The spot package: spotlight highlighting for Beamer*

Anders Hendrickson
Concordia College, Moorhead, MN
ahendric@cord.edu

September 21, 2010

1 Introduction

Beamer's `\alert` command is designed to call the viewer's attention to certain parts of the slide, but under some circumstances it can be less effective than desired. First, because the actual colors produced depend on the projector hardware used and the ambient lighting in the room, you can find to your dismay that the color change provided by `\alert` is barely discernible onscreen. Of course you can fiddle with `\setbeamercolor{alerted text}{`⟨*color*⟩`}` to achieve better results on your own projector, but you may still face surprises when taking your presentation to another institution. Moreover, the `\alert` command is often ineffective at highlighting just one or two characters on a full slide; in such a situation, a color change alone may not be striking enough to draw the viewer's eye. For just such circumstances, this package provides a `\spot` command to paint a "spotlight" of color painted behind the highlighted text, as in this example.

2 Usage

2.1 General Usage

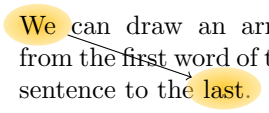
`\spot` The `\spot` command has the following syntax:

`\spot <`⟨*overlay spec*⟩`>(`⟨*node name*⟩`)[`⟨*node options*⟩`]{`⟨*highlighted text*⟩`}`

The first three parameters, `<`⟨*overlay spec*⟩`>`, `(`⟨*node name*⟩`)`, and `[`⟨*node options*⟩`]`, are optional, and any combination of them may be omitted, but their order must not change. For example, `\spot<5->[fill=red]{George}` is legal, but entering `\spot[star](mynode){Joe}` will produce erroneous output.

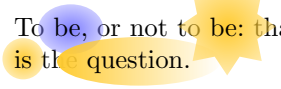
*This document corresponds to spotlight v1.0, dated 2010/09/21.

The ⟨*overlay spec*⟩ should be a standard Beamer overlay specification; for example, `\spot<2-3>{Fred}` highlights the word "Fred" only on slides 2 and 3. The actual "spotlight" shape is a Ti*k*Z node. If you wish to access the node later—for example, to point an arrow at it—you can specify a ⟨*node name*⟩. Consider the following example:




```
\spot(first){We} can draw an
arrow from the first word of this
sentence to the \spot(last){last}.
\tikz[remember picture, overlay]{
 \draw (first) to[->] (last);}
```

The default behavior of `\spot` is to surround its argument with a gold-colored ellipse, most intense at its center and fading towards the edges. That behavior can be altered—for example, to change the shape or color—for a specific instance of `\spot` by specifying ⟨*node options*⟩. Consider the following example:




```
To \spot*[fill=blue!50]{be,} or not
to \spot*[star, star points=8]{be:}
that \spot*[ball color=red]{is} the
\spot*[path fading=east]{question.}
```

For a complete understanding of possible ⟨*node options*⟩, please see the PGF/Ti*k*Z documentation. To change the default behavior for subsequent calls to `\spot`, the following commands are provided.

`\setspotlightcolor` `\resetspotlightcolor` `\spotlightcolor`

The command `\setspotlightcolor{`⟨*color*⟩`}` makes all subsequent invocations of `\spot` use the specified ⟨*color*⟩. The default shade of gold is named `spotlightgold`, and may be restored with the command `\resetspotlightcolor`. The current color is saved in the macro `\spotlightcolor`, making commands such as `\setspotlightcolor{\spotlightcolor!50}` possible. Note that if `fill=`⟨*color*⟩ is given in the ⟨*node options*⟩ or with `\setspotlightstyle`, it takes precedence over the ⟨*color*⟩ in `\spotlightcolor`.

