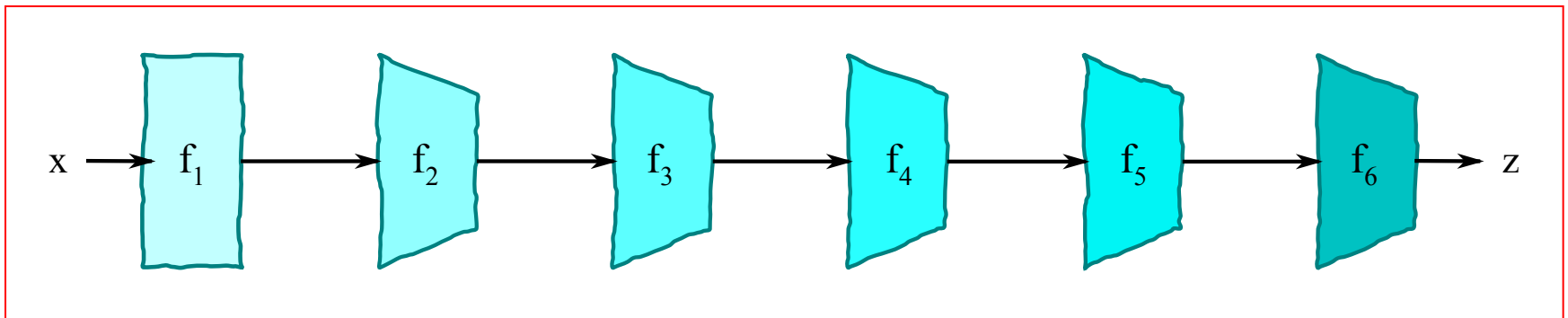
This allows CSS styling and js interactions.
For instance, we set css :hover to blur.

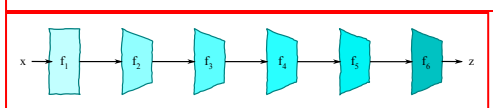
Move the pointer on any part of the image to test.

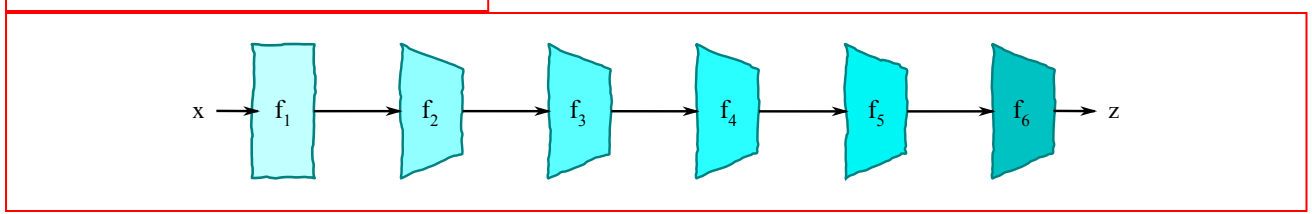


InlineSvg, specifying dimensions

The height and/or width can be specified, and will get inserted in the root `svg` element

