Don Lancaster's

ASK THE GURU

October, 1990

A new PostScript wish list
Using the Adobe Distillery
Double distilled compiling
Graphics on a Bezier surface
Meals-in-minutes packaging

Rumor has it that the highly touted Apple IIe card for the Mac LC is incapable of running *AppleWriter*! Especially in its modem record mode essential for useful PostScript work.

I'll try and work up some sort of a fix on this, so stay tuned.

Let's see. Apple has now quietly dropped the *Apple IIc Plus* and its related printers. The end of an era, for sure.

1. **PLATFORM INDEPENDENT DISPLAY POSTSCRIPT.**

   Reason: To eliminate the ludicrous transformations needed for on-screen displays. To speed up editing and debugging.

2. **EXPLICIT SUPPORT OF SHARED SCSI COMM.**

   Reason: Most of today's PostScript printing is severly baud rate limited. Shared SCSI comm provides as much as a 50:1 speedup.

3. **FONT LOCK PERMANENTLY REMOVED FROM PATHFORALL.**

   Reason: Instantly available font paths greatly simplify and ridiculously speed up the nonlinear transformations needed for perspective, starwars, banner, flag, and similar creative display typography.

4. **A READABLE AND RECORDABLE FRAMEDEVICE.**

   Reason: Dramatic speedups of most step-and-repeat routines. Greatly simplified book-on-demand publishing, especially when combined with CD-ROM. Incredibly powerful editing and post-processing possibilities.

5. **CEXEC OPERATOR DOCUMENTED IN DETAIL.**

   Reason: To level the playing field and give all end users the same power tools that only a favored few have today.

6. **A VIDEO FRAMEDEVICE OUTPUT**

   Reason: Fast and useful debug tool, especially when speeding up code. Also opens up alternate applications such as printed circuit production, CAD/CAM, Santa Claus machines, sign routers, and vinyl cutters.

7. **SYSTEM MONITOR AVAILABLE TO ALL**

   Reason: To allow rapid end user correction of problems such as the fatal copypage flaw in the duplex mode.

8. **A FULLY OPEN FONT CACHE ARCHITECTURE**

   Reason: By allowing the cacheing of anything of any size, instead of just small typography, such things as multiple logos and other calculation intensive routines could end up running much faster.

9. **FULL SCSI HARD DISK DOCUMENTATION**

   Reason: Host interaction with the actual on-disk font cache allows many speedup and post-processing opportunities.

Fig. 1 – My PostScript wish list.

Apple does have some interesting new publications out. They have a new *Apple II Guide* and the revised *Macintosh Development Tools and Languages Guidebook*. And for those of you interested in disabilities, the handicapped, or special education, there's a revised *Connections Guide: Computer Resources for the Disabled*. There is also a new *Apple Computer Resources for Special Education and Rehabilitation*.

Contact your dealer or local user group for info on the first three. The final book is available through *DLM Teaching Resources* for $19.95.

The 3-M folks have now come up with *Post-It* notes in a precut and laser printable sheet form. Product #7709. Availability does still seem limited at this writing. The obvious uses include custom notepads, both personal and for resale.

Two great PostScript books now include that *gray* book from Glen Reid, otherwise known as *Thinking in PostScript*, and the *beige* book authored by Ross Smith, and titled *PostScript – A Visual Approach*. I do try and stock *all* of the very best PostScript books for you. Write or call for a complete list.

A reminder here that we now do have this great PSRT RoundTable up on *GEnie* with lots of PostScript and all the other *Ask the Guru* and *LaserWriter Corner* stuff on it. And including lots of fully debugged and ready-for-use downloads of most of the printed routines you see here and in our earlier reprints.

You can voice call (800) 638-9636 for connect info. I have also now put together a PSRT sampler disk for you. Call or write me per the end trailer for details.

What Do You Really Want From PostScript?

Did you know that all of those LaserWriter printers have a built-in system monitor that lets you view

memory, add or alter any portions of the internal code, and change or repair any or all routines? All you do is jumper two secret connections in the serial cable to activate your monitor. A set of predetermined bytes in your ROM chips will then preselect the "privilege" level of access you are granted.

