Dynamic Fonts and Type 1 Fonts

by

Daniel M. Berry

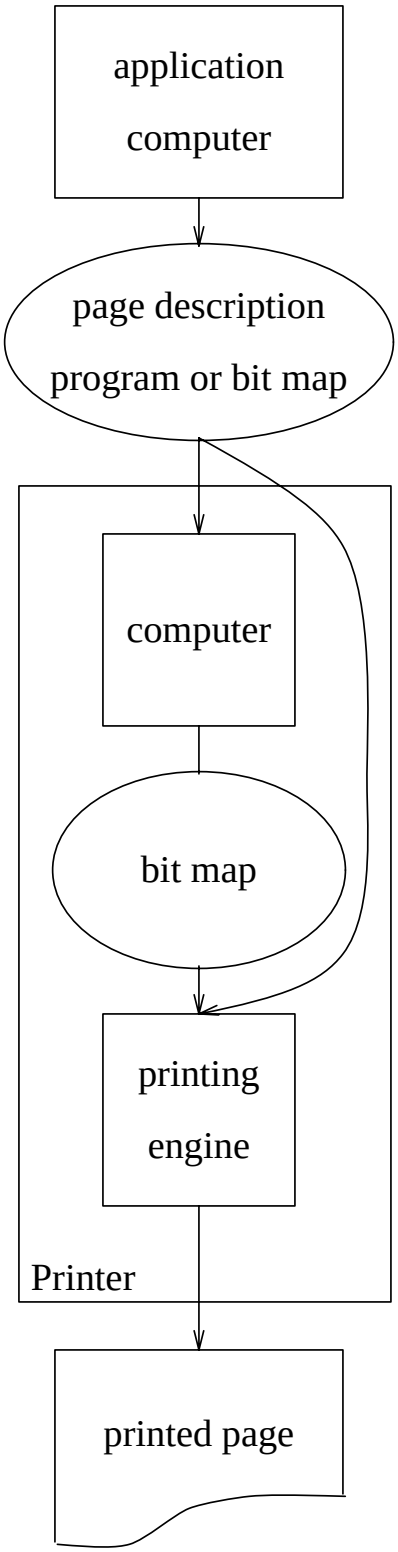
Problem with dynamic font solution

Font definition cannot be made type 1

Nu? What’s so bad about that?

First must explain type 1 fonts and implications on cache and ATM (Adobe Type Manager) mechanisms

But even before that, must explain how images are printed on printers



PRINTING CONFIGURATION

Two choices:

Compilation:

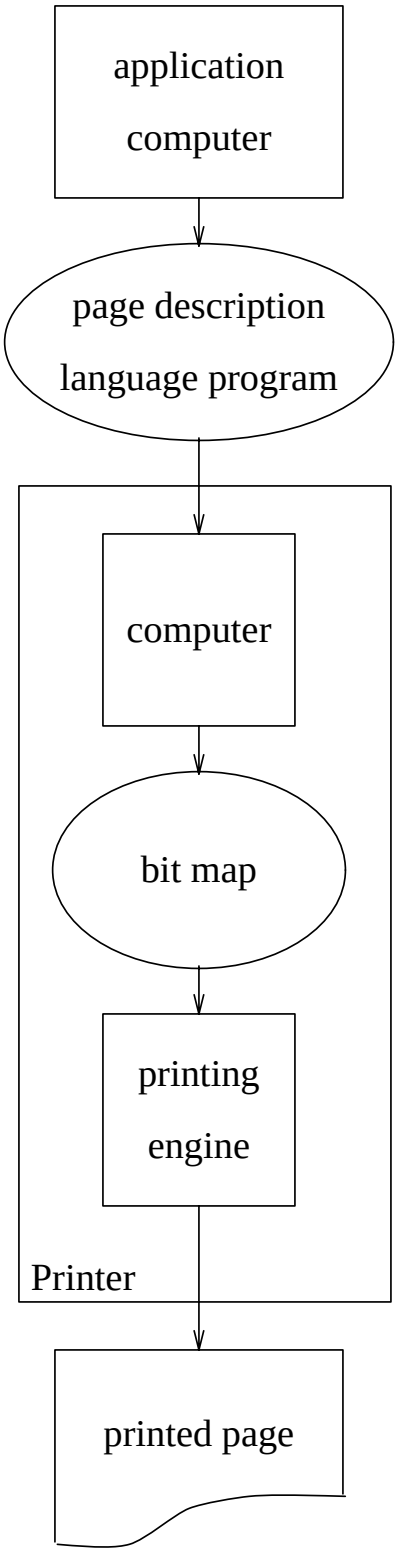
Application computer produces bitmaps, e.g.,

