OpTEX slides

Petr Olšák

petr@olsak.net

http://petr.olsak.net/optex

Basics

- The the simple document looks like:

```
\slides       % style initialized
%\wideformat  % 16:9
\slideshow    % partially uncovering ideas

\sec First slide
text
\pg;

\sec Second slide
text
\pg.
```

- If `\slideshow` is missing or commented out then “partially uncovering ideas” (see later) are deactivated. It is useful for printing.
- The `\slideshow` must be the last command in the declaration part of the document.
- By default, the slides have A5 landscape format. You can declare `\wideformat`. Then the height is the same but width is 263 mm, i. e. the ratio width:height is 16:9.

Title slide

- Title of the document (used at the first slide) is created by `\tit Title` (terminated by end of line).
- The `\subtit Author name etc.` (terminated by end of line) can be used after `\tit` at the first slide.
- You can use `\nl` for a new line in paragraphs or titles.

Default design

- The paragraph texts are ragged right.
- Titles, subtitles, and section titles are centered.
- The `\sec` and `\secc` are printed without numbers.
- No paragraph indentation, a little vertical space between paragraphs.
- The Heros font family (aka Helvetica) is initialized as default.
  Sans-serif FiraMath font for math typesetting is used.
  If `\fontfam[`⟨*famname*⟩`]` precedes `\slides` then it rewrites default.
- The items in lists are started by a blue square (`\type X` and `\type x`).

One slide (one page)

- Top-level item list is activated by default. The asterisk * opens new item at the top-level list.
- Nested items lists (second and more level) should be created in the `\begitems...\enditems` environments.
- Each slide (page) must be terminated by `\pg;` command.
- The last slide must be terminated by `\pg.` command or by `\bye`.

```
\sec My ideas

* First idea
* Second idea
  \begitems
  * First sub-idea
  * Second sub-idea
  \enditems
* Final idea
```

- Note: each page is processed in a group, so: put your own definitions (if exist) before `\slideshow` or use `\global` assignment.

Partially uncovering ideas

- The control sequence `\pg` must be followed by:

Partially uncovering ideas

- The control sequence `\pg` must be followed by:
  - the character `;` – normal next page,

Partially uncovering ideas

- The control sequence `\pg` must be followed by:
  - the character `;` – normal next page,
  - the character `.` – the end of the document,

Partially uncovering ideas

- The control sequence `\pg` must be followed by:
  - the character `;` – normal next page,
  - the character `.` – the end of the document,
  - the character + – next page keeps the same text and a next text is added (usable for partially uncovering of ideas).

Partially uncovering ideas

- The control sequence `\pg` must be followed by:
  - the character `;` – normal next page,
  - the character `.` – the end of the document,
  - the character + – next page keeps the same text and a next text is added (usable for partially uncovering of ideas).
- Summary:

```
\pg;     ... next page
\pg.     ... the end of the document
\pg+     ... uncover next text on the same page
```

Partially uncovering ideas

- The control sequence `\pg` must be followed by:
  - the character `;` – normal next page,
  - the character `.` – the end of the document,
  - the character + – next page keeps the same text and a next text is added (usable for partially uncovering of ideas).
- Summary:

```
\pg;     ... next page
\pg.     ... the end of the document
\pg+     ... uncover next text on the same page
```

- When `\slideshow` is not declared then `\pg+` is deactivated.

Partially uncovering ideas

- The control sequence \pg must be followed by:
  - the character ; – normal next page,
  - the character . – the end of the document,
  - the character + – next page keeps the same text and a next text is added (usable for partially uncovering of ideas).
- Summary:

```
\pg;     ... next page
\pg.     ... the end of the document
\pg+     ... uncover next text on the same page
```

- When \slideshow is not declared then \pg+ is deactivated.
- The \pg+ creates a new “virtual page”, so the current paragraph is terminated.