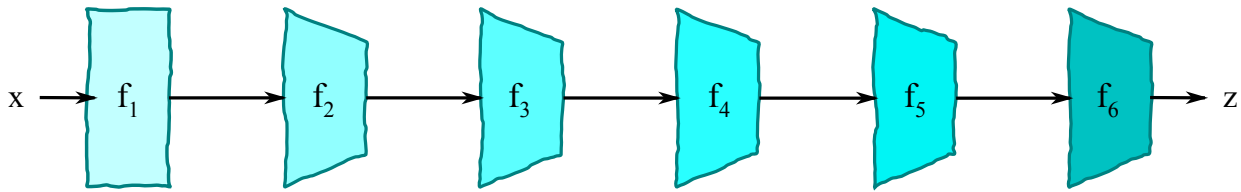




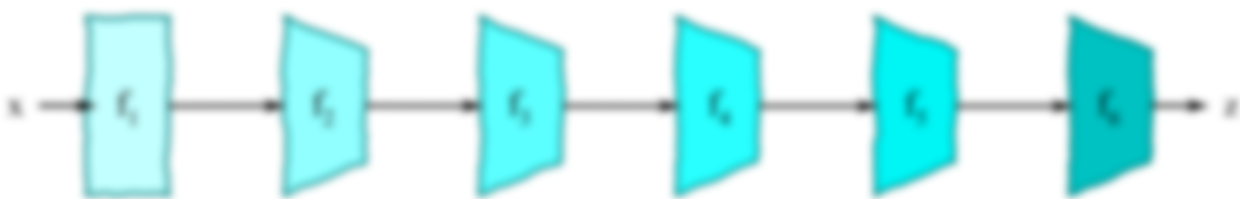


InlineSvg, wrapping

By default, the `InlineSvg` directly ends up as `svg` (with no added properties)



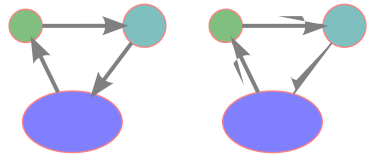
… `<InlineSvg wrap src=.../>` wraps it (inside a `div`), that inherits the properties (e.g. a `blurry` class)



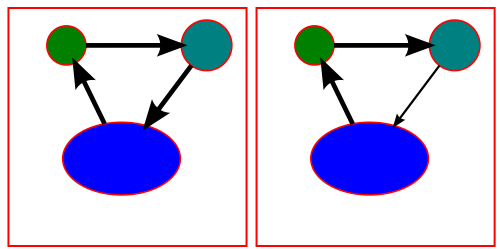
NB: height and/or width are applied to the `svg` element

InlineSvg, automated id rewrite (et al.)

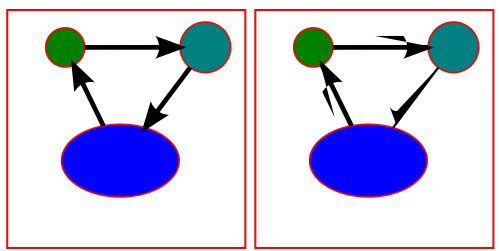
Two SVG images, here as `img`, with funky arrows in the second one.



As inlined SVG (both use the same ids, so there is a contamination problem)



Fixed using idRewrite (set by default)



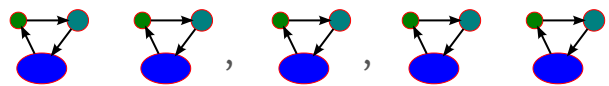
(more on the next slide)

InlineSvg, options and workarounds

Configuring the InlineSvg element.

In addition to width/height/wrap attributed, more advanced options like `idRewrite` can be specified as

- `<InlineSvg :opts="{...}" />`
- a list of options can be found at `./components/InlineSvg.vue`

NB: In case **missing arrows in your SVG** 

- set `:opts="{markersWorkaround: true}"`
- this is caused for instance by Inkscape exporting with SVG2 constructs (context-fill, context-stroke), that are not handled by browsers (chrome ignores it, firefox do not render markers at all).

Default options can also be set globally in your presentation with

(copy)

```
addonsConfig: { ultracharger: { inlineSvg: { markersWorkaround: true } } }
```


NEXT FEATURE
Click animations (fine grained clicks)

Click animations (fine grained clicks)

Animating (clicks) more precisely.

v-clicks every="2"

(feature from slidev)

Can make children appear several at a time.