`\setspotlightstyle` `\resetspotlightstyle` `\spotlightnodeoptions`

The command `\setspotlightstyle{`⟨*node options*⟩`}` adds ⟨*node options*⟩ to the nodes produced by all subsequent invocations of `\spot`. The effect of multiple calls is cumulative. Default options may be restored by `\resetspotlightstyle`. The current options are saved in the macro `\spotlightnodeoptions`.


```
\setspotlightcolor{red!50}
\spot{A} \quad
\setspotlightstyle{star, fill=green!50}
\spot{B} \quad
\setspotlightstyle{star points=7}
\spot{C} \quad
\resetspotlightstyle
\spot{D} \quad
\resetspotlightcolor
\spot{E}
```

2.2 Non-Beamer Usage

Although the `spot` package is chiefly designed for use with Beamer, it may also be used in documents of other classes. There are two peculiarities, however, the reasons for which will be explained in section 3. First, if `\spot` is used within a `\parbox`, a `minipage` environment, a header or footer, or a `p` column in a `tabular` environment, the error message "`LaTeX Error: Float(s) lost`" will result. The remedy is to use the the starred version `\spot*` instead, and to follow the instructions in the following paragraph.

`\spot*`

`\dospots`

Second, if the last call to `\spot` on a page is in math mode, in a TEX inner mode (such as a `tabular` environment), or the starred version `\spot*`, then the command `\dospots` must be issued somewhere later on that same page, after the math mode or `tabular` or `\parbox` has ended. (The actual criterion is that `\dospots` must be able to call `\marginpar`, which cannot be done in those environments.) If the page ends before a `\dospots` is encountered, an error message will be generated.

`\dospotsheader`

`\dospotsfooter`

Remembering to place a `\dospots` after each problematic `\spot` can be a hassle. The best solution is to use the `fancyhdr` package to place the command `\dospotsheader` or its synonym `\dospotsfooter` into a header or footer appearing on every page, obviating the need for any manual `\dospots`. Moreover, if a spotlight is desired *within* a header, footer, or marginpar, `\spot*` should be used and `\dospotsheader` or `\dospotsfooter` should follow.

3 Strategy

This section describes the mechanism `spot` uses to draw its spotlights. It would be simple enough to wrap the highlighted text within a Ti*k*Z node, keeping it in line with its surroundings, but the result is not perfect. The preceding highlighting[1] was painted over previous text but under subsequent text, creating both an uneven effect and an unintentional emphasis on the beginning of the following word. The solution is to leave a blank space in the text, coming back once the entire frame has been typeset to draw the text and its highlighting over their surroundings.

Suppose then that `\spot<2>{foo}` is called in a Beamer frame. On slide 1 of the frame, the text `foo` is typeset as the sole node in a Ti*k*Z picture, with node op-

[1] Produced with

```
\tikz[baseline]{
  \path[use as bounding box]
    node[anchor=base, inner sep=0, outer sep=0, opacity=0]
     (temp) {highlighted text};
  \path (temp)
    node[anchor=center, outer sep=0,
         shape=ellipse, inner sep=0.5ex,
         fill=\spotlightcolor, path fading=spot@fade] {highlighted text};}
```

tions chosen so that it occupies its place in the paragraph just like ordinary typeset text. The node is given a name such as `spot@vii`, and the `remember picture` option is used.

When slide 2 is typeset, the node `spot@vii` is drawn as before, except with `opacity=0` so that it is transparent. After the contents of the entire slide are typeset, another Ti*k*Z picture uses the `overlay` option to return to the location of `spot@vii` and typeset `foo` in a node with a suitably eye-catching shape, `fill` color, and fading.

In Beamer, the code to paint the highlighted node is attached to `\end{frame}`. The challenge for implementing `spot` in classes other than Beamer is the lack of a hook for the end of a page; even the `afterpage` package cannot return to the previous page to typeset new material. The solution used is to insert the `\spot@paintspot` macro into a `\marginpar`. This is the source of the two peculiarities mentioned on page 3, for under certain circumstances, LaTeX cannot process a `\marginpar` correctly. These circumstances include TeX's inner vertical mode, display math, and `\parbox`es. In these situations `spot` must save its painting commands for later. The command `\dospots`, called from a "safe" location, inserts those saved commands into a `\marginpar`. The alternative commands `\dospotsheader` and `\dospotsfooter` execute the saved paint commands immediately, making them suitable for inclusion in a header or footer. Because this second mechanism, though more reliable, places an extra burden on the user, it was not chosen as the default behavior for `\spot`.

