pdfsplit

(as \vsplit, but for pdfs)

1. Starting point:

\clip[...]{\externalfigure[..][..]}

\setupclipping[...,...=...,...]

nx	number
ny	number
x	number
y	number
width	dimension
height	dimension
hoffset	dimension
voffset	dimension

mp name

watch for mp: a metapost path

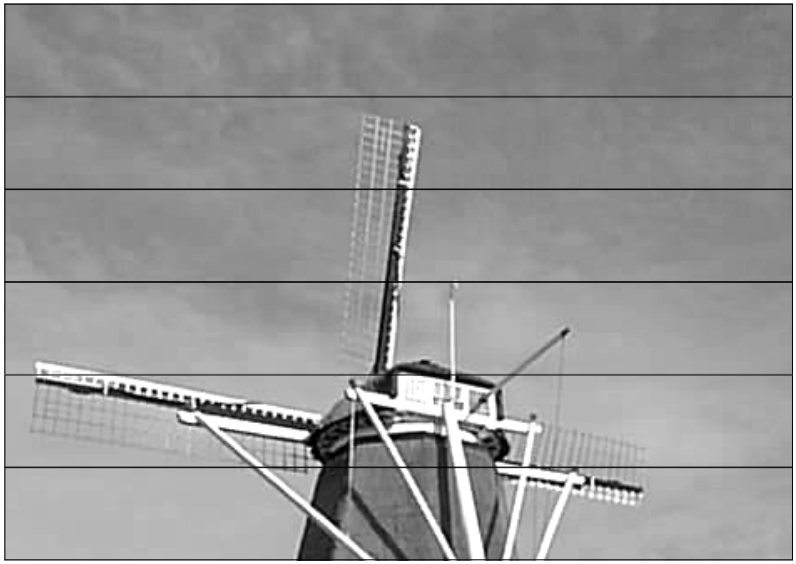
2. Next: a bit of Lua code (which never hurt) to generalize

```
while vh <= H do
  S =string.format("
  {\\ruledvbox{\\clip[voffset=\%dsp,
                     width=\%dsp,
                     height=\%dsp]
                   {\\externalfigure[\%s][\%s]}}}
                    \\par\\nointerlineskip\\blank[1sp]",
                    vh,W,step,
                    Fig,FigOpt)
  tex.sprint(tex.ctxcatcodes,S)
  vh = vh + step
end
```

"H/step" \vboxs of width "W" and height "step".

Ok, let's pack it into a function and make some tests:

```
\ctxlua{document.lscarso.LuaSliceIt("mill.png",
"\the\Hfig","\the\Wfig",
"\the\dimexpr 1.0\lineheight\relax",<- this is the step
"width=\\textwidth,height=1.0\\textheight,factor=fit")}
```



Ok, maybe it works for images... but what about a pdf with texts ?

It’s also possible to slice a pdf, but just one example shows that it is doesn’t work:

$$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345$$ 001

$$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{14639190792407439662689546747109297683610 83}{28167031592803840774471658809866170636947 2} = 5.19727850773863016179521668 6$$ 002

$$\sum_{x=0}^{150} \frac{1}{x+1} = \frac{41955696676761358111539698151370732349445617467329193399143372019 27}{74950290119682722826682048179211899329291912740854280826732942400 0} = 5.59780310520017018796571597 3$$ 003

$$4\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{18472920064106597929865025} = 3.17384233719074940869022414 0$$ 004

$$4\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.14491490355885179920458621 2$$ 005

$$4\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.14192587583979015127120007 5$$ 006

$$4\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.14162598581204323815399369 2$$ 007

$$4\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.14159598691201548846261251 9$$ 008

$\sum_{x=0} 2x+1$

$4 \sum_{x=0}^{3000000} \frac{-1^x}{2x+1} = 3.14159298692301546071264338 0$

009

010

011

smart move...and now ?

