No. 50 November-December 1989 $3.95

THE MICRO TECHNICAL JOURNAL

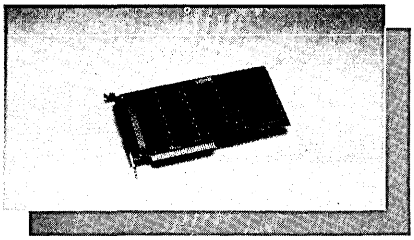
MICRO CORNUCOPIA

3D Graphics In Depth

This whole issue is a project. Put UNIX on your 386, build a board to grab video images, build another to analyze voices, and then display all your graphics in 3D.

0 74470 19388 3 11

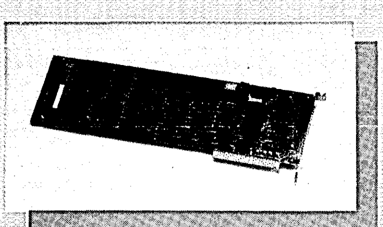
Turn your PC/XT/AT Into a powerful Laboratory and Engineering tool...



Digital I/O and Counter Card

- **32 Digital Input Channels**
  - TTL compatible
  - Low loading: 0.2 mA at 0.4V input
- **32 Digital Output Channels**
  - TTL compatible
  - Driving capacity: Sink 24 mA, source 15 mA
- **Intel 8253 Timer/Counter**
  - 3 channels of timer/counter
  - Breadboard area for flexible user configuration

PCL-720..............................$160.00



12 Bit Multi-Lab A/D+D/A+DIO+Counter

- **Analog Input (A/D converter)**
  - 16 single-ended channels, 12 bit
  - Input range: +5V to -5V, +1V to -1V
- **Analog Output (D/A converter)**
  - 2 channel, 12 bit
  - 0 ~ +5V full range
- **Digital I/O**
  - 16 channels each, TTL compatible
- **Counter**
  - 1 channel of timer/counter
  - Programmable pacer function

PCL-712..............................$295.00

Introducing...

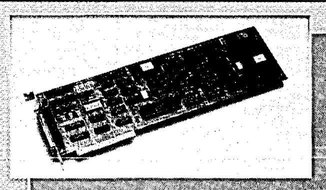
The PC-LabCard Family

Only from HSC Electronic Supply

IBM PC/XT/AT and it's compatible models are moving into Industrial/Laboratory applications at an increasing rate. The reasons for this include their price/performance ratio and short user learning curve. PC-based data acquisition boards are now taking the place of the traditional data loggers or recorders which cost several times more.

"**PC-LabCard**" is a family of add-on cards to turn your PC into a high performance data acquisition/ testing system at an attractive price.

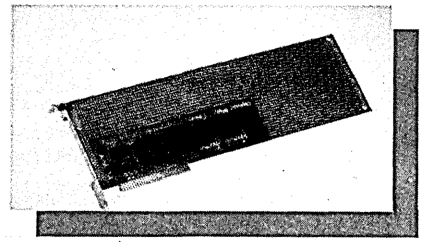
It includes not only the hardware cards, but also the software, accessories and application support packages which come together to make a thoughtful solution to your PC-based automation needs.



Stepping Motor Control Card

- Independent, simultaneous operation of up to 3 motors
- Programmable speed from 3.3 to 3410 PPS
- Built-in acceleration control
- One clock (Pulse, Direction) or two clock (CW, CCW Pulses) output mode
- Opto-isolated pulse-train outputs
- Read-back step position
- Crystal-based timing
- 8 bit digital Input and Output

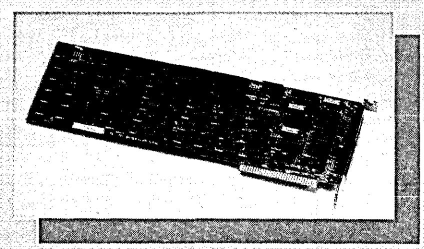
PCL-738..............................$395.00



Prototype Development Card

- Large breadboard area (3290 holes)
- Independent memory and I/O address decoders built-in
- Memory and I/O ports are jumper selectable
- All bus signals are buffered, marked, and ready for use

PCL-750..................................$74.00



14 Bit Super-Lab A/D+D/A+DIO+Counter

- **Analog Input (A/D converter)**
  - 16 different channels
  - 14 bit, 25K/sec sample rate
  - Input range: +5V to -5V, +1V to -1V
- **Analog Output (D/A converter)**
  - D/A Channels: 1 standard, 1 optional
  - 14 bit, +/- 5V full rage
- **Digital I/O**
  - 16 channels each, TTL compatible
- **Counter**
  - 1 channel of timer/counter
  - Programmable pacer function

PCL-714..................................$495.00

This partial listing of PC-LabCard represents only one of the many [illegible]