METAFONT

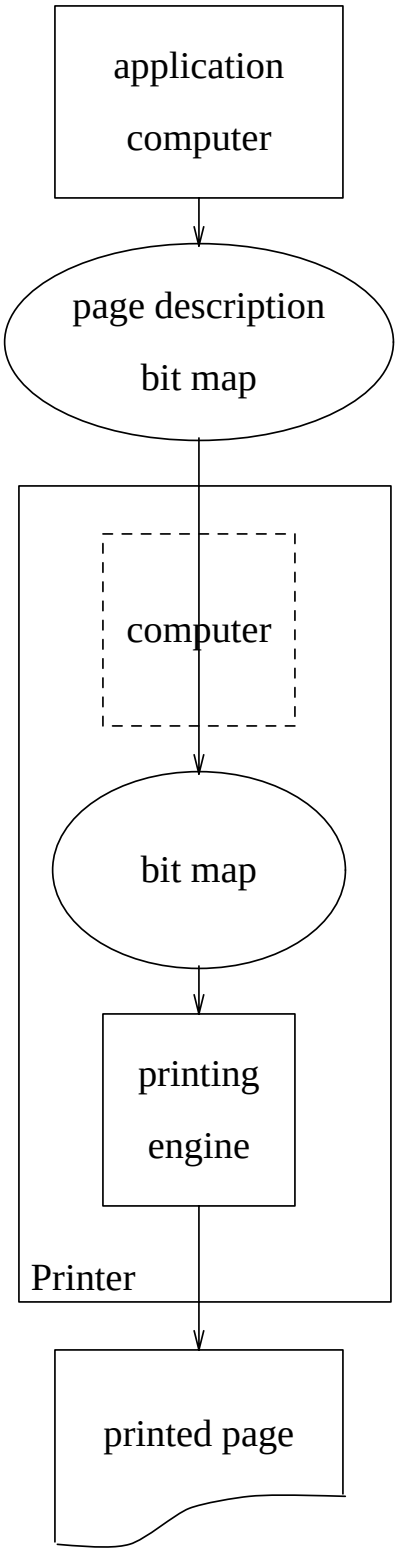
Interpretation:

Printer computer produces bitmaps, e.g.,

POSTSCRIPT printers, e.g., LaserWriter, QMS 800, Linotronic 300



INTERPRETATION



COMPILATION

Most popular interpretation language: POST-SCRIPT

Favorite method to specify characters in fonts (also with TrueType) is by specifying path of outline of character together with instruction to fill the inside of the path

When blow up or shrink (scale) bitmap get jaggies (staircases) or lose information

If outline specified as lines and curves through real number (actually in integral points at .000014 of inch), path can be scaled and filled with bits at current resolution without introducing jaggies or losing information

But, in the last analysis, need bitmaps

For later on, note that at 300 dpi ~ 118 dpc

An upper case letter of point size 10 (really 8 or 9 points high) is only 38 dots high

A lower case letter of point size 10 (really 5, 6, or 7 points high) is only 29 dots high

and stems are often only 2 or 3 dots wide!

For point size 5, half of that!

Problem with outline fonts

While outline font
    is scaleable and bitmap is not
and
    takes much less space than anyone bitmap
    and even less than that less of a complete
    collection of bitmaps for (only some)
    needed point sizes

Sending outline to printer,
    the printer tracing the outline,
    the printer filling the outline with bits, and
    the printer printing the bitmap

takes a lot more time than

sending bitmap to printer, and
    the printer printing the bitmap
directly

However, ...

In most fonts, once the point size is fixed, the bitmap of most characters does not change from one printing to another

So maybe can cache the bitmap for a given font, size, and character (F,S,C)

So that once a bitmap has been calculated, until the cache is filled and the bitmap is replaced by another, the same bitmap can be used if the same (F,S,C) is encoutered later, thus saving the tracing and filling of the outline

If cache is big enough to hold, say 3 full fonts worth of characters at size 12, then most documents will get up to bitmap downloading speed after the first printing of all the characters, generally on the first page

Notice how printer slows down for banner pages, pages with figures, and between troff and TEX jobs!

