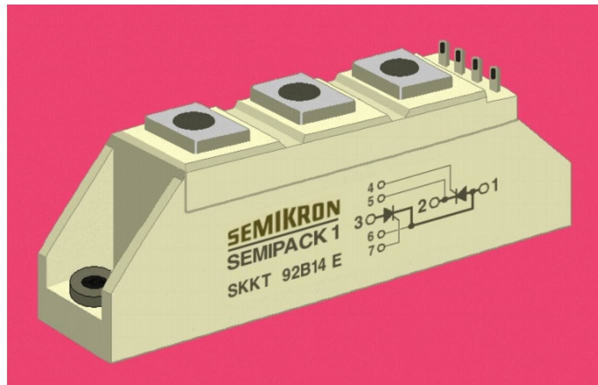
Some Bitmap Perspective Lettering Algorithms and Utilities

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2008 pub 6/08 as GuruGram #91
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

There are times and places where you might like to add perspective lettering or artwork to an existing .BMP image...



For instance, on an **eBay** product image, you could combine **both** your digital camera **and** a scanner. The camera to give you an attractive 3-D view. And the scanner to give you sharp accurate lettering **with an infinite depth of field**. Per the details in our **Image Post Processing Tools** tutorial of **GuruGram #88** .

The general problem is then to **distort** your flat or "detail" image so it can get positioned on the "main" image in a realistic and believable perspective manner. With its lettering or whatever getting smaller in the distance. And rising or falling as needed to fit properly. Perhaps aided by our **Bitmap Typewriter**.

Some nonobvious variations of our earlier **Architect's Perspective** of **GuruGram #90** can give us perspective pasteins. A new **BMPERLT1** utility can create the distorted images for you. Which can be transferred by the usual cut and paste.

Your perspective pasteins can be "right handed" or "left handed" ones, viewable either from "below" or "above". A different (and more complex) algorithm is required for "on the top" pasteins and is **not** provided for here.

BMPERLT1 is an ordinary textfile that executes by **Using Acrobat Distiller as a PostScript Computer**.

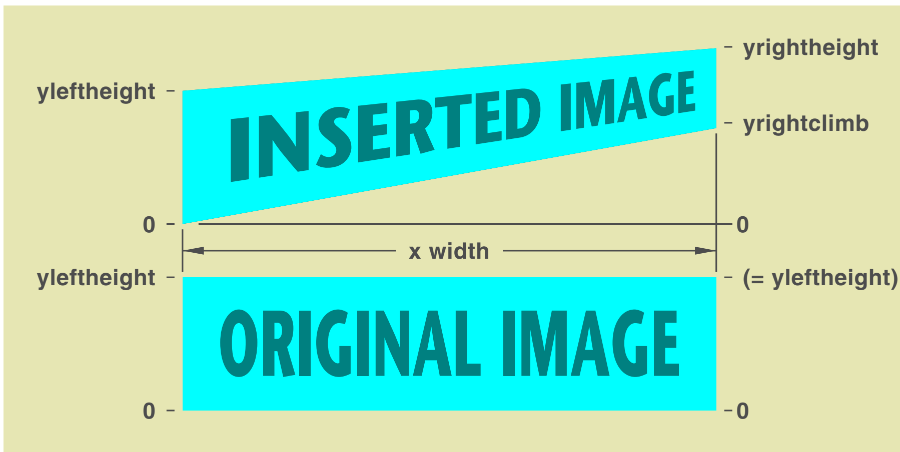
While the utility is written in **PostScript** and optionally uses my **Gonzo Utilities**, no knowledge of PostScript programming is required for routine use.

One Gotcha:

> **Acrobat Distiller versions newer than 8.1 default to preventing diskfile access.**
>
> **The workaround from Windows is to run "Acrodist -F" from the command line**
>
> **Solutions for other systems are found here.**

Some Geometry

We will assume your "detail" bitmap is the correct height and correct width needed for pastein...



Note that we do not normally want to just "shift up". For true perspective, lettering or artwork near the left end should end up **wider** than lettering or artwork near the right end. Proportional to the distance from the viewer.

Note further that all of the image is "squashed" somewhat. More so progressively to the right. Since the left margin tends to be "magnified", **the active image area should start very near the left image margin.**

We will define **yleftclimb** to always be zero. Regardless of whether the actual image is left or right handed, rising or falling.

The width and height of the original image will already be known. All you need to input are two new variables of **yrightheight** and **yrightclimb**.

As we'll shortly see, these will get related to two internal variables of **zzz** which is **the distance from the right image margin to the vanishing point**. And **ycen**, which **proportions the top and bottom image climb** as needed.

It turns out that **the locus of all possible vanishing points is a vertical line.** And all that **ycen** does is slide you up or down that line. For an "above", "below", or "centered" viewing.

Our **forward nonlinear transforms** will be…

```
xnew = xold * [zzz/(zzz + xold - xwide)]
ynew = (yold - ycen)*[zzz/(zzz + xold)] + ycen
```

This form of **xnew** includes an automatic scaling factor that keeps the original and the inserted images the same width.