The starred version `\spot*` uses the second mechanism in place of the first. Moreover, the `\spot` macro will also switch to the second mechanism when it can detect the necessity. If `\ifinner` is true, then `\marginpar` would fail; likewise if `\ifmmode` is true, then LaTeX might well be in display math mode where `\marginpar` fails; in these situations, `\spot` acts like `\spot*`. Unfortunately, if `\spot` is called within a `\parbox`, there is no test that can make `\spot` aware of that fact, and the user will see `LaTeX Error: Float(s) lost.` This warning is an indication that even LaTeX itself could not tell in advance whether `\marginpar` would work properly or not.

4 Limitations and Tips

- Because `\spot{foo}` places the text `foo` in a Ti*k*Z node, it will not allow line breaks within `foo`. For highlighting that permits automatic line breaking, consider the `\hl` command of the `soul` package.
- The `\spot` command does not work well in a Beamer frame broken into multiple slides by `allowframebreaks`.
- In non-Beamer class, spotlights meant for the bottom of one page may sometimes show up at the bottom of the next page.
- It is possible to call `\spot` in math mode, but as the argument will be set in a box within a Ti*k*Z node, the results may need tinkering. For ex-

ample, `$a^{\spot{b}}$` produces $a^{\spot{b}}$, not $a^{\spot{b}}$; to make the superscript the correct size, you must write `$a^{\spot{\scriptstyle b}}$`. Likewise to produce $a\spot{=}b$ requires writing `$a\mathbin{\spot{=}}b$`, since merely writing `$a\spot{=}b$` produces the poorly spaced $a\spot{=}b$.

- As mentioned in section 2.2, when using `spot` in a document class other than Beamer, it is best to include `\dospotsheader` in a header appearing on every page. A minimal solution is to put the following in the preamble:

```
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhf{}
\rhead{\dospotsheader}
```

5 Implementation

5.1 Packages and Hooks

The Ti*k*Z package must be loaded with the necessary libraries.

```
\RequirePackage{tikz}
\usetikzlibrary{shapes}
\usetikzlibrary{fadings}
```

We next check whether we are in the Beamer class or not, and we abbreviate `\expandafter` in the usual way.

```
\newif\ifspot@beamer
\@ifundefined{beamer@frameslide}{\spot@beamerfalse}{\spot@beamertrue}
\let\xa=\expandafter
```

In Beamer, we will need hooks for running code at the beginning and end of each frame. The commands `\AtEveryBeginFrame` and `\AtEveryEndFrame` store commands to be run in every frame, while the command `\AtEndFrame` applies only to the current frame.

```
\ifspot@beamer
  \g@addto@macro\beamer@frameslide{\spot@everybeginframe@hook}
```

Whereas `\g@addto@macro` makes it easy to add `\spot@everybeginframe@hook` to the end of the `\begin{beamer@frameslide}` code, adding hooks to be executed right *before* the existing `\end{beamer@frameslide}` code is a little more involved.

```
  \let\spot@oldmaterial=\endbeamer@frameslide
  \def\spot@newmaterial{%
        \spot@endframe@hook%
        \gdef\spot@endframe@hook{}%
        \spot@everyendframe@hook}
  \xa\xa\xa\def
  \xa\xa\xa\endbeamer@frameslide
  \xa\xa\xa{%
        \xa\spot@newmaterial\spot@oldmaterial}%
```

Having placed hooks into the `beamer@frameslide` environment, we now initialize those hooks as empty and create the commands `\AtEndFrame`, `\AtEveryEndFrame`, and `\AtEveryBeginFrame`.

```
\def\spot@endframe@hook{}
\def\spot@everyendframe@hook{}
\def\spot@everybeginframe@hook{}
\long\def\AtEndFrame#1{\g@addto@macro\spot@endframe@hook{#1}}
\long\def\AtEveryEndFrame#1{\g@addto@macro\spot@everyendframe@hook{#1}}
\long\def\AtEveryBeginFrame#1{\g@addto@macro\spot@everybeginframe@hook{#1}}
```

In classes other than Beamer, we will need the `afterpage` package to run code between pages.

```
\else
  \RequirePackage{afterpage}
\fi
```

5.2 Spotlight options

We next define the options. The `\spotlightcolor` macro stores the working color; it can be edited with the commands `\setspotlightcolor` and `\resetspotlightcolor`.

```
\def\spotlightcolor{spotlightgold}
\def\setspotlightcolor#1{\xdef\spotlightcolor{#1}}
\def\resetspotlightcolor{\gdef\spotlightcolor{spotlightgold}}
```