(demo content) E.g., slidev is

- 📝 **Text-based** - focus…
- 🎨 **Themable** - theme …
- 👩‍💻 **Developer Friendly** - code highlighting, …
- 🤹 **Interactive** - embedding Vue components …
- 🎥 **Recording** - built-in recording and camera view
- 📤 **Portable** - export into PDF, …
- 🛠️ **Hackable** - anything possible on a webpage

They appear 2 by 2

```
spec="li:nth-child(2n) | strong | h1 | li:nth-child(2n+1),a"
```

(it uses a custom syntax to decide what to show, step by step, steps are separated by `|`,
each step is here a css selector of the elements to show at this step
these elements are automatically hidden at the beginning of the slide

(demo content) Slidev is

Read more about

NB:

- defined in `./components/Anim.vue`
- some style in `./style.css`
- more features in the next slides (and in section animating SVG)

`<Anim>` : Hide things, add/remove classes

spec=

```
"-.test | @+class bg-red-500 li | @+class blurry li li, code | @-class bg-red-500 li"
```

- NB: we can add a `-` to hide an element , e.g. this one
- NB: bg-red-500 is a class that comes from windicss, blurry is defined in the slide
- NB: no problem with css selectors that contain spaces like `li li, code`
- test
  - test2
  - test2
- test

`<Anim>` : several actions in a step/click

spec=
`"a | -a ^ strong | -strong ^ em ^ @+class blur div | a,strong ^ @+class blur strong,em"`

some div here

A link (a)

Some bold (strong)

Some italic (em)

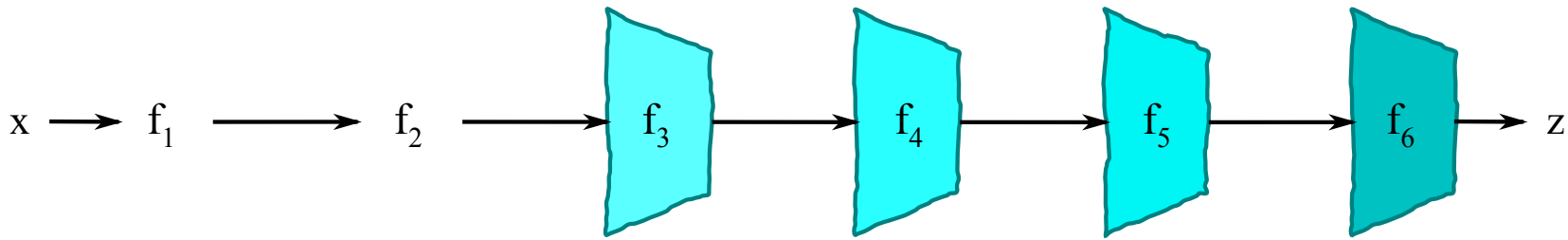

NEXT FEATURE
SVG animations (and related)

SVG animations (and related)

(using `<Anim>` and `<InlineSvg>`)

`<Anim>` works with SVG elements

```
spec="#rect846 | #path930 | @+class blur text | @-class blur #text26519"
```