Example with partially uncovering ideas

The previous page was created by:

```
\sec Partially uncovering ideas

* The control sequence `\pg` must be followed by:\pg+

\begitems
* the character `.` -- normal next page,\pg+
* the character `;` -- the end of the document,\pg+
* the character `+` -- next ... \pg+
\enditems

* Summary
...
* When `\slideshow` is not declared
  then `\pg+` is deactivated.\pg+
* The `\pg+` creates a new \"virtual page",
  so the current paragraph is terminated.
\pg;
```

Notes to `\slideshow`

- When `\slideshow` is active then references created by `\ref` point to the first uncovering “virtual” page where the destination is and references created by `\pgref` point to the last “virtual” page.
- If the text overfulls the page (slide) then it follows to the next page without saying explicitly `\pg;`. But `\slideshow` cannot work in this case.

Notes to \slideshow

- When \slideshow is active then references created by \ref point to the first uncovering “virtual” page where the destination is and references created by \pgref point to the last “virtual” page.
- If the text overfulls the page (slide) then it follows to the next page without saying explicitly \pg;. But \slideshow cannot work in this case.
- If \slideshow then each part of page between two \pg’s or between \slideshow and the first \pg is processed in a local group.
- If not \slideshow then the document is not separated to groups. This can create different results. So, you can put \slideopen command instead of \slideshow. Then local groups are opened exactly as when \slideshow is used but \slideshow is not activated. Example:

```
\slides
\def\foo...{...} % global definitions.
%\slideshow
\slideopen       % opens group for first page.
... first page
\pg;             % closes group and opens group for second page.
... second page
\pg.             % closes group of the last page.
```

More about design

- You can use `\backgroundpic{`⟨*image-file*⟩`}` for putting an image to the background.
- You can re-declare `\footline` or re-define internal macros for design as you wish.
- The TeXGyre Heros font is used as default text font, the FiraMath is used for math.
- If you want to use another text font fmaily, use `\fontfam` *before* `\slides` command.
- If you want to use different math font, use `\loadmath{[font]}` before `\fontfam` (if used) and before `\slides`. For example:

```
\loadmath{[Asana-Math]} % Math font: Asana
\fontfam[Termes]        % Text font: Termes
\slides
...
```

- You can put the images or text wherever using `\putimege` or `\puttext` macros...

HELLO

Putting images and texts wherever

- `\puttext` ⟨*right*⟩ ⟨*up*⟩ {⟨*text*⟩} puts a ⟨*text*⟩ to the desired place: It moves the current point ⟨*right*⟩ and ⟨*up*⟩, puts the ⟨*text*⟩ and returns back, so the typesetting continues from previous position. The parameters ⟨*right*⟩ and ⟨*up*⟩ are dimensions. For example

```
\puttext 0mm 50mm {\Red HELLO}
```

  prints red HELLO, as shown here.

Putting images and texts wherever

- `\puttext` ⟨*right*⟩ ⟨*up*⟩ {⟨*text*⟩} puts a ⟨*text*⟩ to the desired place: It moves the current point ⟨*right*⟩ and ⟨*up*⟩, puts the ⟨*text*⟩ and returns back, so the typesetting continues from previous position. The parameters ⟨*right*⟩ and ⟨*up*⟩ are dimensions. For example

```
\puttext 0mm 50mm {\Red HELLO}
```

  prints red HELLO, as shown here.

- `\putpic` ⟨*right*⟩ ⟨*up*⟩ ⟨*width*⟩ ⟨*height*⟩ {⟨*image-file*⟩} puts the image with desired ⟨*width*⟩ and ⟨*height*⟩ at the position like `\puttext` puts the text.

Putting images and texts wherever

- `\puttext` ⟨*right*⟩ ⟨*up*⟩ {⟨*text*⟩} puts a ⟨*text*⟩ to the desired place: It moves the current point ⟨*right*⟩ and ⟨*up*⟩, puts the ⟨*text*⟩ and returns back, so the typesetting continues from previous position. The parameters ⟨*right*⟩ and ⟨*up*⟩ are dimensions. For example

  ```
  \puttext 0mm 50mm {\Red HELLO}
  ```

  prints red HELLO, as shown here.

- `\putpic` ⟨*right*⟩ ⟨*up*⟩ ⟨*width*⟩ ⟨*height*⟩ {⟨*image-file*⟩} puts the image with desired ⟨*width*⟩ and ⟨*height*⟩ at the position like `\puttext` puts the text.
- The ring above is the result of

  ```
  \putpic .8\hsize 20mm 30mm \nospec {op-ring.png}
  ```

  used at beginning of this paragraph.