This is one example of the many secret features of the PostScript language which you end users are prohibited from using. At one time long ago and far away, some of the restrictions may have made slight sense to somebody, somehow. But all they do today is grievously slow down, incapacitate, and very much infuriate the end user.

Figure one is a list of some of the more crucial intentionally crippling restrictions forced upon all of you PostScript end users. I like to call this my *wish list*.

Note that most insiders already have access to some or all of these features. Many could be found by nosing around the larger university related BBS systems. The others by rounding up all the usual suspects. Thus, there appears to be a rather uneven playing field. And a very foggy one as well.

Note also that no new technology is needed here. A few simple words on an interoffice memo or two can instantly provide everything the entire wish list asks for.

And one of the big side effects of fulfilling my wish list would be to make PostScript so powerful to all the end users that it would become totally unassailable by any clone or competing technology.

Word has it that the upcoming PostScript Level II should have unlocked font paths. Which would be one major step in the right direction. One down, nine to go.

Sigh.

Compiled PostScript?

Sort of. And certainly usefully.

You'll find two main methods of handling most any computer language. If the language is *interpreted*, each instruction gets used when and as it occurs at the highest level. If your language gets *compiled*, some special process is gone through to make the final run time code be as compact, fast, and as efficient as is reasonably possible.

Compiling also divorces the run time code far away from the original applications package that generated that code. Which can get handy for such things as providing fully device independent printed circuit art on BBS systems without the need for any applications support.

This can solve a crucial problem of both the hardware hackers and all their technical editors. Using pseudo-compiled PostScript in its EPS format to directly provide the end-users accurate and first quality printed circuit layouts, dialplates, templates, whatever.

Repeated use of compiled code often could run much faster than interpreted code at the triple costs of all your time required for your compiling process, deferred error messages, and moderately longer program files.

Compiled code is also quite hard to edit or change. To the point that you'll nearly always edit and then compile and rarely vice versa. This, of course, is true of just about *any* compiling. Not just PostScript.

PostScript is an interpreted language. While a true compiling of your PostScript output down to the machine language level is not yet readily available, there are several tricks you can attempt to pseudo-compile your PostScript code.

The results can be an incredible speedup. For instance, we routinely make up a three column and 6000 character page with two figures, a header and footer in three seconds

```
/bdef { bind def } bind def
/ldef { load def } bind def
/selectfont { exch findfont exch scalefont setfont } bdef
/DF { selectfont currentfont def } bdef
/BEGINPAGE { pop /pagesave save def } bdef
/ENDPAGE { pop pagesave restore showpage } def
/AW { moveto awidthshow } bdef
/F /setfont ldef

1 BEGINPAGE
/F1 /Palatino-Bold 10 DF -0.6 0 32 0.1 0 (Compiled  PostScript?) 128.73 720 AW
/F2 /Palatino-Roman 9.5 DF
0.65 0 32 0.1 0 (Sort of. And certainly usefully.) 110 695 AW
F2 F 1.9 0 32 0.108918 0 (You'll find two main methods of ) 110 682.5 AW
F2 F 0.00576103 0 32 0.1 0 (dealing with any computer language.) 100 670 AW
F2 F 1.9 0 32 0.282986 0 (If the language is ) 100 657.5 AW
/F3 /Palatino-Italic 9.5 DF 1.9 0 32 0.282986 0 (interpreted) 187.447 657.5 AW
F2 F 1.9 0 32 0.282986 0 (, each ) 232.788 657.5 AW
F2 F 0.855241 0 32 0.1 0 (instruction gets used when and as it ) 100 645 AW
F2 F 0.634014 0 32 0.1 0 (occurs at the highest level. If the lan- ) 100 632.5 AW
F2 F 0.222581 0 32 0.1 0 (guage is ) 100 620 AW
F3 F 0.222581 0 32 0.1 0 (compiled) 138.481 620 AW
F2 F 0.222581 0 32 0.1 0 (, a special process is ) 173.234 620 AW
F2 F 1.54727 0 32 0.1 0 (gone through to make the final run ) 100 607.5 AW
F2 F 0.33065 0 32 0.1 0 (time code be as compact, as fast, and ) 100 595 AW
F2 F 0.65 0 32 0.1 0 (as efficient as possible.) 100 582.5 AW
F2 F 1.10939 0 32 0.1 0 (Compiling also "divorces" the run ) 110 570 AW
F2 F 0.459408 0 32 0.1 0 (time code far away from the original ) 100 557.5 AW
F2 F 1.74072 0 32 0.1 0 (applications package that generated ) 100 545 AW
F2 F 1.62958 0 32 0.1 0 (that code. Which can get handy for ) 100 532.5 AW
F2 F -0.158258 0 32 0.1 0 (such things as providing fully device-) 100 520 AW
F2 F 1.48734 0 32 0.1 0 (independent printed circuit artwork ) 100 507.5 AW
F2 F 0.397416 0 32 0.1 0 (on ) 100 495.0 AW
/F4 /Palatino-Roman 9 DF 0.397416 0 32 0.1 0 (BBS) 113.788 495.0 AW
F2 F 0.397416 0 32 0.1 0 ( systems without needing any ) 129.811 495.0 AW
F2 F 0.65 0 32 0.1 0 (applications package support.) 100 482.5 AW
F2 F 0.65 0 32 0.1 0 (  ) 110 470 AW
1 ENDPAGE

% Total non-header length:              1808 bytes
% Typical baud rate limited run time:   1064 milliseconds on QMS Turbo PS820
% Raw PostScript run time:              150 milliseconds on QMS Turbo PS820
```