Call the ***Halted Electronic Resource*** BBS! (408) 732-2814

We ship UPS-COD
We ship to APO/FPO

HSC of Santa Clara
3500 Ryder Street
Santa Clara, CA 95051

Electronics · Prototyping Supplies
Computers · Lasers

HSC of Sacramento
5549 Hemlock St.
Sacramento, CA 94928
(916) 338-2545

HSC of Santa Rosa
6819 Redwood Dr.
Cotati, CA 95841
(707) 792-2277

VISA · MasterCard · American Express

Call Now! **Outside California *(800) 4-HALTED***
California Residents *(408) 732-1573*

Terms: Minimum order $10. California residents add 7% sales tax. Prepaid orders sent freight C.O.D. or call for charges. Shipping will be added to credit card and C.O.D. orders. $2 handling charge on orders less than $25. Send money order or certified check. Please do not send cash. Some items limited to stock on hand. Prices subject to change without notice. Foreign orders use credit card only.

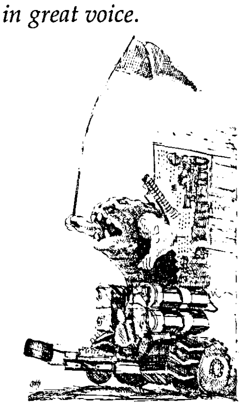
Reader Service Number 11

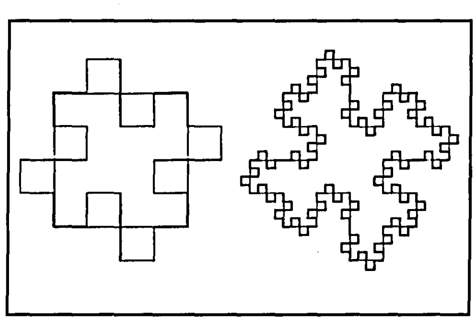
THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

NOVEMBER/DECEMBER 1989 - ISSUE NO. 50

FEATURES



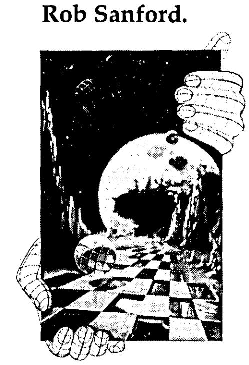


COLUMNS

FUTURE TENSE

COVER

Cover Illustration by Rob Sanford.



THE MICRO TECHNICAL JOURNAL
MICRO CORNUCOPIA

Editor and Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Larry Fogg
Cary Gatton

Contributing Writers
Anthony Barcellos
Bruce Eckel
Michael S. Hunt
Scott Ladd
Laine Stump

Advertising & Distribution
Erik Huston

Accounting
Sandy Thompson

Reader Services
Nancy Ellen Locke

Graphic Design & Production
Carol Steffy

MICRO CORNUCOPIA (ISSN 0747-587X) is published bi-monthly for $18 per year by Micro Cornucopia, Inc. 155 NW Hawthorne, Bend, OR 97701. Second-class postage paid at Bend, OR and additional mailing offices. POSTMASTER: Send address changes to MICRO CORNUCOPIA, PO Box 223, Bend, OR 97709.

SUBSCRIPTION RATES:

1 yr. (6 issues)	$18.00
2 yr. (12 issues)	$34.00
3 yr. (18 issues)	$48.00
1 yr. Canada & Mexico	$26.00
1 yr. Other foreign (surface)	$36.00
1 yr. Foreign (airmail)	$50.00

Make all orders payable in U.S. funds on a U.S. bank, please.

CHANGE OF ADDRESS:
Please send your old label and new address to:

MICRO CORNUCOPIA
P.O. Box 223
Bend, Oregon 97709

CUSTOMER SERVICE:
For orders and subscription problems call 503-382-8048, 9 am to 5 pm, Pacific time, M-F.

TECHNICAL ASSISTANCE
For help call 503-382-8048, 9 am to noon Pacific time, M-F
1-800-888-8087

BBS - 24 hrs. 300-1200-2400 baud
8 Bits, No Parity, 1 Stop Bit 503-382-7643

ISSN 0747-587X

The Audit Bureau

AROUND THE BEND

By David J. Thompson

Fifty Isn't Old If You're A Magazine...

Half A Century

As Sandy and I hand-stapled all 500 copies of the first issue of *Micro C* at our kitchen table, we had no idea that eight years later *Micro C* would still be around, and we'd be working on our 50^{th} issue. Yep, and in those 50 issues there have been so many changes in this silly technology that even the changes have changed.

And innovations? Boy, have we seen innovations. Once we had only software. Now we have freeware, shareware, crippleware, vaporware, underware, and beware.