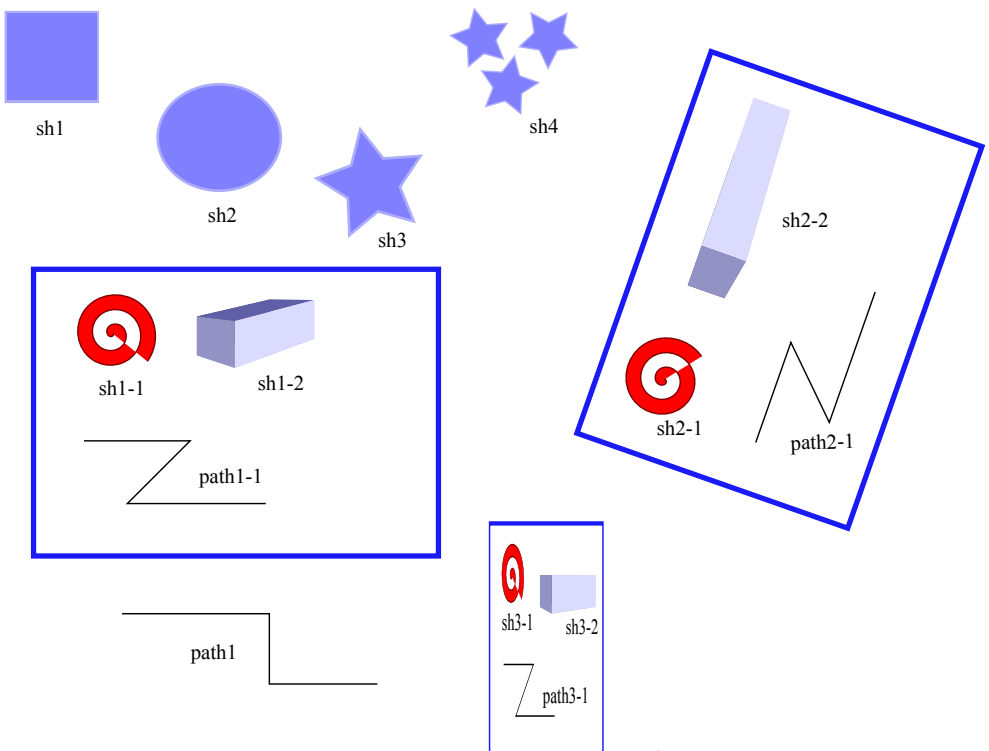


`<Anim>` allows any mix

```
spec=".svg2 | .svg1 | #fr1,#fr2 | .svg1 #fr3 | .svg2 #fr3 | -text | em"
```

`<Anim>` : move along SVG path

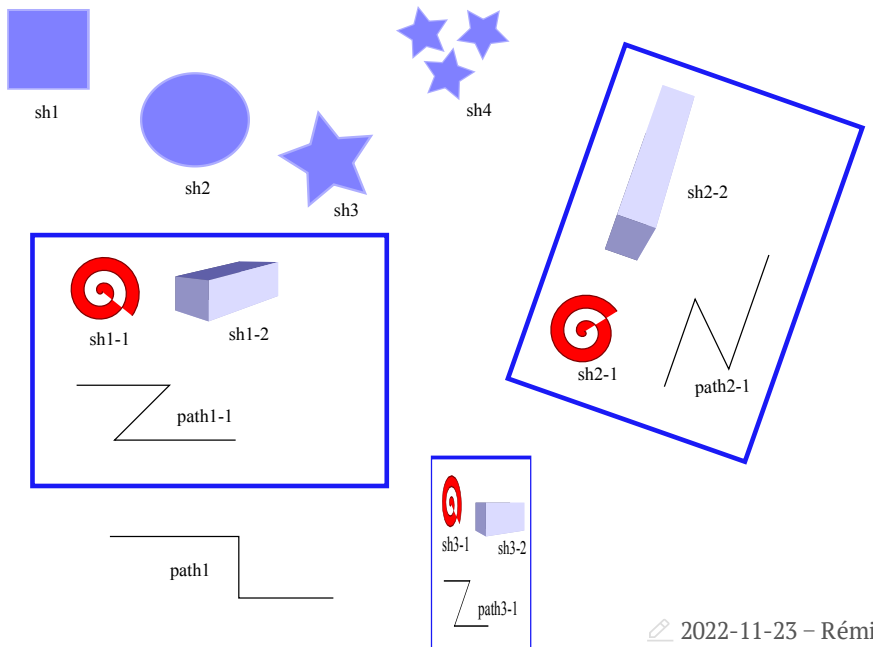
```
spec="@along #path1 #sh1,#sh2 | @along #path1 #sh3 | @along #path1 #sh1-1,#sh2-1,#sh3-1"
```



`<Anim>` : move along, control speed

```
dur="700ms"
spec="@along #path1 #sh1,#sh2 | @alongd #path1 2.5s #sh3 | @along #path1 #sh1-1,#sh2-1"
```