Fig. 2 – Example of Distillery Compiled PostScript.

or so. From AppleWriter on a IIe.

Most of the time, there is no point in compiling PostScript unless you are (1) completely happy with your uncompiled version in its final form except for its speed; (2) are going to use your file at least several times in the future, and (3) are using a comm channel that is not at all baud rate limited, especially with the possibly longer compiled code.

Or, separately (4) if you do not want the end user to require or use the original applications package.

Or, finally (5) if you are using a horrendously slow phototypesetter for all of your final high resolution camera-ready art. Even with only a single use, you can sometimes use a Pseudo-compiling to save bunches of time and money.

Thus, while compiled PostScript code is outstanding for book-on-demand publishing from hard disk based files or for a wide distribution of a printed circuit board pattern, you might not want it for everyday routine use.

There are several approaches to compiling. Rather than the strict definition of "make it all machine language", we'll define a compiling as any one-time stunt you can pull to reduce to an absolute minimum the work PostScript has to perform during your future printings. For instance, there is no point in making justification calculations each time. Instead, you save *only the results* of those calculations and use these results instead.

One simple and rather obvious compiling trick is called *binding*. If you simply use a *bind def* instead of *def*, PostScript objects get linked directly to other objects, eliminating the name lookups. Those *//name* immediately executed names also do the same thing. Binding sometimes gives you a ten to fifteen percent speedup. But it can also make any later code modifications difficult to do. Full details in the red book.

By far the most general way of compiling PostScript code is with an *Adobe* product called the *Distillery*. Adobe has graciously uploaded a freeware copy of the Distillery to our *GEnie* PSRT RoundTable as our file #186 DISTILL.PS.

At any rate, what the Distillery does is intercept any operator that would make any marks on the page, such as *lineto*, *awidthshow*, and all the other markers.

It then asks "What is the absolute minimum amount of info needed to use this operator?" And then either returns that info to your host for recording, or else can write it to a selected hard disk file. The new file becomes your run time code.

Their Distillery works great for some routines and only so-so for others. Often you have to try it and see. You just might want to modify your programming style or your original applications packages to make better use of the Distillery.

While the Distillery often makes code much shorter, at times it can make a simple and short routine into an unbelievably complex data string. Obviously, in these cases, you may want to mix the original code with the compiled code to get the tightest final package.

While it is amazing what it does and how well it generically handles pretty near any input, there are some Distillery bugs. For instance, superscript and subscript aren't supported in the font matrix unless you make the character height and width at least slightly different. And original code that insists on using individually spaced words, rather than calling PostScript's powerful *awidthshow* operator, will produce inherently longer code. As much as three times longer on the average.