3. We can improve things by using a small `step` and grouping the lines together.

Let's try for example with 4mm:

$$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345$$

$$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{14639190792407439662689546747109297683610 83}{28167031592803840774471658809866170636947 2} = 5.197278507738630161795216686$$

$$\sum_{x=0}^{150} \frac{1}{x+1} = \frac{41955696676761358111539698151370732349445617467329193399143372019 27}{74950290119682722826682048179211899329291912740854280826732942400 0} = 5.597803105200170187965715973$$

$$4\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{18472920064106597929865025} = 3.173842337190749408690224140$$

$$4\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914903558851799204586212$$

$$4\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141925875839790151271200075$$

$$4\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$$

$$4\sum_{x=0}^{300000}\frac{-1^x}{2x+1} = 3.14159598691201548846261251 9$$

$$4\sum_{x=0}^{3000000}\frac{-1^x}{2x+1} = 3.14159298692301546071264338 0$$

A set of good breakpoints are given by slices `4,7,10,13,16` and we can hence create the right partitions:

$$\sum_{x=0}^{10}\frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345 004$$

$$\sum_{x=0}^{100}\frac{1}{x+1} = \frac{1463919079240743966268954674710929768361083}{2816703159280384077447165880986617063694 72} = 5.19727850773863016179521668 6 007$$

$$\sum_{x=0}^{150}\frac{1}{x+1} = \frac{419556966767613581115396981513707323494456174673291933991433720192 7}{749502901196827228266820481792118993292919127408542808267329424000} = 5.59780310520017018796571597 3 010$$

$$4\sum_{x=0}^{30}\frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{18472920064106597929865025} = 3.17384233719074940869022414 0 013$$

$$4\sum_{x=0}^{300}\frac{-1^x}{2x+1} = 3.144914903558851799204586212$$ 016

$$4\sum_{x=0}^{3000}\frac{-1^x}{2x+1} = 3.141925875839790151271200075$$

$$4\sum_{x=0}^{30000}\frac{-1^x}{2x+1} = 3.141625985812043238153993692$$

$$4\sum_{x=0}^{300000}\frac{-1^x}{2x+1} = 3.141595986912015488462612519$$

$$4\sum_{x=0}^{3000000}\frac{-1^x}{2x+1} = 3.141592986923015460712643380$$ 031

The last partition is a big one (laziness....)

We collect all good breakpoint in a ``global'' Lua table

```
\startluacode
 document.lscarso.GoodPoints = {4,7,10,13,16}
\stopluacode
```

and modify a bit the previous function to collect boxes:

```
\ctxlua{document.lscarso.LuaSliceItAndCollect(
          "pari_example.pdf",
          "\the\Hfig","\the\Wfig",
          "\the\dimexpr 4mm\relax",
          "width=\\textwidth","")}
```

4. OK, now the next step: avoid manual calculations as much as possible.

A solution is to convert the pdf into a black and white bitmap, find all the white rows (a white row is a row where all the pixels are white) and use them to create partitions

(uh, easy to say....)

It’s easy to convert a pdf in bitmap, because there are several programs available for this; for example `pdftoppm`.

```
pdftoppm -mono -r 72 -f 1 -l 1
      pari_example.pdf pari_example}
```

converts the pdf in a 72dpi pbm file `pari_example-000001.pbm`.

Aditya’s filter module is very useful here.

The next step is a bit more complicated.

a. We don’t use a program with many command line switches: we use a `C` library. The library is `Leptonica`, because is ``well written’’;

b. we don’t build a custom program with the library: we build a `Lua` binding. We gain an enormous flexibility, but we loose something on speed;

c. we don’t manually build the binding: we use `SWIG`, because it is a good match with `Lua`

This is the script `leptonica.i` used by `SWIG` to build the `C` code for the binding

```
%module leptonica
%{
#include "allheaders.h"
%}
%ignore setPixMemoryManager;
%include "leptonica/environ.h"
%include "leptonica/leptprotos.h"
%include "leptonica/imageio.h"
%include "leptonica/morph.h"
/* My util function  */
/*int test(char *argv); */
l_uint32 uti_valref_l_uint32(l_uint32  *v) ;
l_uint32 *uti_getref_l_uint32() ;
l_uint32 uti_pixGetPixel(PIX       *pix,
                         l_int32    x,
                         l_int32    y);
 end
```

And this is the shell script to build the binding:

```
/opt/swig-1.3.40/bin/swig  -lua leptonica.i

gcc -I./leptonica -I/opt/swig-1.3.40/include
  -c leptonica_wrap.c -o leptonica_wrap.o

gcc -c -I./leptonica -L.  lept-uti.c -o lept-uti.o

gcc -Wall -shared \
  -I./leptonica -I/opt/swig-1.3.40/include \
  -L./ -L/opt/swig-1.3.40/lib \
  leptonica_wrap.o lept-uti.o  liblept.a libz.a\
  -ltiff  -lgif \
  libpng12.a -o leptonica.so
```