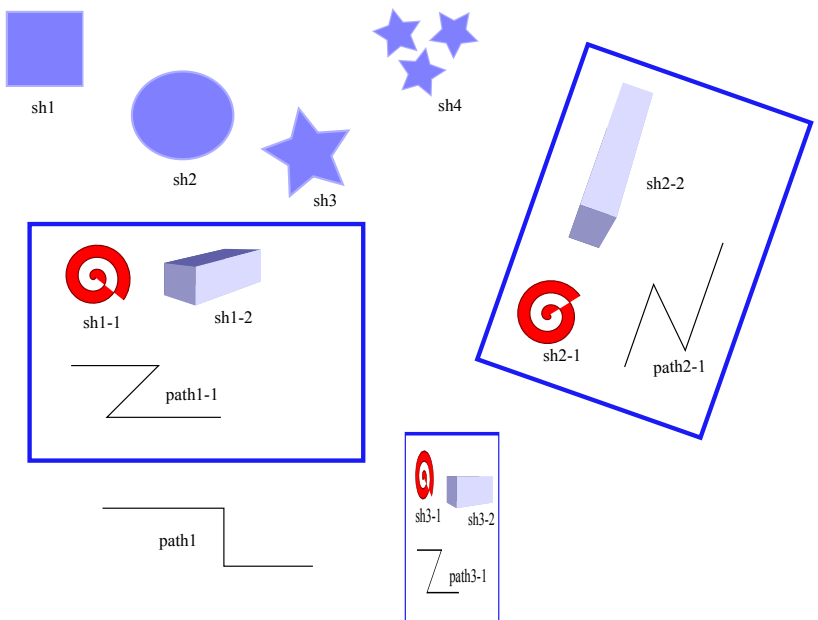
`dur="..."` to set the default duration + `@alongd` instead of `@along` to specify a duration for the step.



`<Anim>` : successive move along and backward move

```
spec="@along #path1 #sh1,#sh3-1  |@along #path1 #sh1,#sh3-1
     |@along -#path1 #sh1,#sh3-1 |@along #path1 #sh1,#sh3-1
```

NB: newlines are ok in `spec`



`<Anim>` : move along on non-SVG elements

```
spec=" @along #path1 .stuff |@along #path1-1 .stuff |@along #path1-1 .stuff"
```

NB: still needs an SVG path, also svg vs html units can be tricky.

... stuff ... evolve - 0 + ...


NEXT FEATURE
Playing with the (SVG) viewBox

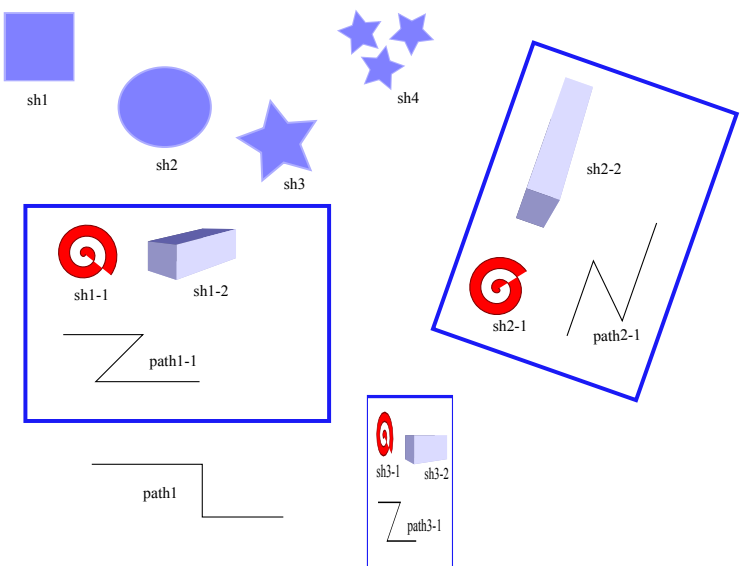
Playing with the (SVG) viewBox

(still using `<Anim>`)

`<Anim>` : Playing with the SVG viewBox, zooming

```
spec="@viewbox #fr1 | @alongd #sh1-1 1500 #sh1-2
    | @viewbox #rect6173 | @viewbox svg | @viewbox #rect6173
    | @viewbox #sh3-1 | @viewboxd #sh1-2 1500 | @viewbox svg"
```

NB: using `@viewbox` for default duration or `@viewboxd` to specify the duration.