The Distillery does not seem to know how to save a path for reuse. This means that your *entire* path gets repeated for such things as a fill and stroke. Obviously, you could

```
/F {exch findfont exch scalefont setfont} bind def
/A {moveto 0 exch 0 32 5 2 roll awidthshow} bind def

 /Palatino-Bold 10 F
 -0.6 .1(Compiled  PostScript?)128.7 720 A

 /Palatino-Roman 9.5 F
 .65 .1(Sort of. And certainly usefully.)110 695 A
 1.9 .1089(You'll find two main methods of)110 682.5 A
 .005761 .1(dealing with any computer language.)100 670 A
 1.9 .2830(If the language is)100 657.5 A
 1.9 .2830(, each)232.8 657.5 A
 .8552 .1(instruction gets used when and as it)100 645 A
 .6340 .1(occurs at the highest level. If the lan-)100 632.5 A
 .2226 .1(guage is)100 620 A
 .2226 .1(, a special process is)173.2 620 A
 1.547 .1(gone through to make the final run)100 607.5 A
 .3307 .1(time code be as compact, as fast, and)100 595 A
 .65 .1(as efficient as possible.)100 582.5 A
 1.109 .1(Compiling also "divorces" the run)110 570 A
 .4594 .1(time code far away from the original)100 557.5 A
 1.74072 .1(applications package that generated)100 545 A
 1.630 .1(that code. Which can get handy for)100 532.5 A
 -.1583 .1(such things as providing fully device-)100 520 A
 1.487 .1(independent printed circuit artwork)100 507.5 A
 .3974 .1(on)100 495.0 A
 .3974 .1( systems without needing any)129.8 495.0 A
 .65 .1(applications package support.)100 482.5 A

 /Palatino-Italic 9.5 F
 1.9 0.2830 (interpreted)187.4 657.5 A
 0.2226 .1 (compiled)138.5 620 A

 /Palatino-Roman 9 F
  .3974 .1 (BBS)113.9 495.0 A

 showpage


% Total non-header length:          1282 bytes
% Typical baud rate limited run time:  754 milliseconds on QMS turbo PS820
% Raw PostScript run time:            133 milliseconds on QMS turbo PS820
```

Fig. 3 – Example of Double Distilled Compiled PostScript.

hand edit your intermediate code to get around this limitation.

Figure 2 shows you some typical stock Distillery output.

I have just released the latest version #13 of my new *Guru's Gonzo Justify Power Tools*. This code now includes an optional automatic and Distillery-compatible but text-only compiling feature. You can easily download a shareware copy and its documentation from PSRT on *GEnie*, or else write me directly for a free printed listing.

It is also possible to use a sneaky new technique which I call *double distilling*. Most PostScript end users most of the time become baud rate limited, especially if they are using AppleTalk. If you can pick up find any tricks that *shorten* your distillery files while adding only a minor computation speed penalty, you might further speed up your run times by as much as an extra 35 percent or so.

For instance, you could drop any leading or trailing zeros. For most users most of the time, you could round off to four place accuracy.

You can eliminate spaces before or after self-delimiting parentheses. And trailing spaces within strings. And empty strings. At a tiny speed penalty, you can eliminate the "0", "0", and "32" and their intervening spaces which are used with a horizontally set *awidthshow*.

You can also sort your files so that each font only gets changed *once* on each page. The Distillery already predefines "made" or "scaled" font dictionaries such that only the very fast and VM free *setfont* operator is used for an actual change. So your gain by sorting is not spectacular. Still, as the figures show us, you do gain a fair amount of speed and save enough characters this way to end up more than worthwhile.

Figure 3 shows you how your Distillery file can be further shortened by *double distilling*. At this time, this process is custom and labor intensive. Let me know if you have any further interest in seeing improved versions of this double distilling process.

When you are baud rate limited, your processing time is directly proportional to the total number of characters in your file. The "Baud Rate Limited" times shown here are based on the AppleTalk on a typical Mac that's running at a 17 kilobaud effective rate. Your own baud rate limiting could easily become much worse than this.

This is why we *must* have shared SCSI comm *now*.

Any New Product Packaging Ideas?

All of your laser printed output, book-on-demand published stuff, or similar software goodies should be delivered to the end customer in an attractive, "see-thru", and protected package of some sort.

