Line breaking

Victor Eijkhout

Notes for CS 594 – Fall 2004

- Problem: break horizontal list into lines
- n words: 2^n possibilities
- (much) better is possible
- Aim: visually even 'colour'

What is a paragraph?

- Boxes: words, math, other solid objects
- Glue: white space, possibly with stretch/shrink
- Penalties: prevent or force breaks

Boxes

- Words, formulas, actual boxes
- described by height, width,depth
- (hyphenation)

Penalties

- Inserted by user, macro, automatically
- Example: `\def~{\penalty10000 \ }`
- Automatic: paragraph ends with `\unskip\penalty10000\hfill`
- also: penalties for two consecutive hyphenated lines; last full line hyphenated

Glue

- Denotations: `\hskip 2cm plus 1cm minus .5cm`
- Automatically inserted: `\leftskip`, `\abovedisplayskip`
- Adding glue together: `2cm plus 1cm` and `2cm plus -1cm` total no stretch
- Infinite glue: `\hfill` or `\hskip 0pt plus 1fil` (`fill`, `filll`)
- infinite glue present: all other stretch/shrink ignored

Line break locations

- Foremost: at a glue, but only if preceeded by a non-discardable
- At a hyphen or hyphenation location
- At a penalty (this can override the above clauses)
- (complicated rules for defining extent of a word)

Examples

Centered text

```
\begin{minipage}{4cm}
\leftskip=0pt plus 1fil \rightskip=0pt plus 1fil
\parfillskip=0pt
This paragraph puts infinitely stretchable glue at
the left and right of each line.
The effect is that the lines will be centered.
\end{minipage}
```

Output:

This paragraph puts infinitely stretchable glue at the left and right of each line. The effect is that the lines will be centered.

Centered last line

```
\begin{minipage}{5cm}
\leftskip=0pt plus 1fil \rightskip=0pt plus -1fil
\parfillskip=0pt plus 2fil
This paragraph puts infinitely stretchable glue at
the left and right of each line, but the amounts cancel out
The parfillskip on the last line changes that.
\end{minipage}
```

Output:

This paragraph puts infinitely stretchable glue at the left and right of each line, but the amounts cancel out. The parfillskip on the last line changes that.

Hanging punctuation

- Put punctuation in the right margin
- Make right margin look more straight/solid

Here is a bit of text in a minipage, with too much punctuation. Every sentence, long or short, is overly, yes, overly, punctuated. This should show off 'hanging punctuation'.

hanging punctuation code

```
\newbox\pbox \newbox\cbox
\setbox\pbox\hbox{.} \wd\pbox=0pt
\setbox\cbox\hbox{,} \wd\cbox=0pt
\newdimen\csize \csize=\wd\cbox
\newdimen\psize \psize=\wd\pbox

\catcode`,=13 \catcode`.=13
\def,{\copy\cbox \penalty0 \hskip\csize\relax}
\def.{\copy\pbox \penalty0 \hskip\psize\relax}
```

Mathematical Reviews

- Reviewer signature separated from review text
- but if possible on the same line

This review is rather negative, almost devastating. A. Reviewer

This review is als very negative but it's longer than the other.
A. Nother-Reviewer

mathematical reviews code

```
\def\signed#1{\unskip
  \penalty10000 \hskip 40pt plus 1fill
  \penalty0
  \hbox{}\penalty10000
                \hskip 0pt plus 1fill
  \hbox{#1}%
                \par
  }
```

TEX's line breaking algorithm

Glue ratio

- Consider a line, and desired length
- compare to natural width, stretch, shrink

$$\rho = \begin{cases} 0 & \ell = L \\ (\ell - L)/X & \text{(stretch:) } \ell > L \text{ and } X > 0 \\ (\ell - L)/Y & \text{(shrink:) } \ell < L \text{ and } Y > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- note: negative for shrink

Badness

- Stretching/shrinking too far is 'bad':

$$b = \begin{cases} 10\,000 & \rho < 1 \text{ or undefined} \\ \min\left\{10\,000, 100|\rho|^3\right\} & \text{otherwise} \end{cases}$$

- Stretch beyond 100% possible, shrink not.
- Categories:

tight (3) if it has shrunk with $b \geq 13$
decent (2) if $b \leq 12$
loose (1) if it has stretched with $100 > b \geq 13$
very loose (0) if it has stretched with $b \geq 100$

Note $100 \times (1/2)^3 = 12.5$

- discourage 'visually incompatible' lines

Demerits

- Add together badness and penalties
- `\linepenalty`, `\doublehyphendemerits`, `\finalhyphendemerits`

Breaking strategies

First fit

- Wait until word crosses the margin, then
- if the line can be shrunk: shrink
- otherwise, if it can be stretched: stretch
- otherwise, try hyphenating
- otherwise, really stretch

Best fit and total fit

- (Best fit) Decide between stretch/shrink/hyphenate based on badness
- (Total fit) Add all badnesses together; minimize over whole paragraph

Program structure

- Loop over all words, check if feasible breakpoint

Program structure

- Loop over all words, check if feasible breakpoint
- Inner loop: over all previous words, check if feasible start of line

Program structure

- Loop over all words, check if feasible breakpoint
- Inner loop: over all previous words, check if feasible start of line
- Optimization: cutoff on previous words