`<Anim>` : TODO

- TODO: allow specifying ease-in-out etc (like duration)
- TODO: ^ maybe need special steps that set defaults for all coming steps (as in the original @anim)
- TODO: not necessarily possible: fix the @steps issue with opacity animation on nested elements
- TODO: might use :nth-child(… of …) when/if available in browsers
- TODO: implement viewBox-like anim for non-SVG


NEXT FEATURE
Marker-based steps/clicks

Marker-based steps/clicks

(still using `<Anim>`)

- defined in `./components/Anim.vue` with some CSS in `./style.css`

Marker-based steps, raw, verbose

```
spec="@step 1 | @step 2 | @step 3 | @step 4 | @step 5
    | -strong | @step 6 | @step 7 | @step 99"
```

NB: using class "step" on the marks
NB: here some CSS to show the mark
NB: the content starts hidden (low opacity)
NB: any big number is ok to show everything

- first level
- hum
- ok now we'll **nest**
  - nested 1
  - nested 2 and 2.5
  - and 3
- voila!
- tada!

Marker-based steps, control what is initially shown

```
spec="@step 1 | @step 2 | -strong | @step 42"
```

NB: use a mark with class `step0` (or just `<s0/>`) to decide up to where it is initially shown

- first level
- hum
- ok now we'll nest
  - nested 1
  - nested 2 and 2.5
  - and 3
- voila!
- tada!

Marker-based steps, `@steps` shortcut + special `&|`

```
spec="@steps 1-4 | -strong | @steps 5-"
```

NB: can use `@steps` and a range to simplify writing several `@step`
NB: can use an open range as in `@steps 5-` to go until the end
NB: spec defaults to `@steps 1-`
NB: (TODO UPDATE) can use `&|` to insert a span with class `step` (implemented in `./vite.config.ts`)

- first level
- hum &|
- ok &| now we'll **nest**
  - nested 1 &|
  - nested 2 &| and 2.5 &|
  - and 3 &|
- voila! &|
- tada!


NEXT FEATURE
Katex "align" (equation block) and code animation

Katex "align" (equation block) and code animation

```
spec="@maths 1-"
```

`<Anim>` katex, specifying a context

```
spec="@mathsc 1- .my>:nth-child(1) | .my>:nth-child(2) | @mathsc 1-3 .my>:nth-child(3)"
```

`<Anim>` code, with context (just code, no codec) (highlight is on next slide)

```
spec="@code 1,3 .c2 | @code 1- .c1 | @code 2 .c2"
```

And more code later on the left

Some code later on the right

`<Anim>` code line highlight, with context

```
spec="@code {1;3} .c2 | @code {3-} .c1 | @code {2} .c2"
```

And more code later on the left

```
function add(
  a: Ref<number> | number,
  b: Ref<number> | number
) {
  return computed(() => unref(a) + unref(b))
}
```

Some code later on the right

```
h1 {
  color: red
}
```


NEXT FEATURE
Anim Layout (shortcut)

Anim Layout (shortcut)

Defined in `layouts/anim.vue`

It wraps the content into an anim, and expects a `spec` header as in

```
layout: anim
spec: '@step 1 | @step 2 | -strong | @step 42'
```

- first level
- hum
- ok now we'll nest
  - nested 1
  - nested 2 and 2.5
  - and 3
- voila!
- tada!