The traditional method that the printshops use is the same *shrink wrapping* used in your grocery store. You can get further information on shrink wrap products in *Printer's Shopper*, or *Quick Printing*, *Printing Impressions*, or the *Instant & Small Commercial Printer* mags.

But I think I've found a better method. A good quality look, better touch, and best durability.

A number of years ago, several firms introduced *Meals in Minutes* pouch packaging systems. These were intended for sealing leftovers in plastic baggies that could be frozen, boiled, or nuked. The best of these included a powered vacuum pump that actually sucked the extra air out of the package.

I think these are absolutely great, and these are one of the few things around here that I actually use for its intended purpose. Well, some of the time anyway.

Apparently the product bombed, or at least the market got flooded, for all the discount houses are now offering these at bargain prices.

In particular, do check out that *DecoSonic* from *The Lighter Side*, the *Seal-A-Meal* from *COMB*, *PaknSave* from *Heartland*, or *Vac-U-Pac* from *Damark*. The prices are in the $19-$39 range. Do make sure your model includes a vacuum pump.

Now for the secret part. These work beautifully with the plain old *Zip-lock* bags, especially those heavy duty "freezer" versions.

One tip: You usually throw away the zip-lock part and just make the rest of the plastic tightly fit your product. To do this, simply peel away your scrap side while your plastic is still hot and you are still holding the lever down. With some practice, you'll get a tight and uniform edge seal every time.

We use them here at *Synergetics* to seal all our book-on-demand and

Fig. 4 – Accurate Graphics and Text on a Bezier Surface.

```
% Copyright c 1991 by Don Lancaster and Synergetics, Box 809, Thatcher AZ, 85552.All
% commercial rights reserved. Personal use permitted if this header remains preset.
% LaserWriter Secrets book+disk $29.50 VISA/MC. Free voice helpline (602) 428-473.
% Available on GEnie PSRT library as #202 TWIXTBEZ.PS.

/mychardict 10 dict def mychardict dup

/F {611 {258 326 mt 373 320 391 304 423 177 ct 444 177 li 444 510 li 423 510 li 39 383 373 367
258 361 ct 258 590 li 258 641 265 650 301 649 ct 387 647 li 544 643 552 508 562 47 ct 582 474
li 582 681 li 17 681 li 17 660 li 69 660 99 627 99 575 ct 99 106 li 99 54 69 21 1721 ct 17 0 li 340
0 li 340 21 li 288 21 258 54 258 106 ct cp }} put dup /R {722 {498 0 mt 695 0 li 65 21 li 683 21
673 16 645 57 ct 455 335 li 516 350 561 374 594 425 ct 634 488 610 569 569 611 ct 15 666
416 681 338 681 ct 26 681 li 26 660 li 78 660 108 627 108 575 ct 108 106 li 108 5478 21 26 21
ct 26 0 li 349 0 li 349 21 li 297 21 267 54 267 106 ct 267 317 li 292 317 li cp 267 606 mt 267 633
288 645 308 647 ct 383 655 417 622 437 586 ct 451 560 458 476 442 427 ct 412 338 36 350
267 349 ct cp }} put dup /E {667 {262 326 mt 377 320 395 304 427 177 ct 448 177 li448 510 li
427 510 li 395 383 377 367 262 361 ct 262 590 li 262 641 269 650 305 649 ct 391 64 li 548 643
556 508 566 474 ct 586 474 li 586 681 li 21 681 li 21 660 li 73 660 103 627 103 57 ct 103 106 li
103 54 73 21 21 21 ct 21 0 li 593 0 li 637 206 li 610 206 li 539 62 462 18 319 30 t 268 34 262 48
262 99 ct cp }} put dup /space {250 {250 0 mt }} put dup /O {778 {733 336 mt 733 44 681 569
591 627 ct 525 669 455 690 388 690 ct 321 690 251 669 185 627 ct 95 569 43 454 43 36 ct 43
218 95 103 185 45 ct 251 3 321 -18 388 -18 ct 455 -18 525 3 591 45 ct 681 103 733 18 733 336
ct cp 388 8 mt 370 8 349 12 327 23 ct 287 42 259 80 236 147 ct 221 190 211 261 211336 ct 211
411 221 482 236 525 ct 259 592 287 630 327 649 ct 349 660 370 664 388 664 ct 406 64 427
660 449 649 ct 489 630 517 592 540 525 ct 555 482 565 411 565 336 ct 565 261 555 10 540
147 ct 517 80 489 42 449 23 ct 427 12 406 8 388 8 ct cp }} put dup /N {722 {617 -1 mt 617 575 li
617 627 645 660 697 660 ct 697 681 li 498 681 li 498 660 li 550 660 578 627 578 57 ct 578 249
li 230 681 li 20 681 li 20 660 li 47 660 57 639 71 622 ct 108 577 li 108 106 li 10 54 80 21 28 21
ct 28 0 li 227 0 li 227 21 li 175 21 147 54 147 106 ct 147 529 li 588 -10 li cp }}put dup /T {667
{409 649 mt 433 648 498 661 554 603 ct 603 553 602 505 607 479 ct 629 479 li 629 61 li 30
681 li 30 479 li 52 479 li 57 505 56 553 105 603 ct 161 661 226 648 250 649 ct 250106 li 250 54
220 21 168 21 ct 168 0 li 491 0 li 491 21 li 439 21 409 54 409 106 ct cp }} put po

/pcurveto {8 copy /y3 exch def /x3 exch def /y2 exch def /x2 exch def /y1 exch def/x1 exch def /y0
exch def /x0 exch def} def /xtt {x3 x2 3 mul sub x1 3 mul add x0 sub tt 3 exp mul 2 3 mul x1 6
mul neg add x0 3 mul add tt dup mul mul add x1 3 mul x0 3 mul neg add tt mul add x add} def
/ytt {y3 y2 3 mul sub y1 3 mul add y0 sub tt 3 exp mul y2 3 mul y1 6 mul neg add y 3 mul add tt
dup mul mul add y1 3 mul y0 3 mul neg add tt mul add y0 add} def
```