We've progressed from public domain programs with bugs to commercial programs with bugs. (Actually, commercial programs always had bugs, it's just that the nasty little critters are getting harder to work around.)

Nowadays we're aiming for a computer on every desk. When we began Micro C, we hoped for one in every garage.

Anyway, with the big 50 upon us, I guess that means the honeymoon is over. It's time to get down to serious business—nose to the grindstone and ear to the ground. We've got to get off the fence and take a stand. ((Larry, is there anything we can do about these stupid clichés without Dave noticing? -Cary) (Probably not. -Dave))

Salmon BBQ at the Port Alberni SOG

Continued on page 73

Lattice Tools & Libraries for DOS and OS/2

We're the company that writes the language *and* the tools. Our libraries give you hundreds of ready-to-use functions, compatible with our compiler and other products.

Compiler Companion — $100
Ten proven UNIX-like tools provide a complete programming environment, and reduce file handling tasks, regardless of programming language.

dBC III & dBC III Plus — $250/$500
Networking & non-networking libraries let you write fast C programs to create, access and update files compatible with dBASE III.

Communications Library — $250
A comprehensive set of high- and low-level functions for asynchronous communications programs using XMODEM, YMODEM, KERMIT or ASCII protocols.

C-Food Smorgasbord — $150
A selection of utility functions including a BCD decimal arithmetic package, I/O functions, IBM PC BIOS interface, a Terminal Independence Page and functions including directory, clock and string.

Curses V Library — $125
84 C screen management functions and macros that help you port between UNIX System V and the PC.

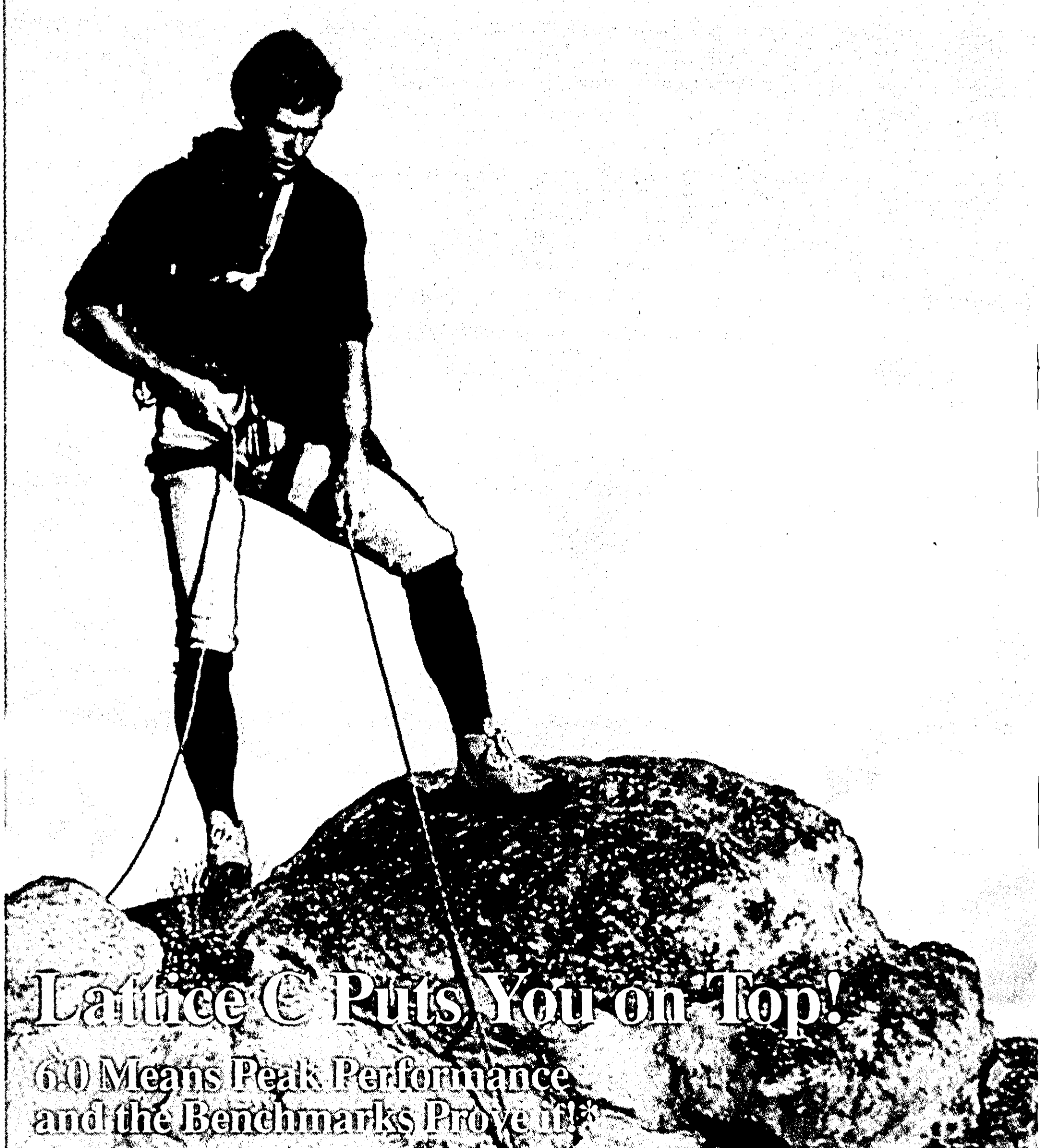
SSP/PC — $350
More than 145 subroutines eliminate tedious and difficult mathematical programming. Includes scientific, engineering and statistical mathematical routines as extensive and accurate as similar packages used on mainframes.