<- this is the shared library used in lua

The shared library is like a `Lua` module: it has its functions and tables. We can then build the function `lept_get_breaks` that, given a pbm `filename` at `res` dpi fills the `GoodPoint` tavble with y positions of the white rows:

```
require("leptonica")
function document.lscarso.lept_get_breaks(filename,res)
    ...
    return GoodPoint
end
```

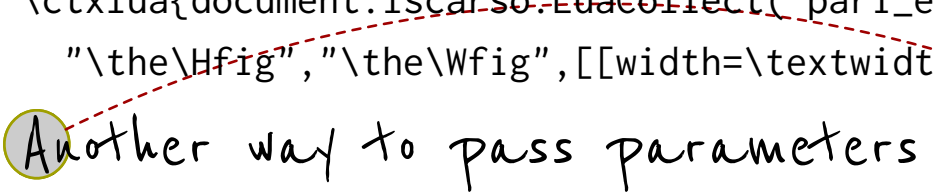
The heart of the function is a loop over the rows of the pbm:

```
local GoodPoint = {}  -- document.lscarso.GoodPoints or {}
if pixs then
 GoodPoint[1] = 0 <-Lua arrays start from 1....
 for y=0,h-1 do
    out = '', storey = true
    for x=0,w-1 do
       bit =  lua_pixGetPixel(pixs,x,y)
       if bit == 1 then storey = false; break end
    end
    if storey and (math.floor(0.5+ (y/res) *72.27*2^16)
            ~= GoodPoint[#GoodPoint]) then
         GoodPoint[#GoodPoint +1] = math.floor(0.5+ (y/res) *72.27*2^16
    end
    storey = true
 end
 GoodPoint[#GoodPoint +1] = math.floor(0.5+ (h/res) *72.27*2^16)
 else
  tex.sprint("ERROR")
 end
 return GoodPoint
```

hey, these are scaled points...

Then the LuaCollect function create the partitions.

```
\executesystemcommand{pdftoppm -mono -r 72 -f 1 -l 1 pari_example.pdf
pari_example}
\ctxlua{document.lscarso.GoodPoints =
   document.lscarso.lept_get_breaks("pari_example-000001.pbm",72)}
\ctxlua{document.lscarso.LuaCollect("pari_example.pdf",
  "\the\Hfig","\the\Wfig",[[width=\textwidth]])}
```



Ok, fire.

Just vskip a bit

$$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345$$

$$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{14639190792407439662689546747109297683610 83}{28167031592803840774471658809866170636947 2} = 5.197278507738630161795216686$$

$$\sum_{x=0}^{150} \frac{1}{x+1} = \frac{41955696676761358111539698151370732349445617467329193399143372019 27}{74950290119682722826682048179211899329291912740854280826732942400 0} = 5.597803105200170187965715973$$

$$4\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{18472920064106597929865025} = 3.173842337190749408690224140$$

$$4\sum_{x=0}^{300}\frac{-1^x}{2x+1} = 3.144914903558851799204586212$$

$$4\sum_{x=0}^{3000}\frac{-1^x}{2x+1} = 3.141925875839790151271200075$$

$$4\sum_{x=0}^{30000}\frac{-1^x}{2x+1} = 3.141625985812043238153993692$$

$$4\sum_{x=0}^{300000}\frac{-1^x}{2x+1} = 3.141595986912015488462612519$$

$$4\sum_{x=0}^{3000000}\frac{-1^x}{2x+1} = 3.141592986923015460712643380$$

A good breakpoint is marked with an horizontal rule: a black stripe means that good points are very tight there.

OK, but still some problems....

The algorithm that finds a good breakpoint is rather simple: for example the are some unwanted good breakpoints between a $\sum$ and the its subscript. This leads to break an object that should not be broken even if it has some white rows inside. A practical solution is to consider a line with thickness grater than 1 pixel (but it's better to use metric dimensions). These ideas are also valid for scanned text but then we must take care of artifacts (and Leptonica can help a lot here).

We can also try to replace `pdftoppm` with another binding. The `Mupdf` is a good candidate but the binding is more complicated, the debugging is not easy, and the speed quite low. It's still better to use an external program.

That’s all

Thank you !