Fig. 5a – Excerpts From my Twixt Bezier Routines…

book+disk products, as well as for protecting business cards and out-the-door orders. The sealers are also great for protecting stuff at trade shows from wear and tear.

You could get "real" polyethelene tubing through the *Associated Bag Company* or from *Harwil*, but that is no fun at all.

As usual, all of our *Names and Numbers* have now been gathered together for you in the ending appendix to this volume.

For this month's contest, just tell me about any other gross abuse or obscene misuse that you've been making in applying something well away from its intended purpose. For fun or profit.

This Month's PostScript Utility?

I thought I'd throw a real heavy at you this month. A new method that I've recently developed that lets you quickly and accurately place text and graphics on any surface defined by a pair of cubic spline curves. One typical "flag waving" example is shown you in figure four, while enough excerpts of my PostScript code to get you going appear in figure five.

Other obvious uses for my new TWIXTBEZ.PS routines are for banner fonts, text on a cylinder, twisted film effects, pennants, any curved perspective surfaces, fancy logos, lettering on a scroll, a funhouse mirror reversal, and just about any other distortion you can think of. The processing speed is ridiculously faster than the earlier *pixel line remapping* we looked at earlier.

The method uses bits and pieces of stuff from previous columns and from the *Ask The Guru III* reprints. Especially the nonlinear transformations, the length of Bezier curves, and stopping at the selected point along a single Bezier curve.

The new black magic added here, though, will involve *automatically converting straight lines into smooth flowing curves, given only the end points of those lines!* Thus, the tops and bottoms of all your letters will properly curve themselves to conform to your new surface.

In the routines shown here, all vertical lines are kept that way. This is what you usually will want in any text-oriented logo or clip art.

Where do you start? Your text or graphics must first get *completely* converted into combinations of one of four allowed routines. These are *mt* movetos, *li* linetos, *ct* curvetos, and *cp* closepaths. Characters in any font must be predefined into a 1000 point high proc definition of form *{ -charwidth- {char defs using only mt, li, ct, and cp}}*.

Now for the tricky part. You can optionally convert any of those *lt* linetos into a curve that smoothly flows over your new surface! To do this, you first set *showlinesascurves* to *true*. Your code then takes any linetos and neatly breaks them up into individual *numcurvesperlineto* segments. Each individual segment is then changed into a cubic spline, both of whose influence points lie *halfway* along their original line path, thus "bending" it.