Cache is set up with a triple (F,S,C) indexing each saved bitmap

Cached bitmap is used whenever possible before calculating bitmap

LRU or similar replacement scheme when cache is full

In normal POSTSCRIPT outline fonts, no guarantee when start executing the outline for a particular (F,S,C) that its bitmap is constant

Therefore, it is required for a character definition to declare whether its generated bitmap is to be cached

> If to be cached, say "*width* `0` *bounding box* `setcachedevice`"
>
> If not, say "*width* `0 setwidth`"

at beginning of character definition

No algorithm to decide from character definition if the bitmap is constant (≡ to deciding whether program halts)

In case of dynamic font, for a given (F,S,C), bit-map is not constant

In the specific case of using dynamic font to implement keshida, for a given (F,S,C, character width), the bitmap is probably constant, but not for a given (F,S,C)

In the fully general case, such as with Jacques André's examples, the bitmap is not constant even for (F,S,C,W)

> ransom note font, using random number generator in character definition
>
> letter serifs that extend to page boundary

Now back to Type 1 fonts!

A type 1 font is a font specified in a so-called type 1 subset and extension of the full POST-SCRIPT which guarantees that no matter what you write, for any given (F,S,C) the bitmap is constant

> Sublanguage: no variables in arguments of path operators
>
> Extension: hinting for improved appearance at small sizes (relative to device resolution)

Unfortunately, hinting is not available in type 3 font definitions that can make use of the full (except for hinting) POSTSCRIPT language; some hints do not work when the ratio between widths of stems depends on variables

Also, the POSTSCRIPT engine can optimize many instructions if it knows that no variables can show up in argument lists

In fact, there are special versions of general path following instructions that permit no variable arguments and allow further optimization by the engine

Thus, on a general POSTSCRIPT printer, type 1 fonts are faster, and because of the hinting, better looking

They are faster both in the faster computing of the first bitmap for an (F,S,C) and guaranteed use of the cache!

Adobe Type Manager (ATM)

Basically, ATM is a POSTSCRIPT interpreter sitting inside your computer (rather than inside the printer) for computing bitmaps, at correct resolution, from font defintions for both

displaying on your screen

sending down to non-POSTSCRIPT printers

But Adobe does not really want you to have a full POSTSCRIPT interpreter on your computer

1. You won't buy the more expensive POSTSCRIPT printer if you can get by with a cheaper bitmap printer

2. It is easier to reverse engineer the POSTSCRIPT interpreter when it is sitting inside the computer (with all your secret-cracking

software) than when it is locked away, execute only, inside an inaccessible computer inside your printer

Type 1 fonts make the perfect compromise!

Enough of the PostScript language to compute the bitmaps for most font definitions

Not enough to print an arbitrary PostScript program that draws all sorts of fancy pictures

Restricted language interpreter does not even contain the really juicy secrets that are in the full PostScript interperter

But, sigh!

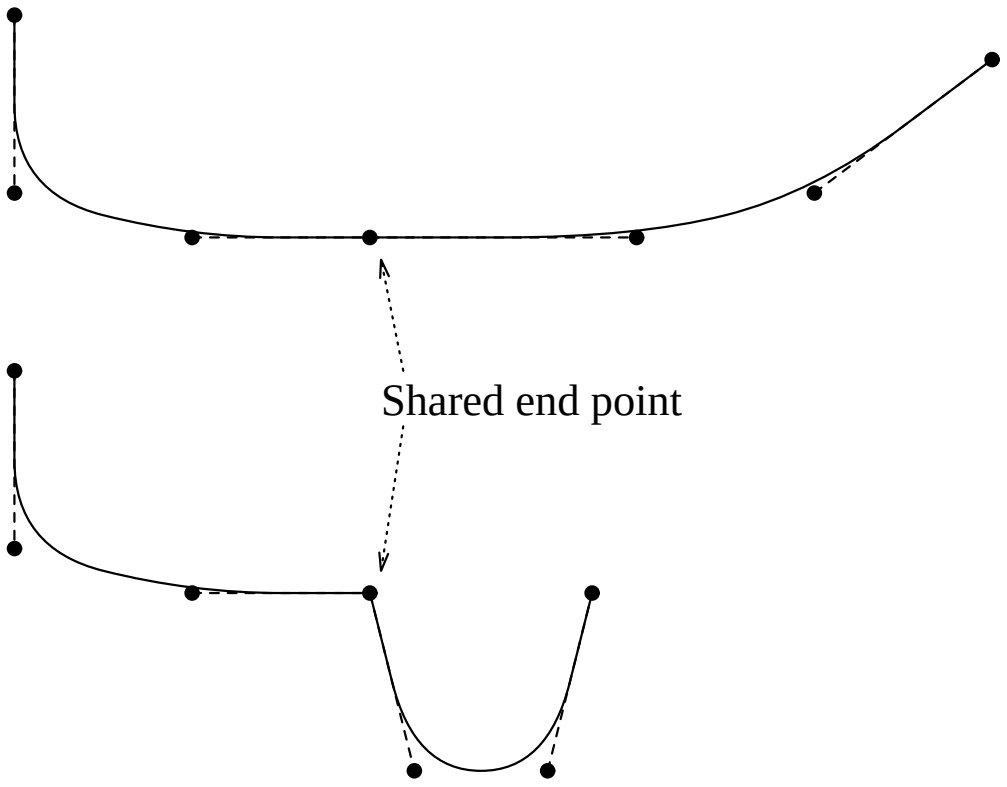
Will not be able to make type 1 fonts Arabic fonts with arbitrary stretching letter keshida and Farsi fonts with arbitrary slanting base lines