PANEL Plus with Source — $495
A collection of interactive screen design tools plus more than 150 functions to help you create application screens.

HighStyle –Programmer's Publishing Tool — $375
HighStyle gives you everything you need to create attractive, highly polished documentation for your programs. Automatically creates tables and charts! Includes: Word Processor, Page Previewer, Snapshot Utility, Style Guides, Spelling Checker, Icon Editor, Bar Code Generator, Font Manager, and more.

To order Lattice C 6.0, send $250 in check or money order to: Lattice, Inc., 2500 S. Highland Avenue, Suite 300, Lombard, IL 60148. Or order by credit card at (800) 444-4309. FAX # (312) 916-1190, TELEX 532253.

Reader Service Number 153



Lattice C Puts You on Top!

6.0 Means Peak Performance and the Benchmarks Prove it!*

Only Lattice C 6.0 for DOS and OS/2 can make molehills out of mountains. Because it's a complete development system that incorporates every single tool needed by professional programmers.

6.0 incorporates our macro assembler, project maintenance tool, overlay linker plus CodePRobe, our powerful full-screen symbolic debugger.

It includes comprehensive database, communications, graphics and screen management libraries. And Lattice C 6.0 is fully ANSI compatible and includes comprehensive documentation.

6.0 provides register variables, in-line functions and keywords that give you greater control over your program's size and performance.

And on top of all this, our new optimizer will make your applications run even faster.

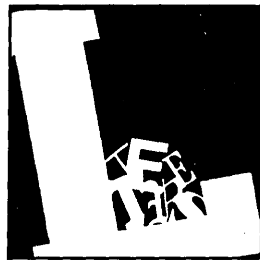
On top of everything, Lattice support is free. Our bulletin board, hotline and BIX network support are the best in the business! *Lattice C 6.0 even has a 30-day money back guarantee!*

If you're reaching for the top, order today. Lattice C 6.0 will bring you to the peak of professional programming.

*Send to Lattice for a free analysis of the benchmarks.

Lattice

Professional Programming Tools Since 1981
2500 S. Highland Ave., Lombard, IL 60148



Letters

More Metaphysics

Reading your recent letters regarding *Micro C's* excursion into the metaphysical realm, I thought I should write and add my vote of support for your editorial experiment. One of the reasons (there are many) I like *Micro C* is that those behind it come through as people, and interesting people at that. Your editorial was part of that. The same applies to the bits of *National Geographic* material another letter writer referred to.

Anyway, those who complained are entitled to their opinions. But I think an editor should be able to write whatever he wants in an editorial. As for those who cancelled their subscriptions, it seems to me pretty silly to give up 94 pages of great technical information just because you don't like the other two pages.

John Wells
467 Fraser St.
Victoria, B.C.
Canada V9A 6H2

Editor's note: Thanks John. After the initial flurry of letters (which ran 50% pro and 50% con), we've received a steady stream of letters, BBS messages, and calls, most of them starting out discussing something else but ending with comments like yours. These are running 99% in favor.

I've noticed some local controversy now that our hospital is teaching "healing touch" therapy to its nurses. It's a healing technique in which (with patient permission) the nurses support the patient's energy field. (Very similar to the Reiki healing techniques I've learned.)

Several people wrote angry letters to the local paper and one even picketed the hospital. But after that initial outburst, the nurses have received solid support from both the local lay community and the medical profession.

Reflections On The Radar Equation

The confusion between the radar equation and the behavior of reflected light on the part of Bob Nansel and Don Sweet is because the radar return consists of scattered radiation. The usual target is irregularly shaped and each surface element reflects the incident radiation in a different direction.

So the target looks like a new source whose brightness is proportional to the inverse square of its distance from the radar. The signal returned to the radar is subject to the same law, so you do indeed multiply the two inverse squares to get the signal strength received per unit cross-section.

The mirror, on the other hand, simply changes the direction of the incident light (if we consider it to be a perfect reflector), and it is proper to use the inverse square of the total distance.