Forward transforms are a "goes to" sort of function. Per **these details**, we will also need the reverse or "comes from" nonlinear transforms…

```
xold = xnew*[zzz/(zzz + xwidth - xnew)]
hhh = [zzz/(zzz + xold)]
yold = [ynew + ycen*(hhh-1)]/hhh
```

Two convenience operators can be derived from **yleftheight**, **yrightheight**, **yrightclimb**, and **xwidth**. As before, **yleftclimb** will always be defined and held at zero. Our first operator is **zzz**, which is the horizontal distance from the right side of the image to the vanishing point…

```
lrratio = (yrightheight - yrightclimb)/yleftheight
zzz = xwidth*[lrratio/(1 - lrratio)]
```

This fundamental perspective equation is based on proportional triangles. Curiously, it also happens to be very similar to the voltage out expression of a potentiometer.

Our second convenience operator is **ycen**. This decides how far the right end of the image will be slid up or down. **ycen** can be found by calculating its value at either **yrightheight** or **yrightclimb**. The latter is slightly easier…

ycen = yrightclimb*[1 - zzz/(zzz+xwidth)]

Getting Fancy

If we were only interested in the most common "climbs and gets smaller to the right" pastein, our output pastin bitmap would start at y=0 and have a height of **yrightheight**. If, instead, we want to handle all four cases of right side versus left side and climbing versus falling, we have to slightly modify the size and positioning of our output pastein bitmap…

pastetop = the greater of yleftheight and yrightheight
pastebot = the lesser of zero and yrightclimb

pasteheight = pastetop - pastebot

Since negative values are not allowed in an image's final x-y positioning, should **pastebot** be negative, a **pasteshift** value of **minus pastebot** will be needed to keep your pixels in bounds. This **only** applies to your final out-the-door bitmap and does not affect any earlier calculations.

There will typically be three areas to your final pastein bitmap: The useful stuff in the middle, typically a climbing or falling parallelogram; an upper out-of-bounds wedge, and a lower out-of-bounds wedge. **Most often, you will want your out-of-bounds wedges to be a pure white.** Allowing them to become transparent with a Paint or other copy.

Variables of **backgroundred**, **backgroundgreen**, and **backgroundblue** are optionally provided. Should you want your wedges to be visible for a special effect, rather than transparent for the usual pastein.

A Working Utility

You can explore the above perspective lettering and artwork equations by using our **BMPERLT1.PSL** utility. What follows here can be better understood by having **BMPERLT1.PSL** up in a separate textfile window.

Much of the program style and coding follows the techniques we looked at earlier in our **Architect's Perspective** tutorial.

Our new utility needs only an input bitmap and five pieces of information: The names and locations of the original image bitmap and the new inserted image to be created;. Plus **yleftheight**, **yrightheight**, and **yrightclimb**.

A reminder again that **yleftclimb** is always held at zero. And that an **Acrodist-F** command line is needed to activate disk access on newer Acrobat releases.

As usual, our utility is altered in a standard ASCII textfile and then sent to Acrobat Distiller. By **Using Distiller as a General Purpose PostScript Computer**. Distiller will then read the original bitmap, make the necessary distortion calculations, and then write a new pastein bitmap for you.

Your original bitmap sets the **width** and the **left height** of the inserted image. The other two variables of **yrightheight** and **yrightclimb** decide whether you'll end up left or right-handed or above or below in your final image.

With the present code, **yleftheight** and **xwidth** pixel counts must be manually redefined and must match the vertical and horizontal size of your original bitmap. I'm not yet convinced these values should be auto captured.

As with earlier examples, our utility is arranged in four parts: The **Gonzo** core utilities and array-of-strings generic code; exportable lesser utilities; the specific perspective lettering routines; and an ending example. Only the example code portion normally needs modified for each new project.

Our main proc is called **makeperspaste**. It first converts the input bitmap into the usual **PostScript** array-of-strings for manipulation. The needed output bitmap pastin size is then calculated, followed by a new triad of arrays-of-strings for the output. Called **redAOS2**, **greenAOS2**, and **blueAOS2**.

The actual work is done by a **procperspaste** subproc, followed by moving and rewriting the newly created array-of-strings to an output .BMP bitmap disk file.

procperspaste is basically a loop within a loop that exercises each and every pixel. The x pixel position is done as the outer loop since this can have speed advantages. The loops find the needed **zzz**, **ycen**, **hhh** (an intermediate work variable), **oldx**, and **oldy**. These are passed to yet another subproc we call **remappixels** to do the actual pixel-by-pixel perspective transforms.

remappixels first does a limit check to find out whether "live" or "background" pixels are to be mapped. The usual background will be white for copy-and-paste transparency, but other colors can be chosen for debug or special effects.

The live pixels undergo a **bilineal interpolation** to find the best approximation to their values on the original bitmap. Speed modified interpolators are used for the red, green, and pixel planes. The results are converted to integer 0-255 values and written to the array-of-strings that become your output perspective bitmap.

Some Further Comments

Any attempt to resize a bitmap to less than half its height can introduce dropouts or Moire effects. These have **not** been compensated in this code and, in general, can be rather difficult to handle gracefully. **BMPERLT1.PSL is best used with your right side height at least one half of the left side height.**

There are no dropout issues on magnification.