Open research problem: redesign the cache-type-1-font-ATM mechanism to work *well* when bitmaps are constant only for (F,C,S,W)!

The outline of any character is a series of {curves, lines, arcs} such that

elements sharing an end point are tangent to the same line which passes through the point, if the outline is supposed to be smooth through the point

elements sharing an end point have tangents at an angle at the point if the outline is supposed to have a corner at the point



Stretching should not introduce or remove any corner

This means that tangents at shared end points of elements should not change

All stretching is in horizontal direction only

A horizonal line is easy to stretch by A units:

> Add A to all X values at and to the right of the right end point of the line

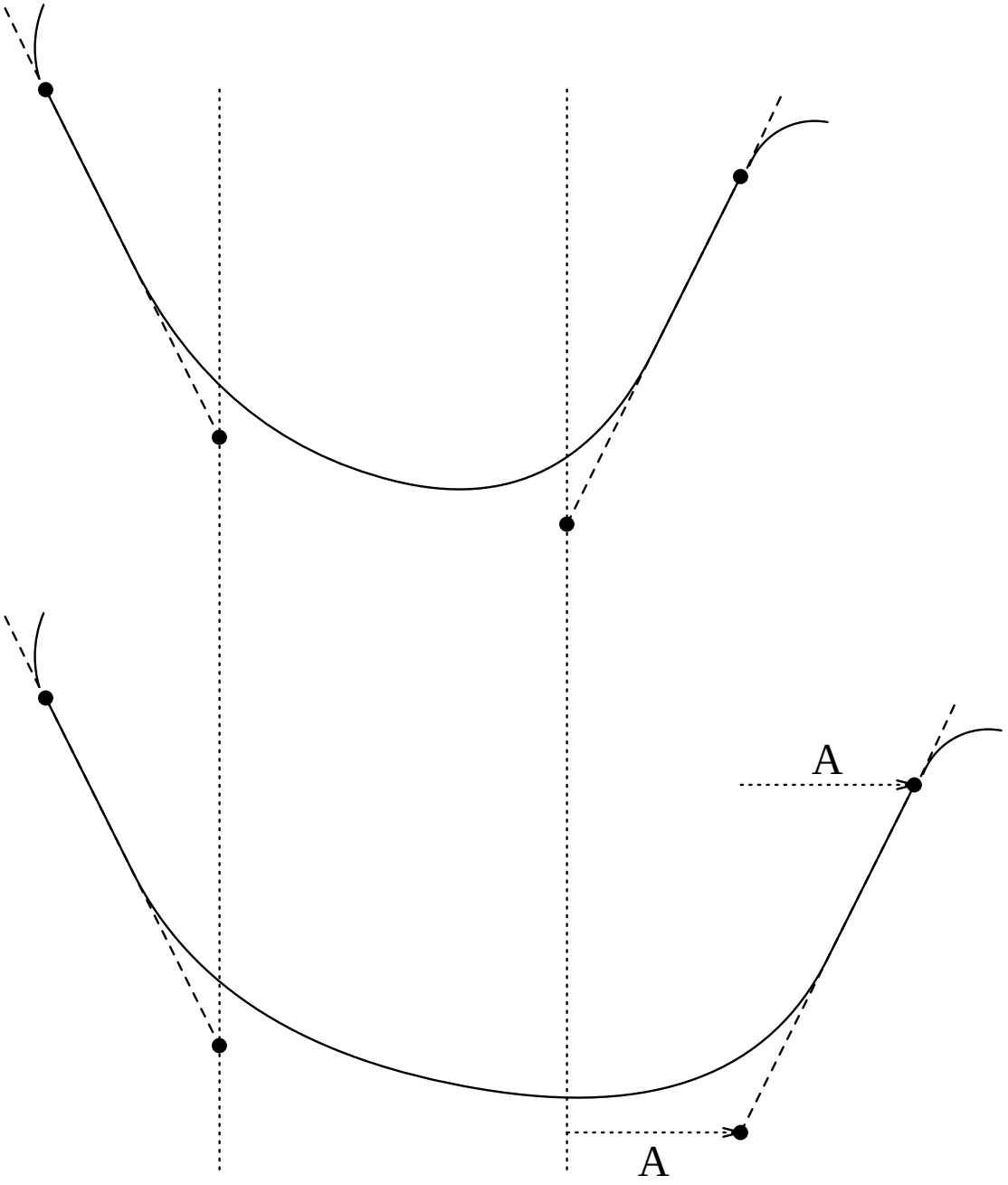
A vertical or slanted line cannot be stretched!