In other words, the proper analog to the radar is obtained by replacing the plane mirror with a polished ball bearing or a matte surface, which I think is what Nansel assumed for his maze runner. The radar equation has nothing to do with mirrors.

No doubt you will get a number of responses on this one, but maybe this will help.

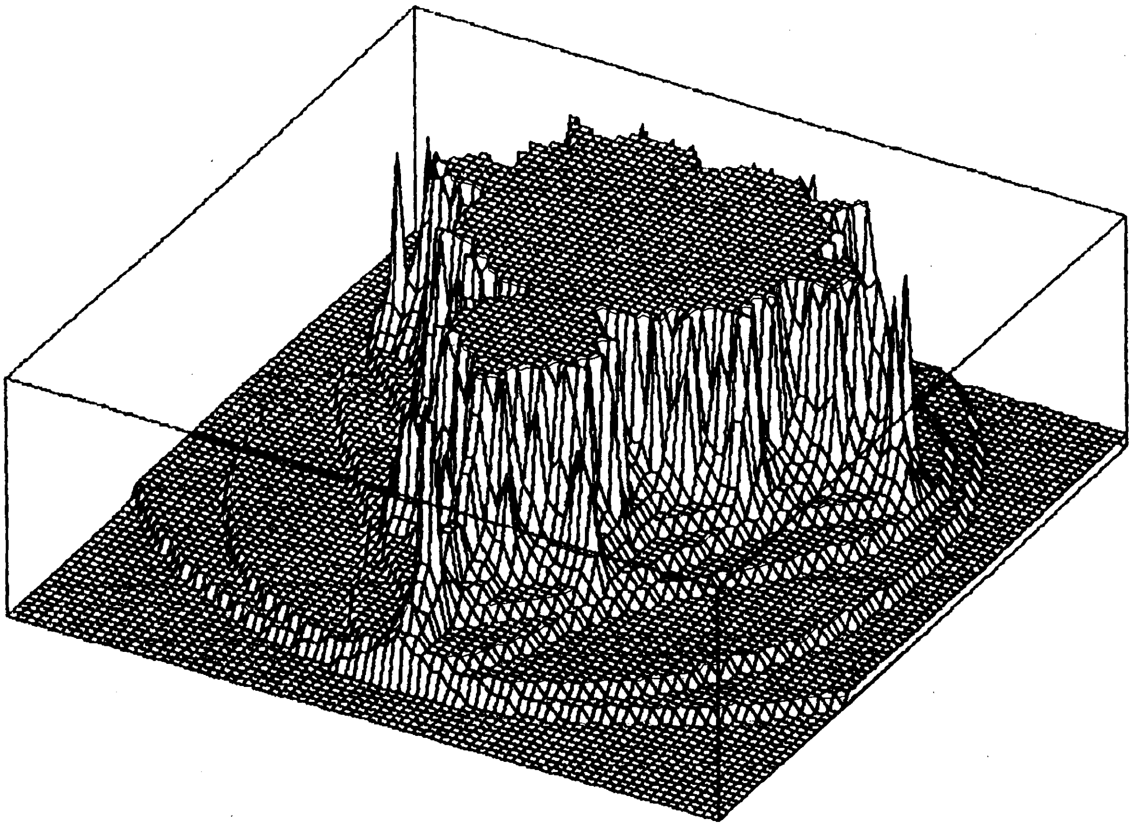
Karl Theobald
1030 Granite Dr.
Granite Shoals, TX 78654

Editor's note: From the flood of responses to this raging controversy, it appears that interest in a topic is proportional to the fourth power of that topic's abstrusity (means it's not perfectly clear to everyone on first glance). Read on for another illuminating view.

Letters continued on page 77

3D-Surface Generation

An In-Depth Look At Graphics, Part 1



Rereading the last 35 issues of Micro C, *you're probably thinking that PCX and Mandelbrot are all you need to know about graphics. Wrong, Buffy. There's something else, and it's three dimensional. (We go for depth.)*

I thought it would be nice if there were a public domain utility that did surface plotting, with hidden line removal. It would let you view the surface from any horizontal angle from 0° through 360° and any elevation angle from -90° to +90°. I couldn't find one, so I wrote my own.

I planned to turn it over to the public domain, so I scrupulously reinvented the wheel at every step to avoid stepping on someone else's toes. My Bresenham's line drawing function turned out to be nowhere near as nice as the one Professor Rasala of Northeastern University wrote in Pascal. So I translated his implementation to C and asked his permission to use it. He was amused that I bothered to ask, since Bresenham's is so standard. But if you think it's easy to do one that's compact, true, and fast, go ahead and write it yourself.