Putting images and texts wherever

- `\puttext` ⟨*right*⟩ ⟨*up*⟩ {⟨*text*⟩} puts a ⟨*text*⟩ to the desired place: It moves the current point ⟨*right*⟩ and ⟨*up*⟩, puts the ⟨*text*⟩ and returns back, so the typesetting continues from previous position. The parameters ⟨*right*⟩ and ⟨*up*⟩ are dimensions. For example

```
\puttext 0mm 50mm {\Red HELLO}
```

prints red HELLO, as shown here.

- `\putpic` ⟨*right*⟩ ⟨*up*⟩ ⟨*width*⟩ ⟨*height*⟩ {⟨*image-file*⟩} puts the image with desired ⟨*width*⟩ and ⟨*height*⟩ at the position like `\puttext` puts the text.
- The ring above is the result of

```
\putpic .8\hsize 20mm 30mm \nospec {op-ring.png}
```

used at beginning of this paragraph.

- Use `\nospec` for ⟨*width*⟩ or ⟨*height*⟩ of the image if you don't want to specify both dimensions (because you don't want to change the image aspect ratio).

Limits of the \pg+ sequence

- The \pg+ sequence (partially uncovering ideas) cannot be used inside a group.
- The exception is the nested environment \begitems...\enditems.
- The \pg+ always finalizes the current paragraph. It is impossible to hide only a part of the horizontal mode.

Limits of the \pg+ sequence

- The \pg+ sequence (partially uncovering ideas) cannot be used inside a group.
- The exception is the nested environment \begitems...\enditems.
- The \pg+ always finalizes the current paragraph. It is impossible to hide only a part of the horizontal mode.

The \layers...\endlayers environment

If you really need something unsupported by \pg+ then you can use

```
\layers ⟨number⟩
⟨layered text⟩
\endlayers
```

- The \layers opens ⟨*number*⟩ following pages with the same surrounding text. The counter \layernum is incemented from one to ⟨*number*⟩ . The ⟨*layered text*⟩ should use \layernum including conditions like \ifnum\layernum or \ifcase\layernum. See next page...

Example of `\layers` environment

The `\slides` style provides a shortcut `\use` and a macro `\pshow` (means partially show):

```
\def\use#1#2{\ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\use{=#1}\Red \use{<#1}\Transparent \ignorespaces}
```

`\use{=`⟨*num*⟩`}{`⟨*something*⟩`}` does ⟨*something*⟩ only if `\layernum=`⟨*num*⟩.

The `{\pshow`⟨*num*⟩ ⟨*text*⟩`}` prints ⟨*text*⟩ in Red when current layer is equal to ⟨*num*⟩ or it prints ⟨*text*⟩ normally when the current layer is greater than ⟨*num*⟩. The transparent (invisible) text is used in other cases.

The following dance:

First text.

Example of `\layers` environment

The `\slides` style provides a shortcut `\use` and a macro `\pshow` (means partially show):

```
\def\use#1#2{\ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\use{=#1}\Red \use{<#1}\Transparent \ignorespaces}
```

`\use{=`⟨*num*⟩`}{`⟨*something*⟩`}` does ⟨*something*⟩ only if `\layernum=`⟨*num*⟩.

The `{\pshow`⟨*num*⟩ ⟨*text*⟩`}` prints ⟨*text*⟩ in Red when current layer is equal to ⟨*num*⟩ or it prints ⟨*text*⟩ normally when the current layer is greater than ⟨*num*⟩. The transparent (invisible) text is used in other cases.

The following dance:

Second text. First text.

Example of `\layers` environment

The `\slides` style provides a shortcut `\use` and a macro `\pshow` (means partially show):

```
\def\use#1#2{\ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\use{=#1}\Red \use{<#1}\Transparent \ignorespaces}
```

`\use{=`⟨*num*⟩`}{`⟨*something*⟩`}` does ⟨*something*⟩ only if `\layernum=`⟨*num*⟩.

The `{\pshow`⟨*num*⟩ ⟨*text*⟩`}` prints ⟨*text*⟩ in Red when current layer is equal to ⟨*num*⟩ or it prints ⟨*text*⟩ normally when the current layer is greater than ⟨*num*⟩. The transparent (invisible) text is used in other cases.

The following dance:

Second text. Third text. First text.