Curves with pronounce horizontal components can be stretched; only X values and no Y values should change

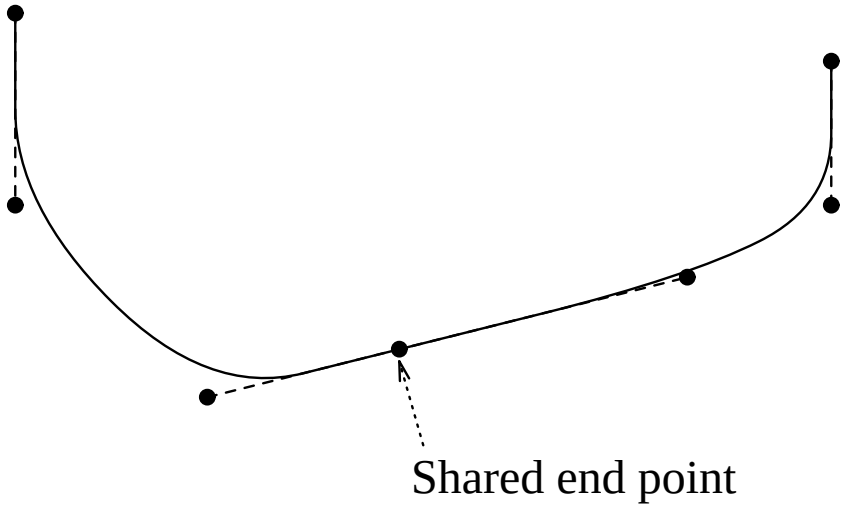
A four-point Beziér curve is easy to stretch if the stretching is in the interior between the two middle points

To stretch such a curve by A:

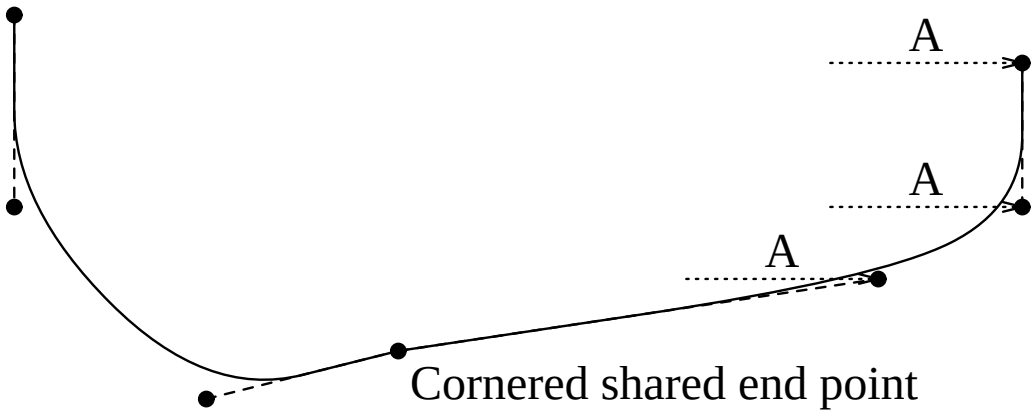
> Add X to the X values of the two right hand points and to all points to the right of the inner one of these