The project turned out to be a bear (actually, more of a female dog). When I got stuck on one thorn (or claw or tooth, to keep the metaphor straight) or another, I amused myself by translating a Pascal high resolution printer graphics module I wrote a few years ago into C. The work dragged on, so I translated another module for scalable, rotatable, justifiable (well, just barely) character string plotting that works with the printer graphics module.

Finally the thing became a true nightmare. I was too far along to quit, and too far gone to continue. I locked myself in my house, along with several pounds of high grade Sumatran coffee, a box of cheap cigars, and more cases of beer than I could count. (I use base 5 to count on my fingers, and with my left hand for the ones, and my right hand for the fives, and a beer tucked in my elbow, I just couldn't get that high.) I sacrificed my Christmas and New Year and my entire right brain to finish the damned thing before my liver failed.

How This Article Happened

Now that I had created it, I had to turn it loose. I sent it to the good folks at Micro C and asked them to distribute it to the public domain for me. Somewhere in the letter (I felt guilty about the inability of the code comments to tell the full story) I offered to write an article. (In my delirium tremins I actually said I would be happy to do so.) They must have smelled the beer on the printout. They took me up on my offer. So, here goes.

What We Have Here

If you've been paying attention, you have a good idea already:

(1) threed.c—a general purpose mod-

By Gregory K. Landheim
123 Greene St.
North Andover, MA 01845

ule for plotting 3-D surfaces with hidden line removal using an axonometric projection, viewable from any horizontal angle from 0° through 360° and elevation angles from -90° through +90°. (See Figure 6.) You can use it, as is, on any plot de-vice that uses left-handed, rectangular, integer coordinates with (0,0) in the upper left corner, and for which you can supply a line drawing function.

(2) grafprt.c—an IBM Graphics Printer (and compatibles, e.g., Epson FX and LX) high-resolution graphics module. It supports both Portrait and Landscape mode drawings, and it draws on 8" by 10⅝" of a page.

(3) grafstr.c—scalable, rotatable, justifiable (left, right and center) string plotting module for use with grafprt.c

(4) arrays.c—contains a function to dynamically allocate 2-dimensional arrays of any type and size (not limited by 64K segments) up to the size of the memory available in the heap. It uses pointers to pointers to any type, but we won't quibble.

Editor's note: Part 2 of Gregory's article (in issue #51) will cover grafprt.c, grafstr.c, and arrays.c)

Why This Article Is So Long

Most graphics articles in technical magazines give you a brief discussion of the math and the method. This is especially true of articles on 3-D surface plotting. They show you how to do it for a special case, then wave their hands and say something like, "Other view angles can be dealt with by appropriate matrix element exchanges and transformations."

This is great if you're a mathematician, but then you wouldn't be reading the article, would you? These authors are perfectly justified by the niggardly attitude of editors who would rather fill their magazines with something useful, like advertising. Fortunately, Micro C's