What this does is convert your original line into an end aligned group of fairly weak curved splines. This guarantees you are at the right point on the surface at each end of each spline, rather than the "short cut" your single straight line would attempt. Since each spline itself can end up curved, this further helps "adhere" to your new surface.

In figure four, four curves per lineto are used to bend the tops and bottoms of all the letters, while sixteen curves per lineto are used for your upper and bottom lines which have to sweep clear across the entire surface. Naturally, the more curves you use, the better looking your result, but the slower the calcs.

While not immediately obvious, the *closepath* operator also needs modified so it does not take any shortcuts. One rather easy way to handle this is to close up the path yourself, and then only use *closepath*

as a zero-length formality.

You select your upper and lower Bezier surfaces using an *[x0 y0 ang0 x1 y1 ang1]* matrix. Regular *Ask the Guru* readers will recognize this as the same matrix employed for a two point gonzo curvetracing. In this matrix, *x0* and *y0* are the starting coordinates of your curve, while *ang0* is the angle you want to head out in as a first step.

Similarly, *x1* and *y1* form your ending coordinates of your curve, and *an*g1 is the final direction the curve is heading in just as it reaches the end point. As usual, 0 degrees is dead east, while 90 degrees points due north.

With a single curve, you can be "bent", have an "S" shaped *inflection point*, could have a single cusp, or might even loop. For fancier paths, you can repeat the entire process as often as you need to.

The upper curved path will only be approximate. This does happen because we want to keep verticals straight up and down, and because things occur faster in the "t" space along the "more bent" portions of any spline. Any exact solution here might involve repeatedly solving ugly cubic equations.

Thus, if your upper path doesn't give you quite what you want on the first cut, just bend it a little more or a little less, and it should fall in place the way you intend it to.

Drawing and graphics gets done by using your *sketchmode* feature. You should call the *sketchmode* proc immediately before any drawing is done. This scales things so you are working directly in points on your Bezier surface. The x direction is now *along* your curved surface, and your y direction stays vertical.

Characters are shown using my new nonlinear variant of the *ashow* operator called *twsshow*. The height and width of each individual letter is selected by a */fontsize [width 0 0 height 0 0] def* matrix.

What you do with your character path is set by *bezcharproc*.

Instead of the simple fill shown here, you could outline and stroke, shadow, choke, spread, flow, or do most any combination of the special effects. Note that this is insanely more flexible than what the original *ashow* operator permitted.

You can do individual character kerning by following the examples detailed in figure five. In this case, we have tightened the distance on either side of the "O" and made the final "T" somewhat narrower than is normal. Note that things happen faster on the "more bent" portions of your curved surface. You can also mix character heights and vertical offsets for special effects, especially for menus or product labels.

We will be seeing several more examples of these non-linear text transformations right here, on my *GEnie* PSRT, and in my *LaserWriter Secrets* reprint series.

Those rumored improvements in PostScript level II should make an on-the-fly font path grabbing and nonlinear transformation a quick and easy process. Sadly, we are not quite there yet.

Fully commented TWIXTBEZ.PS routines appear in the *GEnie* PSRT library as file #202. You can call me for a free printed copy.

