Enhanced BEAMER increments: the `beamincr` package

Maneesh Sahani

October 9, 2023

The BEAMER class provides a number of powerful mechanisms to compose and control "overlays" or "animations," where a single complex slide is built up (or changed) piece by piece. While powerful, these mechanisms may sometimes be usefully extended. This package provides some such enhancements.

1 Background: overlays and increments

The basic BEAMER display unit is the `frame`. A frame may be rendered step-by-step, in which case the individual versions of the frame are called "overlays" or "slides". We will use these terms interchangeably. BEAMER allows you to place material on an arbitrary slide in a frame like this

Example:

```
\begin{frame}
  text on slides 1 and up\\
  \onslide<2->
  text on slides 2 and up\\
  \onslide<3-4>{
  text only on slides 3 and 4\\
  }
  \only<5>{text only on slide 5\\}
  more text on slides 2 and up\\
\end{frame}
```

You can read about the differences between `\onslide` and `\only`, and the many other overlay-sensitive commands, in the BEAMER user guide. Note in particular the difference between the argument form and the declaration forms of `\onslide`. `\only` only works with an argument.

This explicit numbering approach becomes burdensome when you want many overlays. You have to keep track of the numbers explicitly, and if you subsequently add a step early in the sequence you need to re-number the rest. Thus, BEAMER also provides an incremental overlay specification. The following code will produce the same effect as that above.

Example:

```
\begin{frame}
  \resetincr % not standard BEAMER
  text on slides 1+\\
  \onslide<+->
  text on slides 2+\\
  \onslide<+-+(1)>{ % increments counter by 1, despite the two +s
  text only on slides 3-4\\
  }
  \onslide<+>{} % increment counter by another
  \only<+>{text only on slide 5\\}
  more text on slides 2+\\
\end{frame}
```

This form allows easy automation using default overlay specificiations. For instance (from the BEAMER user guide)

Example:

```
\begin{itemize}[<+-| alert@+>]
\item Apple
\item Peach
\item Plum
\item Orange
\end{itemize}
```

There are important and sometimes not-entirely-intuitive differences between the incremental and explicit numbering systems. So we will refer to the steps implied in this way as "increments". They will mostly match slide numbers, but not always, as this example shows:

Example:

```
\begin{frame}
  \resetincr % not standard BEAMER
  text on slide 1+\\
  \onslide<3>{text on slide 3}\\
  text on slide 1+\\
  \onslide<+->
  text on slide 2+\\
  \onslide<4->
  text on slide 4+\\ % increment number is still 2!
  \onslide<+->
  text on slide 3+\\
\end{frame}
```

The increments have their own internal logic (specifically, their own internal counter) which is not affected by any explicit slide specifications that may appear in-between incremental calls. It may make sense to think of the increment number as being associated with *where* in the source file the material appears, rather than (necessarily) on *which slide* it appears.

There are a couple of oddities with the way increments work that often trip up first-time users. There are also some extensions that would be nice, like the ability to refer to a specific increment elsewhere in the frame. These things are certainly possible in stock BEAMER, but take some digging into internals. The tools here make things a bit easier.

As an aside, BEAMER has another incremental overlay system based on the `\pause` command. The `\pause` command uses the same counter as increments, but inteprets it slightly differently. This difference is discussed in Section 6. As a result, the two sets of specifications don't play very well together, at least from the viewpoint of non-experts. More on this below. I strongly suggest avoiding `\pause` entirely.

2 Setting increments

`\resetincr[`⟨*incr*⟩`]`

This command resets the increment number to 1, or to the value defined by ⟨*incr*⟩ if given. It doesn't directly affect the slide on which subsequent text appears, but it does change effect of subsequent `<+>` or `<.>` increments. The command may be useful to synchronise overlays in (say) two columns or between highlighted bullet points and highlighting in a figure.

Example:

```
\begin{frame}
  \resetincr
  \begin{center}
    Two lists \onslide<+->{in sync}
  \end{center}
  \begin{columns}
    \begin{column}{.2\textwidth}
      \begin{itemize}[<+-| alert@+>]
      \item Apple \item Peach \item Plum \item Orange
      \end{itemize}
    \end{column}
    \begin{column}{.2\textwidth}
      \resetincr[2] % restart the increment counter to sync
      \begin{itemize}[<+-| alert@+>]
```

```
      \item green \item yellow \item purple \item orange
      \end{itemize}
    \end{column}
  \end{columns}
\end{frame}
```

The argument ⟨*incr*⟩ must either be a number or be an increment specification enclosed in `/ /` (these are defined in Section 3). It cannot specify any sort of range, or be `+` or `.`, although `/./` and things like `/.(2)/` are allowed.