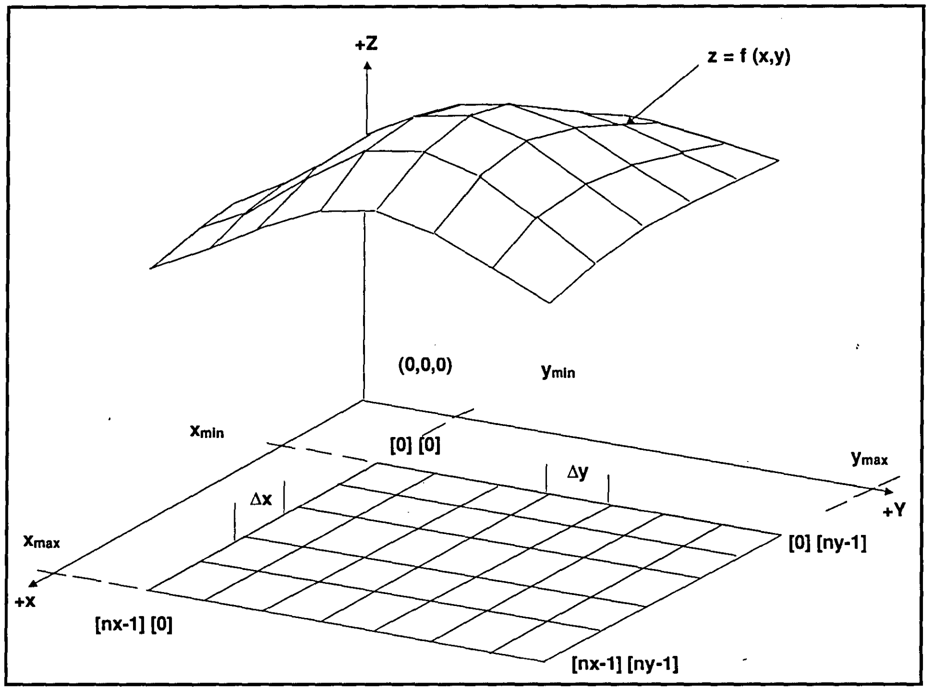


Figure 1—Coordinate Geometry and Matrix Correspondences

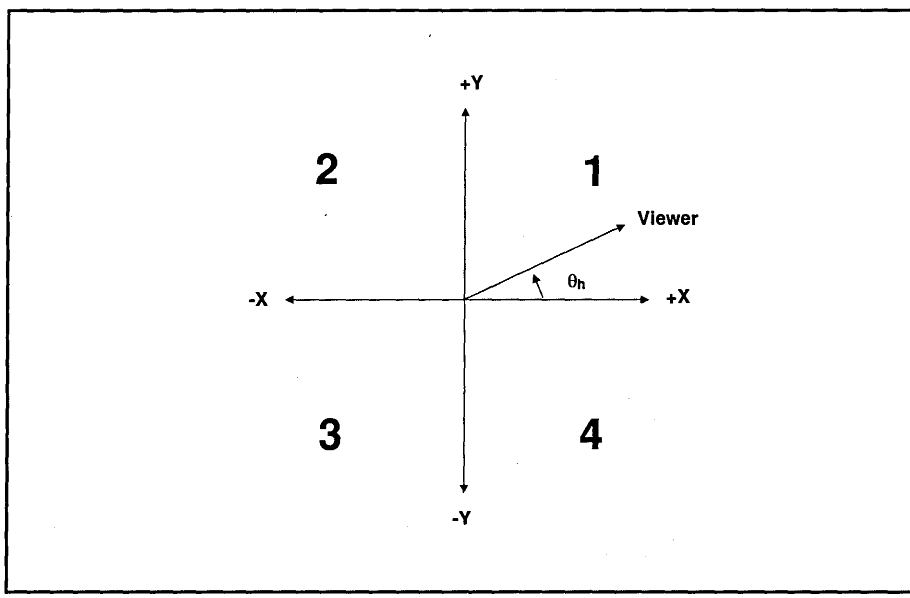


Figure 2—Quadrants and View Angles

golden-hearted editors turn down volumes of advertising to give you in-depth articles and massive amounts of code.

My background is in physics, math, and mechanical engineering, and I have never formally studied computer science. Working with C.S. people, I find a circle of confusion (cold confusion) where they should be keeping coordinate geometry and matrices. So I'll dwell a lot on the basics.

I'll explain the methods behind the 3-D surface plotting module, the printer graphics module, and the associated string drawing module. Nonetheless, I shall attempt to keep things brief without sacrificing substance.

Coordinate Systems—The Basics

Cartesian geometry acquired its name from Rene Descartes (1596-1650), who invented the system. Mathematicians, physicists, and other sensible folk like to use right-handed Cartesian systems, while computer scientists prefer left-handed systems.

I suspect that, somewhere in the history of computer science, the math departments wouldn't loan their coordinate transformation routines to the C.S. departments, so the C.S. people did their own and got it backwards. (In fairness, I admit that most plot devices are intrinsically left-handed.)

In all that follows, I assume all the coordinate systems except the plot device's are right-handed and Cartesian. I also ignore a lot of fine points, but I'm just talking about basics.

A coordinate system can be left- or right-handed. Handedness refers to whether rotation angles are specified as clockwise from a reference axis, or counterclockwise. In right-handed systems, the rotations are counterclockwise. Make a fist with each hand, thumbs pointing at you, and note the direction your fingers curl.

If the reference axes are orthogonal (at right angles to one another), the system is rectangular. If the scales are the same on each axis, the rectangular system is called Cartesian.

Points in space are specified by coordinate pairs (x,y) in two dimensions, and triples (x,y,z) in three dimensions. The coordinate elements can be either positive or negative. A coordinate such as (-4.3,2.7,1.5) means you can locate the point by moving 4.3 units in the negative x direction, then 2.7 units in the positive y direction, then 1.5 units in the positive z direction.

In a 3-D right-handed system, if you point the index finger of your right hand along the positive x-axis, the middle finger along the positive y-axis, and the thumb up; the thumb points along the positive z-axis. Which axis actually points up is arbitrary.

When you learn Cartesian geometry, you start out with a flat surface, the x-y plane, and spend a lot of time learning the rules for 2-D. When extending it to 3-D, the instructor is used to drawing things on the blackboard, so y becomes "up" and z becomes "out" (unless they're computer scientists; then z is "in"). I liked to cut classes and learned with my paper flat on my desk, so I chose to make positive z "up".

Since two lines define a plane, in a 3-D system we have an x-y plane, an x-z plane, and a y-z plane. With positive z "up," the x-y plane is the reference plane. This means that surfaces are specified as z = f(x,y). Figure 1 shows the coordinate system used by the 3-D surface module.