Likewise the command `\spotlightnodeoptions` stores additional node options chosen by the user. The command `\setspotlightstyle` adds to it, and `\resetspotlightstyle` empties it.

```
\def\spotlightnodeoptions{}
\def\setspotlightstyle#1{\g@addto@macro\spotlightnodeoptions{#1, }}
\def\resetspotlightstyle{\gdef\spotlightnodeoptions{}}
```

Finally, here are the default color and fading.

```
\definecolor{spotlightgold}{RGB}{255,204,51}
\tikzfading[name=spot@fade,
            inner color=transparent!0,
            outer color=transparent!60]
```

The count register `\c@spot@spots` numbers each spotlight on the frame or page.

```
\newcount\c@spot@spots
\c@spot@spots=1
\ifspot@beamer
  \AtEveryBeginFrame{\global\c@spot@spots=1\relax}
\else
  \afterpage{\global\c@spot@spots=1\relax}
\fi
```

5.3 Input parsing

\spot The macro \spot comes in starred and unstarred versions. We first check for the presence of the star, and set a flag if it is present.

```
\newif\ifspot@star
\spot@starfalse
\def\spot{\@ifnextchar*{\spot@star}{\spot@nostar}}
\def\spot@star#1{\global\spot@startrue\spot@nostar}
```

\spot@nostar As described in section 2, the full syntax of \spot is

\spot <⟨*overlay spec*⟩>(⟨*node name*⟩)[⟨*node options*⟩]{⟨*highlighted text*⟩}

with each of ⟨*overlay spec*⟩, ⟨*node name*⟩, and ⟨*node options*⟩ being optional. If any is not specified, we fill in the appropriate default value.

The default overlay specification is <1->.

```
\def\spot@nostar{%
  \@ifnextchar<{\spot@in}{\spot@in<1->}%
}
```

\spot@in The default ⟨*node name*⟩ is spot@dummynode.

```
\def\spot@in<#1>{%
  \@ifnextchar({\spot@inte<#1>}%
               {\spot@inte<#1>(spot@dummynode)}%
}
```

\spot@inte The default is to have no extra ⟨*node options*⟩.

```
\def\spot@inte<#1>(#2){%
  \@ifnextchar[{\spot@intern<#1>(#2)}{\spot@intern<#1>(#2)[]}%
}
```

\spot@intern Because the spotlight is painted after the rest of the page is drawn, it is important to expand the node options now (including \spotlightcolor and the user-defined \spotlightnodeoptions), so that they will not be affected by redefinitions later on the page. Note that the order of parameters is changed to make the \expandafters work.

```
\def\spot@intern<#1>(#2)[#3]#4{%
  \edef\spot@totaloptions{fill=\spotlightcolor, \spotlightnodeoptions, #3}%
  \xa\spot@internal\xa[\spot@totaloptions]<#1>(#2){#4}%
}
```

\spot@internal Finally, we are ready to call the macros which actually do the work, depending on whether we are running Beamer or some other class. We return the parameters to their normal order.

```
\def\spot@internal[#1]<#2>(#3)#4{%
  \ifspot@beamer%
    \spot@internal@beamer<#2>(#3)[#1]{#4}%
  \else%
    \spot@internal@static<#2>(#3)[#1]{#4}%
  \fi%
}
```

5.4 Beamer version

\spot@internal@beamer

The macro \spot@internal@beamer implements the \spot command when used in a Beamer presentation.

```
\def\spot@internal@beamer<#1>(#2)[#3]#4{%
```

The text to be highlighted is first saved in a box named something like \spot@box@vii, where the roman numeral represents the spot counter \c@spot@spots. In this way it will automatically have the correct font size and shape (italics, boldface, etc.). It is set in math mode if necessary. For the sake of readability of the code, \spot@currentbox is let equal to \spot@box@vii.

```
  \@ifundefined{spot@box@\romannumeral\c@spot@spots}%
          {\xa\newbox\csname spot@box@\romannumeral\c@spot@spots\endcsname}{}%
  \ifmmode
    \global\xa\setbox\csname spot@box@\romannumeral\c@spot@spots\endcsname=\hbox{$#4$}%
  \else
    \global\xa\setbox\csname spot@box@\romannumeral\c@spot@spots\endcsname=\hbox{#4}%
  \fi
  \xa\let\xa\spot@currentbox\csname spot@box@\romannumeral\c@spot@spots\endcsname%
```