It is useful to call `\resetincr` at the start of every increment-based slide (as we have in the examples here). This avoids some potentially confusing behaviour that comes from the way the increment counter is implemented in BEAMER:

Example:

```
\begin{frame}
  text on slides 1-\\
  \onslide<+->
  text still on slides 1-\\
  \onslide<+->
  text on slides 2-
  \resetincr\onslide<.->
  text on slides 1-\\
  \onslide<+->
  text on slides 2-
\end{frame}
```

The first call to `\onslide<+->` doesn't advance the slide, unless it has been preceded by a `\resetincr` (or another `\onslide<+->` or a `\pause`).

`\fromincr<`⟨*incr*⟩`>`

This is shorthand for

```
\resetincr[incr]
\onslide<.->
```

It can only be used as a declaration (not with an argument). The restrictions on ⟨*incr*⟩ are the same as above.

3 Labelling and referring to increments

In complicated frames, it may be useful to name certain increments for later reference. For instance, one might want to change a figure at certain steps while progressing through a list of bullet points. Or one might want to redisplay certain slides in the frame with `\afterframe` or `\handoutframe` (described below).

`\incrlabel{`⟨*label*⟩`}`

This command attaches the current increment number to the label ⟨*label*⟩. Once defined, the labelled increment can be recovered in (almost) any overlay spec using the constructs discussed below. The ⟨*label*⟩ can contain most characters, but should not start with a '`.`' or '`!`'.

`\incrref{`⟨*incr spec*⟩`}`

This command returns the increment number defined by increment specification ⟨*incr spec*⟩ as described below.

The general form of an increment specification is

`increment specification: label(offset)`

The label can be a string assigned by a call to `\incrlabel`, or the special label '`.`' which refers to the current increment (this is subtly different to the incremental overlay specification '`.`'). The ⟨*offset*⟩, if given, is added to the increment indicated by the label. It can be negative.

Increment specifications can be used as part of almost any overlay specification by enclosing them with slashes, e.g. `</mylabel(2)/>`.

An example:

Example:

```
\begin{frame}[label=twolists]
  \resetincr
  \begin{center}
    Two lists \onslide<+->{in sync}\\
    \onslide<+->{with more material\\}
    \onslide<+->{at the top}
  \end{center}
  \begin{columns}
    \begin{column}{.2\textwidth}
      \incrlabel{startlist}%
      \begin{itemize}[<+-| alert@+>]
      \item Apple \item Peach \incrlabel{halfway} \item Plum \item Orange
      \end{itemize}
    \end{column}
    \begin{column}{.2\textwidth}
      \resetincr[/startlist/]% keep in sync, even if we add extra topmatter
      \begin{itemize}[<+-| alert@+>]
      \item green \item yellow \item purple \item orange
      \end{itemize}
    \end{column}
  \end{columns}
  \vfill
  \onslide<+->
  The final increment is \incrref{.}.
  \incrlabel{end}
\end{frame}
```

Note that of commands discussed here, `\incrref` expects an increment specification (i.e. `label(offset)`), while `\fromincr` and `\resetincr` expect a restricted overlay specification that might include an increment specification (e.g. `/label(offset)/`) or just a number. Standard overlay-aware commands should all accept overlay specifications that include increment specs.

One BEAMER command (slightly patched in this package) with which named increments are particularly useful is `\againframe`. So

Example:

```
\againframe<1,/halfway/,/end(-1)/-/end/>{twolists}
```

provides an abbreviated tour of the lists. Increment labels are associated with the label of the enclosing frame, and so the same names can safely be reused across multiple named frames.

There is also a similar new command called `\handoutframe` to render more than one overlay from a frame in `handout` or similar modes that, by default, just show a single slide with all the overlays collapsed. See Section 5.

3.1 Setting increments in overlay specifications

It is possible to use the BEAMER syntax for actions in overlays to reset the increment number. This can be done using the explicit `resetincr@`⟨*increment*⟩ action or with an implicit `<!`⟨*increment*⟩`>` specification.