Program structure

- Loop over all words, check if feasible breakpoint
- Inner loop: over all previous words, check if feasible start of line
- Optimization: cutoff on previous words
- 'active list'

Main program

```
active = [0]; nwords = len(paragraph)
for w in range(1,nwords):
    # compute the cost of breaking after word w
    print "Recent word",w
    for a in active:
        line = paragraph[a:w+1]
        if w==nwords-1:
            ratio = 0 # last line will be set perfect
        else:
            ratio = compute_ratio(line)
            print "..line=",line,"; ratio=",ratio
        if ratio<-1:
            active.remove(a)
            print "active point",a,"removed"
        else:
            update_cost(a,w,ratio)
    report_cost(w)
    active.append(w)
```

Data structure

```
def init_costs():
    global cost
    cost = len(paragraph)*[0]
    for i in range(len(paragraph)):
        cost[i] = {'cost':0, 'from':0}
    cost[0] = {'cost':10000, 'from':-1}
```

Cost function; first fit

```
def update_cost(a,w,ratio):
    global cost
    if a>0 and cost[a-1]['cost']<10000:
        if ratio<=1 and ratio>=-1:
            to_here = abs(ratio)
        else: to_here = 10000
        if cost[w]['cost']==0 or to_here<cost[w]['cost']:
            cost[w]['cost'] = to_here
            cost[w]['from'] = a-1
```

First fit, dynamic version

```
def update_cost(a,w,ratio):
    global cost
    if ratio<=1 and ratio>=-1:
        to_here = abs(ratio)
    else: to_here = 10000
    if a>0:
        from_there = cost[a-1]['cost']
        to_here = to_here+from_there
    else: from_there = 0
    if cost[w]['cost']==0 or to_here<cost[w]['cost']:
        cost[w]['cost'] = to_here
        cost[w]['from'] = a-1
```

example

```
You may never have thought of it, but fonts (better:    -0.11111
typefaces) usually have a mathematical definition somehow.  -0.66666
If a font is given as bitmap, this is often             0.888888
a result originating from a more compact description.   0.0
Imagine the situation that you have bitmaps at 300dpi, and  -0.77777
you buy a 600dpi printer. It wouldn't look pretty.      0.25
There is then a need for a mathematical way of          0.555555
describing arbitrary shapes. These shapes can also be   0.0
three-dimensional; in fact, a~lot of the mathematics in  -0.28571
this chapter was developed by a car manufacturer for    0.0
modeling car body shapes. But let us for now only       0.222222
look in two dimensions, which means that the curves     0.125
are lines, rather than planes.
```

Best fit cost function

```
def update_cost(a,w,ratio):
    global cost
    to_here = 100*abs(ratio)**2
    if a>0:
        from_there = cost[a-1]['cost']
        to_here = to_here+from_there
    else: from_there = 0
    if cost[w]['cost']==0 or to_here<cost[w]['cost']:
        cost[w]['cost'] = to_here; cost[w]['from'] = a-1
```

example

```
You may never have thought of it, but fonts (better:    -0.11111
typefaces) usually have a mathematical definition somehow.  -0.66666
If a font is given as bitmap, this is often a           0.5
result originating from a more compact description.     0.5
Imagine the situation that you have bitmaps at 300dpi, and  -0.77777
you buy a 600dpi printer. It wouldn't look pretty.      0.25
There is then a need for a mathematical way of          0.555555
describing arbitrary shapes. These shapes can also be   0.0
three-dimensional; in fact, a~lot of the mathematics in  -0.28571
this chapter was developed by a car manufacturer for    0.0
modeling car body shapes. But let us for now only       0.222222
look in two dimensions, which means that the curves     0.125
are lines, rather than planes.
```

Total fit, data structure

```
def init_costs():
    global cost
    nul = [0,0,0]
    cost = len(paragraph)*[ 0 ]
    for i in range(len(paragraph)):
        cost[i] = nul[:]
        for j in range(3):
            cost[i][j] = {'cost':10000, 'from':-2}
    for j in range(3):
        cost[0][j] = {'cost':10000, 'from':-1}
```

cost function

```
def update_cost(a,w,ratio):
    global cost
    type = stretch_type(ratio)
    to_here = 100*abs(ratio)**2
    if a>0:
        [from_there,from_type] = minimum_cost_and_type(a-1)
        to_here += from_there
    else: from_there = 0
    if cost[w][type]['cost']==0 or\
       to_here<cost[w][type]['cost']:
        cost[w][type]['cost'] = to_here;
        cost[w][type]['from'] = a-1
```

example

You may never have thought of it, but fonts (better:	-0.11111
typefaces) usually have a mathematical definition	1.2
somehow. If a font is given as bitmap, this is often a	-0.45454
result originating from a more compact description.	0.5
Imagine the situation that you have bitmaps at	1.0
300dpi, and you buy a 600dpi printer. It wouldn't look	-0.33333
pretty. There is then a need for a mathematical way of	-0.4
describing arbitrary shapes. These shapes can also be	0.0
three-dimensional; in fact, a~lot of the mathematics in	-0.28571
this chapter was developed by a car manufacturer for	0.0
modeling car body shapes. But let us for now only	0.222222
look in two dimensions, which means that the curves	0.125
are lines, rather than planes.	