Now we are ready to set the text. On the highlighted slides, we actually produce two Ti*k*Z nodes at this time. The first is merely a transparent copy of the text itself; the second is a transparent copy of the entire highlighted node shape, and receives the ⟨*node name*⟩ called by the user. We require the first node (with the use as bounding box option) so as to fit the text seamlessly in line with its surroundings. The second node, bearing ⟨*node name*⟩, must be drawn now, so that the user can refer to ⟨*node name*⟩ within the frame; the ⟨*node name*⟩ could not simply be attached to the visible spotlight, since that is drawn with \AfterEndFrame, *after* all the user's code has been processed. On unhighlighted slides, we simply draw an opaque copy of the text in a Ti*k*Z node.

```
  \tikz[remember picture, baseline]{
    \alt<#1>{\path[use as bounding box]
                  node[anchor=base, inner sep=0, outer sep=0, opacity=0]
                  (spot@\romannumeral\c@spot@spots) {\usebox\spot@currentbox};
             \path (spot@\romannumeral\c@spot@spots)
                  node[anchor=center, outer sep=0,
                       shape=ellipse, inner sep=0.5ex,
                       #3, opacity=0] (#2) {\usebox\spot@currentbox};}
            {\path node[anchor=base, inner sep=0, outer sep=0, opacity=1]
                      (#2) {\usebox\spot@currentbox};}
  }%
```

Finally, we save the information needed to paint the spotlight at the end of the slide. Sometimes calling \AtEndFrame adds a little extra height to the frame's contents, which could cause a slight movement of the frame between highlighted and unhighlighted slides. We therefore make a call to \AtEndFrame on unhighlighted slides as well, to ensure that the same extra height be added even when we're not painting spotlights.

```
  \alt<#1>{\xa\spot@savepaint\xa{\romannumeral\c@spot@spots}[#3]}%
          {\AtEndFrame{\rule{0pt}{0pt}}}%
```

Finally, we increment the counter `\c@spot@spots` and end the macro.

```
  \global\advance\c@spot@spots by 1\relax%
}
```

`\spot@savepaint` The following macro saves the actual spotlight-drawing command to be executed at the end of the slide. The first parameter is a roman numeral labeling the spotlight, while the second contains the node options.

```
\def\spot@savepaint#1[#2]{%
  \AtEndFrame{\spot@paintspot(spot@#1)[#2]{\usebox{\csname spot@box@#1\endcsname}}}%
}
```

`\spot@paintspot` The command `\spot@paintspot` actually draws the spotlight at the saved location. Note that the first parameter is not the user-specified ⟨*node name*⟩, but an automatically generated name such as `spot@vii`.

```
\def\spot@paintspot(#1)[#2]#3{%
  \begin{tikzpicture}[remember picture, overlay, baseline]
      \path (#1) node[anchor=center, outer sep=0,
                      shape=ellipse, inner sep=0.5ex, text opacity=1,
                      path fading=spot@fade, text=black, #2] {#3};
  \end{tikzpicture}%
}
```

5.5 non-Beamer version

To make the spotlight turn out correctly, it is vital to run `\spot@paintspot` *after* all surrounding text has been typeset. Whereas in Beamer the code can be attached to a hook in the `\end{frame}`, in other document classes we place the code in a `\marginpar`, which (being a TEX insertion) is typeset after the rest of the page. If `\spot` is called in certain TEX modes, however, `\marginpar` would produce an error. In this case, `\spot` saves up the commands to be placed into a `\marginpar` later.