Example:

```
\action<3-|resetincr@3>{body}
\action<!3->{body}
```

The increment number can be replaced by a label, with optional offset:

```
\resetincr[2]\incrlabel{x}\resetincr[1]
\action</x/->{body on 2+}
\onslide<.->{this on 1+}
\action<!/x(2)/->{body on 4+}
\onslide<+->{this on 5+}
```

The forms `<!+>` and `<!.>` aren't supported (and wouldn't be useful: `<+>` already advances the increment, while `<!.>` would set it to its current value). However `<!/.(`⟨*offset*⟩`)/>` (note the `/` label notation) can be used to advance the increment counter by multiple (or negative) steps.

The reset only takes effect after the overlay specification has been interpreted and before the body is set. So any `+` or `.` references will be relative to the increment in effect *before* the `\action`.

Example:

```
\resetincr
\action<!/.(2)/-|alert@.>{alert too early}
\action<!/.(2)/->{\alert<.>{alert when uncovered}}
```

It's also possible to issue multiple `!` commands in one overlay spec but only the last of them will take effect.

According to the user guide, BEAMER actions are only supported by overlay specifications to `\action`, `\item`, the `actionenv` environment and block environments like `block` and `theorem`. In the absence of any action spefications, `\action` acts like `\uncover` (or `\onslide` with an argument). Note that `\fromslide` (Section 2) can act like an `\onslide` declaration.

4 $\mathcal{AMS}$-TEX align environments

The `align` and `align*` environments from the `amsmath` package are useful for multiple or multiline equations, but don't really work well with overlays. The current package introduces a partial fix for this, although their are remaining fragilities that need to be worked around. It should be straightforward to extend the approach used here to other `amsmath` equation environments if desired.

`\begin{incralign[*]}[<`⟨*spec1*⟩`>&<`⟨*spec2*⟩`>& ...]`
⟨*environment contents*⟩
`\end{incralign[*]}`

Pre-process the input to `align[*]` to place an `\action<>{}` command around each field, defined as the material appearing between successive `&` and/or `\\` tokens. By default, the first field on a line is called with `\action<+->{}`, and remaining ones with `\action<.->{}`. This has the effect of displaying a full line at a time. The optional argument makes it possible to change this to `\action<spec1>`, `\action<spec2>`, etc. with the sequence of specifications reset to `<spec1>` at the beginning of every line. There must be at least as many specifications given as their are fields used on the longest single line in the environment contents, but lines can have fewer fields. The default specification values can be changed by a calling `\incralignd efaultspec` as below.

The use of `\action` means that beamer will interpret both implicit increments and standard `action@n` actions.

It is also possible to override the default for a single field by starting it with the new specification. This means that if the field content itself starts with `<`, it needs to be protected, e.g. with a leading `{}`.

Example:

```
\begin{incralign*}[<+->&<.->&<+->&<.->] % both sides each eqns appear at once
  x\incrlabel{x} &= y  &  1 &{}< 2 \\
  </x/-> x^2 &= y^2  &  <!/x/-> e^{i\pi} &<+->= -1 \\
  \sum_n f(n) &\to \int f(x) dx
\end{incralign*}
(increment at this line = \incrref{.})\\
```

The pre-processor is not able to distinguish between the `&` alignment characters that apply to the containing environment and any that appear within (e.g.) enclosed `array` environments. So any such environments need to be protected using either a token register or a protected macro. Note that it is still possible to use increments within the environments. These are processed sequentially with those in the containing `align` environment, respecting increment labels, resets etc.

Example:

```
% using token registers
\newtoks\mymatrix
\mymatrix={\begin{pmatrix} 1 & 2 \\ \alt<+->{3}{2} & 4 \\ \end{pmatrix}}
\begin{incralign}
  \incrlabel{mat}\the\mymatrix \resetincr[/mat/]& \text{is \only<+->{not }singular}
```

```
\end{incralign}

% using \protected
\protected\def\mymatrix{\begin{pmatrix} 1 & 2 \\ \alt<+->{3}{2} & 4 \\ \end{pmatrix}}
\begin{incralign}
  \incrlabel{mat}\mymatrix \resetincr[/mat/]& \text{is \only<+->{not }singular}
\end{incralign}
```

It may be wise to put the `\newtoks` declaration outside the frame so as not to consume more of TEX's resources than needed. Also note that equation numbers aren't currently overlay aware.