As a second contest this month, just send me any old clip art, use example, or any extensions to my TWIXTBEZ.PS routines that you've found handy. ❄

```
/bezierlength {pcurveto /oldx x0 def /oldy y0 def /blength 0 def 0 1 numpoints 1 sub div
1.0001 {/tt exch def xtt ytt /newy exch def /newx exch def newx oldx sub dup mul newy oldy
sub dup mul add sqrt blength add /blength exch def /oldy newy def /oldx newx def} for
blength }def /numpoints 30 def

/buildpatharray {/mat exch def mat 0 get /xx0 exch def mat 1 get /yy0 exch def mat 3 get /xx3
exch def mat 4 get /yy3 exch def xx0 xx3 sub dup mul yy0 yy3 sub dup mul add sqrt 3 div
gunghofactor mul /zz0 exch def mat 2 get cos zz0 mul xx0 add /xx1 exch def mat 2 get sin
zz0 mul yy0 add /yy1 exch def xx3 mat 5 get cos zz0 mul sub /xx2 exch def yy3 mat 5 get sin
zz0 mul sub /yy2 exch def mark xx0 yy0 xx1 yy1 xx2 yy2 xx3 yy3 bezierlength]} def

/setlowerpath {buildpatharray /lowerpath exch def} def /setupperpath {buildpatharray
/upperpath exch def} def /botftt {lowerpath twixtftt} def /topftt {upperpath twixtftt} def

/twixtftt {/arry exch def /tt exch def arry 0 get /x0 exch def arry 1 get /y0 exch def arry 2 get
/x1 exch def arry 3 get /y1 exch def arry 4 get /x2 exch def arry 5 get /y2 exch def arry 6 get
/x3 exch def arry 7 get /y3 exch def x3 x2 3 mul sub x1 3 mul add x0 sub tt 3 exp mul x2 3
mul x1 6 mul neg add x0 3 mul add tt dup mul mul add x1 3 mul x0 3 mul neg add tt mul add
x0 add y3 y2 3 mul sub y1 3 mul add y0 sub tt 3 exp mul y2 3 mul y1 6 mul neg add y0 3 mul
add tt dup mul mul add y1 3 mul y0 3 mul neg add tt mul add y0 add} def

/mt {savecp exch fontsize 0 get mul .001 mul xoffset add exch fontsize 3 get mul .001 mul
yoffset add nlt moveto} def /origli {savecp exch fontsize 0 get mul .001 mul xoffset add exch
fontsize 3 get mul .001 mul yoffset add nlt lineto} def /ct {savecp 3 {6 2 roll exch fontsize 0
get mul .001 mul xoffset add exch fontsize 3 get mul .001 mul yoffset add nlt} repeat curveto}
def /cp {closepath} def /savecp {2 copy /ycp exch def /xcp exch def} def /lict {/yy exch def /xx
exch def yy ycp sub numcurvesperlineto dup add div /yd exch def xx xcp sub
numcurvesperlineto dup add div /xd exch def numcurvesperlineto {xcp xd add ycp yd add 2
copy xcp xd 2 mul add ycp yd 2 mul add ct} repeat} def /li {showlinesascurves {lict}{origli}
ifelse} def

/twsshow {/msg exch def /yoffset exch def /xoffset exch def /strrx (X) def msg {strrx exch 0
exch put strrx dup ( ) eq {pop (space)} if cvn mychardict exch get exec exec bezcharproc
fontsize 0 get mul 0.001 mul extrakern add xoffset add /xoffset exch def} forall} def
/sketchmode {/fontsize [1000 0 0 1000 0 0] def /xoffset 0 def /yoffset 0 def} def /nlt {/yyyy
exch def /xxxx exch def upperpath 1 get lowerpath 1 get sub /unitheight exch def xxxx
lowerpath 8 get div /tt exch def tt botftt /ybotpos exch def /xbotpos exch def tt topftt /ytoppos
exch def /xtoppos exch def xbotpos yyyy unitheight div ytoppos ybotpos sub mul ybotpos add
} def

%  //// demo - remove or alter before reuse. ////

100 200 moveto 10 dup scale /showlinesascurves true def /gunghofactor 2 def 106 45 {dup
mul exch dup mul add 1.0 exch sub} setscreen [2 1 80 32 14 55] setlowerpath [2 14 45 32 16
50] setupperpath

sketchmode 0.2 setlinewidth 1 setlinejoin 1 setlinecap /numcurvesperlineto 16 def 0 13 mt 34
13 li 34 0 li 0 0 li cp gsave 0.95 setgray fill grestore stroke

/numcurvesperlineto 4 def /extrakern 0.1 def /bezcharproc {fill} def /fontsize [5.5 0 0 10 0 0]
def 2 3 (FREE F) twsshow xoffset 0.3 sub yoffset (O) twsshow xoffset 0.3 sub yoffset (N)
twsshow /fontsize [4.5 0 0 10 0 0] def xoffset yoffset (T) twsshow showpage quit
```

Fig. 5b – ...Twixt Bezier Routines, continued.