Example of `\layers` environment

The `\slides` style provides a shortcut `\use` and a macro `\pshow` (means partially show):

```
\def\use#1#2{\ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\use{=#1}\Red \use{<#1}\Transparent \ignorespaces}
```

`\use{=`⟨*num*⟩`}{`⟨*something*⟩`}` does ⟨*something*⟩ only if `\layernum=`⟨*num*⟩.

The `{\pshow`⟨*num*⟩ ⟨*text*⟩`}` prints ⟨*text*⟩ in Red when current layer is equal to ⟨*num*⟩ or it prints ⟨*text*⟩ normally when the current layer is greater than ⟨*num*⟩. The transparent (invisible) text is used in other cases.

The following dance:

Second text. Third text. First text.

was generated by

```
\layers 3
{\pshow2 Second text.} {\pshow3 Third text.} {\pshow1 First text.}
\endlayers
```

Example of `\layers` environment

The `\slides` style provides a shortcut `\use` and a macro `\pshow` (means partially show):

```
\def\use#1#2{\ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\use{=#1}\Red \use{<#1}\Transparent \ignorespaces}
```

`\use{=`⟨*num*⟩`}{`⟨*something*⟩`}` does ⟨*something*⟩ only if `\layernum=`⟨*num*⟩.

The `{\pshow`⟨*num*⟩ ⟨*text*⟩`}` prints ⟨*text*⟩ in Red when current layer is equal to ⟨*num*⟩ or it prints ⟨*text*⟩ normally when the current layer is greater than ⟨*num*⟩. The transparent (invisible) text is used in other cases.

The following dance:

Second text. Third text. First text.
was generated by

```
\layers 3
{\pshow2 Second text.} {\pshow3 Third text.} {\pshow1 First text.}
\endlayers
```

- The ⟨*layered text*⟩ is treated as a macro parameter. So, you cannot use verbatim nor `\sec` titles here. Maximal one `\layers` environment can be per one page (terminated by `\pg+` or `\pg;` or `\pg.`).

Comparison OpTEX slides with Beamer[1]

The LaTeX package **Beamer** gives much more features and many themes are prepared for Beamer, **but**

[1] http://www.ctan.org/pkg/beamer

Comparison OpTEX slides with Beamer[1]

The LaTeX package **Beamer** gives much more features and many themes are prepared for Beamer, **but**

- the user of Beamer is forced to *program* his/her document using dozens of `\begin{foo}` and `\end{foo}` and many other programming constructions,

[1] http://www.ctan.org/pkg/beamer

Comparison OpTEX slides with Beamer[1]

The LaTeX package **Beamer** gives much more features and many themes are prepared for Beamer, **but**

- the user of Beamer is forced to *program* his/her document using dozens of `\begin{foo}` and `\end{foo}` and many other programming constructions,
- plain TeX gives you a possibility to simply *write* your document with minimal markup. The result is more compact. You can concentrate on the contents of your document, not on the programming syntax.

[1] http://www.ctan.org/pkg/beamer

Comparison OpTEX slides with Beamer[1]

The LaTeX package **Beamer** gives much more features and many themes are prepared for Beamer, **but**

- the user of Beamer is forced to *program* his/her document using dozens of `\begin{foo}` and `\end{foo}` and many other programming constructions,
- plain TeX gives you a possibility to simply *write* your document with minimal markup. The result is more compact. You can concentrate on the contents of your document, not on the programming syntax.
- User needs to read 250 pages of doc for understanding Beamer,

[1] http://www.ctan.org/pkg/beamer

Comparison OpTEX slides with Beamer[1]

The LaTeX package **Beamer** gives much more features and many themes are prepared for Beamer, **but**

- the user of Beamer is forced to *program* his/her document using dozens of `\begin{foo}` and `\end{foo}` and many other programming constructions,
- plain TeX gives you a possibility to simply *write* your document with minimal markup. The result is more compact. You can concentrate on the contents of your document, not on the programming syntax.
- User needs to read 250 pages of doc for understanding Beamer,
- on the other hand, you need to read only eleven slides[2] and you are ready to use **OpTEX slides**.

---

1 http://www.ctan.org/pkg/beamer

2 this twelfth slide isn't counted

Thanks for your attention

Thanks for your attention

Questions?