The current version does not support use of `\intertext`, but I hope to fix this.

`\makealignincremental`

Make all subsequent uses of the `align` and `align*` environments automatically incremental. The original (non-incremental) forms are available as `amsalign[*]`.

`\makealignams`

Make all subsequent uses of the `align` and `align*` invoke their original non-incremental forms.

`\incralignd efaultspec[`⟨*new default*⟩`]`

With the optional arg, change the default overlay specification for subsequent `incralign[*]` environments (and `align[*]` if `\makealignincremental` has been called). Without the optional arg this restores the package default value which is `<+->&<.->&<.->&<.->&<.->&<.->&<.->`.

5 Misc helper functions

These functions may prove useful under some circumstances.

`\handoutframe[`⟨*mode spec*⟩`]<`⟨*overlay specification*⟩`>{`⟨*frame label*⟩`}`

This command only produces output when compiled in ⟨*mode spec*⟩ (which defaults to `handout`, but can include more than one, e.g., `[beamer|handout]`), in which case it renders the specified overlays from the named frame. If ⟨*overlay specification*⟩ is omitted, all the overlays are rendered. The code works by switching temporarily to `beamer` mode as that seems to be the only way to produce more than one overlay per frame in `handout`, `trans` and `article` modes, although this means it may behave poorly with any mode-specific material within the frame. The idea is from `https://tex.stackexchange.com/questions/455444/beamer-overlays-and-handout-exclude-frames-from-handout/455459#455459`.

Unfortunately, the natural code

Example:

```
\begin{frame}<handout:0>[label=twolists]
  ...
\end{frame}
\handoutframe<1,/halfway/,/done/>{twolists}
```

fails, because the `<handout:0>` spec stops the increment labels from being defined. If running a recent LATEX compiler (post 2021) the command `\framescanonly<handout>` described below provides a workaround.

Example:

```
\begin{frame}[label=twolists]
  \framescanonly<handout|trans>%
  ...
\end{frame}
\handoutframe[handout|trans]<1,/halfway/,/done/>{twolists}
```

`\framescanonly<`⟨*mode specification*⟩`>`

Scan the current frame without producing any output. This is similar to a `<mode:0>` specification to `\begin{frame}`, but as the frame is still scanned it allows side effects such as increment label definitions. The `\frameonly` command should be placed inside the frame contents. It is only available in recent

versions of LaTeX with hook support; a warning is printed in other cases. If used in `beamer` mode the frame will be reprocessed for every overlay. This behaviour can be avoided by also including a `<beamer:1>` or `<beamer:-1>` (but not `<beamer:0>`!) or equivalent specification to `\begin{frame}`.

An example of its use appears above.

`\parseincrspec{`⟨*overlay spec*⟩`}`

Interpret any text enclosed in `/`s within ⟨*overlay spec*⟩ as an increment specification, replacing them with the corresponding numerical values.

Example:

```
\begin{frame}
  \resetincr[2]
  \incrlabel{two}
  \parseincrspec{/two/-/two(3)/} % prints 2-5
\end{frame}
```

6 Internals

These sections discuss more background and some implementation details. This is only likely to be of interest to users who wish to extend the approach.

6.1 Pauses and increments

Both `\pause` and incremental overlay specifications access the same underlying counter called `beamerpauses`, but they use them in different ways.

`\pause`

increments `beamerpauses` and then sets subsequent material on the slide given by the incremented `\value{beamerpauses}`.

`\onslide<+->`

increments `beamerpauses` but then sets subsequent (or argument) material on the slide corresponding to the *previous* value of `beamerpauses`.

`\onslide<.->`

leaves `beamerpauses` alone, but sets subsequent (or argument) material on the slide given by `\value{beamerpauses}-1`, unless `\value{beamerpauses}=1` in which case it puts subsequent material on slide 1.

This conflict in interpretation of the `beamerpauses` counter can cause unintuitive effects. The incremental specfication model is far more flexible and powerful, and so the commands of this package can all be interpreted in terms of an *increment number* which equals `max(\value{beamerpauses}-1,1)`. In fact, they all use the `beamerpauses` counter with this offset. Thus, when commands like `\resetincr` set the increment value, they set `beamerpauses` to the increment + 1. This value then behaves sensibly with `<+>` etc. specifications, but not with `\pause`.