Loading PICTEX without problems

a useful hack for LATEX users

Hans Hagen

April 1 1997

TEX provides 256 ⟨*dimensions*⟩ and 256 ⟨*skips*⟩. In ConTEXt this is no problem, but in packages that have many authors, one can be quite sure that a lot of ⟨*dimensions*⟩ are allocated. Packages that use PICTEX can therefore run out of ⟨*dimensions*⟩ quite fast. This module was written as a reaction to persistent problems with loading PPCHTEX in LATEX and PICTEX deserves a solution. I therefore dedicate this module to Tobias Burnus and Dirk Kuypers, who use PPCHTEX in a LATEX environment and suggested a lot of extensions to the repertoire of PPCHTEX commands.

This module presents a solution that is quite effective: all ⟨*dimensions*⟩ are drawn from the pool of ⟨*dimensions*⟩ and ⟨*skips*⟩, depending on the availability. This is possible because ⟨*dimensions*⟩ are ⟨*skips*⟩ without a glue component. Therefore we can use ⟨*skips*⟩ as ⟨*dimensions*⟩. However, some incompatibility can result from assignments that look like:

```
\somedimen=\someskip
```

In such cases the ⟨*dimension*⟩ equals the fixed part of the ⟨*skip*⟩ or in other words: this assignment strips off the glue. Because PICTEX uses no glue components, I thought I could interchange both register types without problems, but alas, this didn't hold for all ⟨*dimensions*⟩.

In Plain TEX the allocation macros are defined with (as) `\outer`. This means that they cannot appear inside macros, not even in an indirect way. We therefore have to redefine both `\newdimen` and `\newskip` to non–`\outer` alternatives. In most macro packages this redefinition already took place. We save the original meanings, so we can restores them afterwards.

```
1  \let\normalnewdimen = \newdimen
   \let\normalnewskip  = \newskip
```

```
2  \catcode`@=11 % I'd rather used \unprotect
   \def\temporarynewdimen {\alloc@1\dimen\dimendef\insc@unt}
   \def\temporarynewskip  {\alloc@2\skip \skipdef \insc@unt}
   \catcode`@=12 % and \protect.
```

Here comes the trick. Depending on how many ⟨*dimensions*⟩ and ⟨*skips*⟩ are allocated, the `\newdimen` assigns a ⟨*dimensions*⟩ or ⟨*skip*⟩. Plain TEX allocates 15 ⟨*dimensions*⟩ and 17 ⟨*skips*⟩. After loading PICTEX, 71 ⟨*dimensions*⟩ and and 71 ⟨*skips*⟩ are allocated. Indeed, PICTEX needs 110 ⟨*dimensions*⟩!

```
\def\newdimen%
  {\ifnum\count11>\count12
     \let\next=\temporarynewskip
   \else
     \let\next=\temporarynewdimen
   \fi
   \next}
```

When I was testing a new version of PPCHTEX in Plain TEX I had to find out that this exchange of registers sometimes leads to unwanted results. It took me some hours to find out that the source of errors originated in constructions like:

```
\ifdim\DimenOne<\DimenTwo whatever you want \else or not \fi
```

When `\DimenOne` is a ⟨*skip*⟩ and `\DimenTwo` is a ⟨*dimension*⟩, TEX scans for some optional glue component, like in:

```
\skip0=\dimen0 plus 10pt minus 5pt
```

The most robust solution to this problem is:

```
\ifdim\DimenOne<\DimenTwo\relax right \else wrong \fi
```

Some close reading of the PICTEX source however learned me that this problem could be solved best by just honoring the allocation of ⟨*dimensions*⟩ when the name of the macro explictly stated the character sequence `dimen`. A next implementation therefore automatically declared all ⟨*dimensions*⟩ with this sequence in their names with `\dimen`. Again I was too optimistic, so now we do it this way (the comments are from PICTEX, which like TABLE, is an example of a well documented package):

```
3 \catcode`!=11
\temporarynewdimen\!dimenA      %.AW.X.DVEUL..OYQRST
\temporarynewdimen\!dimenB      %....X.DVEU...O.QRS.
\temporarynewdimen\!dimenC      %..W.X.DVEU......RS.
\temporarynewdimen\!dimenD      %..W.X.DVEU....Y.RS.
\temporarynewdimen\!dimenE      %..W........G..YQ.S.
\temporarynewdimen\!dimenF      %...........G..YQ.S.
\temporarynewdimen\!dimenG      %...........G..YQ.S.
\temporarynewdimen\!dimenH      %...........G..Y..S.
\temporarynewdimen\!dimenI      %...BX.........Y....
\temporarynewdimen\!dxpos       %..W......U..P....S.
\temporarynewdimen\!dypos       %..WB.....U..P......
\temporarynewdimen\!xloc        %..WB.....U.......S.
\temporarynewdimen\!xpos        %..........L.P..Q.ST
\temporarynewdimen\!yloc        %..WB.....U.......S.
\temporarynewdimen\!ypos        %..........L.P..Q.ST
\temporarynewdimen\!zpt         %.AWBX.DVEULGP.YQ.ST
```

Tobias tested this module in all kind of LATEX dialects so we were able to find out that we also needed to declare:

```
4 \temporarynewdimen\linethickness
\catcode`!=12
```

After all, the new definition of `\newdimen` became:

```
5 \def\newdimen#1%
  {\ifx#1\undefined
     \ifnum\count11>\count12
       \temporarynewskip#1\relax
     \else
       \temporarynewdimen#1\relax
     \fi
   \fi}
```

Curious readers can still find the previous solution in the source. The next macro is used instead of `\input`. This macro also reports some statistics.

```
6 \def\dimeninput#1 %
  {\message{[before: d=\the\count11,s=\the\count12]}%
   \input #1 \relax
   \message{[after: d=\the\count11,s=\the\count12]}}%
```

Not every package defines `\fiverm`, PICTEX's pixel, so let's take care of that omision now:

```
7 \ifx\undefined\fiverm
  \font\fiverm=cmr5
\fi
```

The actual loading of PICTEX depends on the package. For LATEX users we take care of loading the auxiliary ones too.

```
8 \ifx\beginpicture\undefined
  \ifx\newenvironment\undefined
    \dimeninput pictex   \relax
  \else
    \dimeninput prepicte \relax
    \dimeninput pictex   \relax
    \dimeninput postpict \relax
  \fi
\fi
```

Finally we restore the old definitions of `\newdimen` and `\newskip`:

```
9   \let\newdimen = \normalnewdimen
    \let\newskip  = \normalnewskip
```

and just hope for the best.

```
10  \endinput
```