Speed has not yet been optimized. Especially the need to test each individual pixel for in or out of bounds. But since most pasteins are rather small, the present code should be acceptable. Taking a few seconds at most on faster machines.

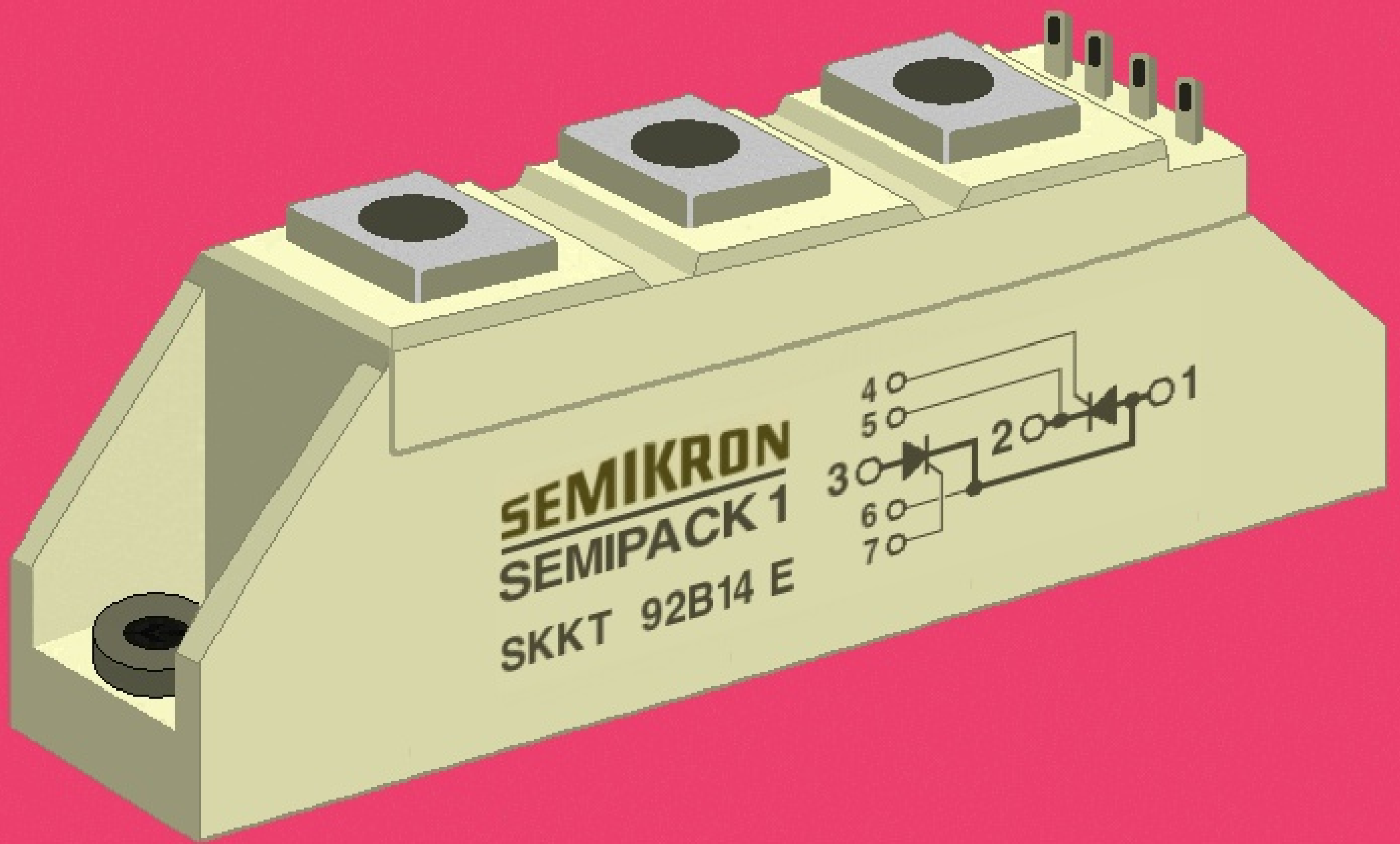
For More Help

A sample input file is available as **FLATART1.BMP**.

The basic full two dimensional **.BMP bitmap to PS Array of Strings** tutorial appears as **BMP2PSA.PDF** with its actual PS utility at **PIXINTP1.PSL**. Two recent additions were **AIRBRUSH.PDF** and **ARCHPERS.PDF**. Additional .BMP manipulation enhancements and expansions are planned.

News about the latest updates and addons should first appear in **WHTNU08.ASP** or later blog entries. Other bitmap manipulation and posptroc files are reviewed in **POSTPROC.PDF**.

Similar tutorials and additional support materials are found on our **PostScript** and our **GuruGram** library pages. As always, **Custom Consulting** is available on a cash and carry or contract basis. As are seminars and workshops. For details, you can email **don@tinaja.com**. Or call **(928) 428-4073**.


SEMIKRON
SEMIPACK 1
SKKT 92B14 E
4
5
3
6
7
2
1