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

1. PLATFORM INDEPENDENT DISPLAY POSTSCRIPT.
   Reason: To eliminate the ridiculous 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 severely 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.

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 caching 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.

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.

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
nosing around the larger university features. Many could be found by having access to some or all of these this my wish list PostScript end users. I like to call restrictions forced upon all of you more crucial intentionally crippling infuriate the end user. all they do today is grievously slow sense to somebody, somehow. But restrictions may have made slight prohibited from using. At one time language which you end users are secret features of the PostScript access you are granted.

preselect the “privilege” level of bytes in your ROM chips will then monitor. A set of predetermined in the serial cable to activate your do is jumper two secret connections repair any or all routines? All you memory, add or alter any portions on an interoffice memo or two can handling most any computer lan-

gUAGE. If the language is, 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

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 nose 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

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 common 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 DISTILLPS.

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

---

```latex
/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
/0.6 .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
/0.6 .1(Compiled PostScript?)128.7 720 A
/%Palatino-Italic 9.5 F
/0.6 .1(A moveto 0 exch 0 32 5 2 roll awidthshow) bind def

/GEnie
/0.8552 .1(instruction gets used when and as it)128.7 720 A
/0.65 .1(applications package support.)100 482.5 A
/0.3974 .1(on)100 495.0 A
/0.3974 .1(independent printed circuit artwork)100 507.5 A
/0.3974 .1(systems without needing any)129.8 495.0 A
/0.65 .1(applications package support.)100 482.5 A

0.2226 .1(compiled)138.5 620 A

/0.65 .1(applications package that generated)100 657.5 A
1.09 .1(If you are using a)110 570 A
0.4594 .1(time code far away from the original)100 557.5 A
1.74072 .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
68.3

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 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 Danmark. 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
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" polyethylene tubing from 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 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 re-mapping 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 closepaths, and cp closepathex. Characters in any font must be predefined into a 1000 point high proc definition of form {’charwidth’ [chardefs using only mt, li, ct, and cp]}.

Now for the tricky part. You can optionally convert any of those hl 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 character formed by hl linetos, or all newlines and words breaking between them, and converts them 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 doesn’t cut up the path yourself, and then only use closedpath...
as a zero-length formality.

You select your upper and lower Bezier surfaces using an \([x_0 \ y_0 \ ang_0 \ x_1 \ y_1 \ ang_1]\) matrix. Regular Ask the Guru readers will recognize this as the same matrix employed for a two point gonzo curvetracing. In this matrix, \(x_0\) and \(y_0\) are the starting coordinates of your curve, while \(ang_0\) is the angle you want to head out in as a first step.

Similarly, \(x_1\) and \(y_1\) form your ending coordinates of your curve, and \(ang_1\) 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 x1 y1 ang1 as a zero-length formality.

ASK THE GURU

x0 point gonzo curvetracing. In this the same matrix employed for a two GEnie guru

Similarly, \(x_0\) and \(y_0\) are the starting coordinates of your curve, while \(ang_0\) is the angle you want to head out in as a first step.

Similarly, \(x_1\) and \(y_1\) form your ending coordinates of your curve, and \(ang_1\) 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 the place 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 height 0 0] def matrix.

What do 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 TWIXTB EZ.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 TWIXTB EZ.PS routines that you’ve found handy. ☺

---

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