A
A

It is a problem to stretch through the shared end point of two Beziér curves that meet in a common tangent running through the shared point

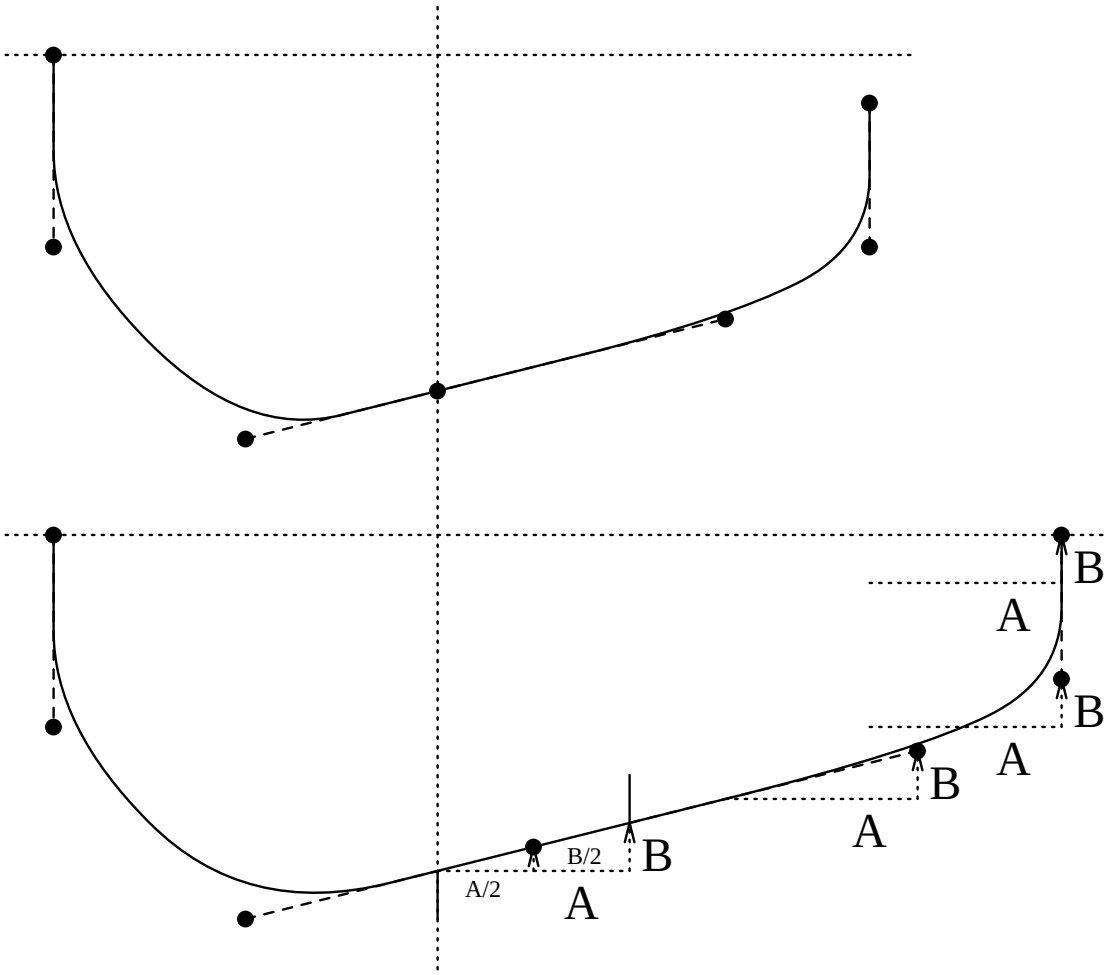


Cannot stretch by just increasing the X values of rightmost three points of the righthand curve



Changes slope of right hand curve through the shared end point and introduces corner that was not there before

Could avoid corner by preserving slope by increasing both X and Y values by amounts consistent with slope of tangent through the shared end point



But now the right end of the curve does not have the same Y value as before

Solution requires redesign of these two adjacent curves into three adjacent curves such that the shared end point of the two curves is in the interior of the middle of the three curves

Messy!

One special case of stretching through shared end point works!

When the tangents through the shared end points are completely horizontal!