`\spot@savedpaintcommands` `\dospots` Whenever `\spot` is called, the `\spot@paintspot` commands will be saved up in the macro `\spot@savedpaintcommands`. A subsequent call to `\dospots`, by the user if necessary, executes those commands and resets the macro to empty.

```
\gdef\spot@savedpaintcommands{}
\def\dospots{%
  \marginpar{\spot@savedpaintcommands}%
  \gdef\spot@savedpaintcommands{}%
}
```

`\dospotsheader` `\dospotsfooter` Since headers and footers are also TEX insertions, the `\spot@savedpaintcommands` could also be run there. The following commands are synonyms.

```
\def\dospotsheader{%
  \spot@savedpaintcommands%
  \gdef\spot@savedpaintcommands{}%
}
\let\dospotsfooter=\dospotsheader
```

`\spot@internal@static` Much of the code in `\spot@internal@static` is identical with that of `\spot@internal`. The chief difference is that the overlay specification is completely ignored.

```
\def\spot@internal@static<#1>(#2)[#3]#4{%
  \@ifundefined{spot@box@\romannumeral\c@spot@spots}%
          {\xa\newbox\csname spot@box@\romannumeral\c@spot@spots\endcsname}{}%
  \ifmmode
    \global\xa\setbox\csname spot@box@\romannumeral\c@spot@spots\endcsname=\hbox{$#4$}%
  \else
    \global\xa\setbox\csname spot@box@\romannumeral\c@spot@spots\endcsname=\hbox{#4}%
  \fi
  \xa\let\xa\spot@currentbox\csname spot@box@\romannumeral\c@spot@spots\endcsname%
  \tikz[remember picture, baseline]{
     \path[use as bounding box]
            node[anchor=base, inner sep=0, outer sep=0, opacity=0]
            (spot@\romannumeral\c@spot@spots) {\usebox\spot@currentbox};
     \path (spot@\romannumeral\c@spot@spots)
             node[anchor=center, inner sep=0, outer sep=0,
                  shape=ellipse, inner sep=0.5ex,
                  #3, opacity=0] (#2) {\usebox\spot@currentbox};
    }%
  \xa\spot@static@savepaint\xa{\romannumeral\c@spot@spots}[#3]%
  \global\advance\c@spot@spots by 1\relax%
  \global\spot@starfalse%
}
```

`\spot@static@savepaint` When `\spot@static@savepaint` is called, it first checks whether TEX is in an inner mode and/or math mode, and whether the starred version `spot*` was used. If any of these three conditions holds, a call to `\marginpar` would likely fail, so the `\marginpar` command is saved to be executed by a `\dospots` command sometime later, and `\spot@checkforlostspots` will be run after the current page is processed. Otherwise, the appropriate `\spot@paintspot` command is immediately placed in a `\marginpar`, and any saved `\marginpar` commands are run at this time with `\dospots`. As with `\spot@savepaint`, the first parameter is the roman numeral identifying which spotlight is being painted, and the second contains the node options.

```
\def\spot@static@savepaint#1[#2]{%
  \def\spot@saveit{%
    \g@addto@macro\spot@savedpaintcommands{%
      \spot@paintspot(spot@#1)[#2]{\usebox{\csname spot@box@#1\endcsname}}}%
    \afterpage{\spot@checkforlostspots}%
  }%
  \ifspot@star
    \spot@saveit
  \else
    \ifinner
      \spot@saveit
    \else
      \ifmmode
        \spot@saveit
```

```
      \else
        \marginpar{\spot@paintspot(spot@#1)[#2]{\usebox{\csname spot@box@#1\endcsname}}}%
        \dospots
      \fi
    \fi
  \fi
}
```

`\spot@checkforlostspots` Between pages, the command `\spot@checkforlostspots` checks whether any `\spot@paintspot` commands were saved into `\spot@savedpaintcommands` but never executed by a `\dospots`. If so, it issues an error message to the user.

```
\def\spot@checkforlostspots{%
  \def\spot@empty{}%
  \ifx\spot@savedpaintcommands\spot@empty%
    \relax%
  \else%
    \bgroup
    \advance\count0 by -1
    \PackageError{spot}%
       {A \protect\dospots\space command is missing
        from page \the\count0.\MessageBreak
        Some highlighted text will not appear in the output}%
       {If the last \protect\spot\space command on a
        page is issued in math mode or a \MessageBreak
        TeX inner mode (such as a tabular environment),
        it must be followed by a \MessageBreak
        \protect\dospots\space command somewhere later
        on the page, outside such a mode.\MessageBreak
        You could also put \protect\dospotsheader\space
        in a header or footer on each page.}
    \egroup
  \fi%
}
```