Grids And How To Use Them

Surfaces tend to be smooth, but drawing a smooth surface takes a lot of computational time. So we fake it by computing the surface at grid locations. A rectangular grid is specified by the number of points in the x-direction and the number of points in the y-direction, referred to, respectively, as nx and ny. To make life easy, we make the spacing between grid points constant in a given direction.

The domain over which we plot the surface extends from x_{min} to x_{max} and from y_{min} to y_{max}. Figure 1 shows the relationship between a grid in the x-y

Figure 3—Axonometric Projection

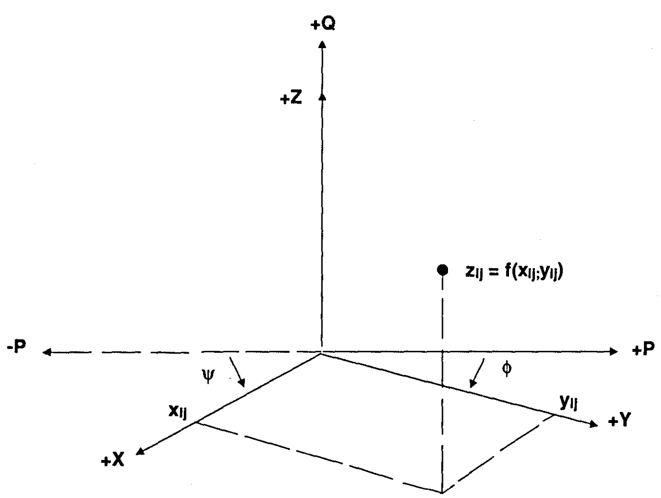
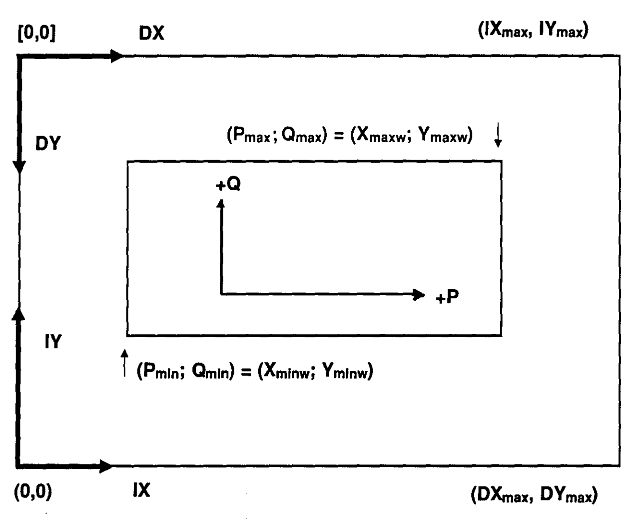


Figure 4—Three Coordinate Systems, CRT Display



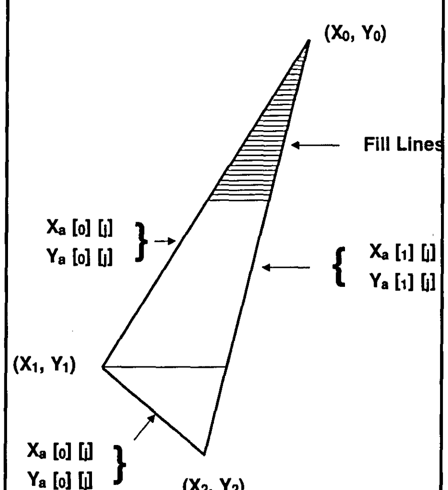
plane and its axial coordinates. The elements in square brackets correspond to matrix subscripts. You could, for example, store all the coordinates that describe the surface in arrays declared:

```
float x[nx][ny],y[nx][ny],z[nx][ny];
```

This quickly eats up storage space. Since we're smart enough to use a Cartesian system, we can completely specify the surface by the values x_{min}, x_{max}, y_{min}, y_{max}, nx, ny, and a 2-D array containing z. For use in surface(), the storage can be accomplished thus:

```
zmin =  1.7e38; zmax = -1.7e38;
deltax = (xmax - xmin) / (nx-1);
deltay = (ymax - ymin) / (ny-1);

x = xmin;
for (i=0; i<nx; i++)
{
  y = ymin;
  for (j=0;j<ny;j++)
  {
    z[i][j] = f(x,y);
    if (z[i][j] < zmin)
      zmin = z[i][j];
    if (z[i][j] > zmax)
      zmax = z[i][j];
    y += deltay;
  }
  x += deltax;
}
```

Figure 5—Filling A Triangle



where f() is an explicitly known function that can compute z for any (x,y) in the plot domain. surface() also requires z_{min} and z_{max}, so we might as well find them when we store the function values at the grid points in z.

The 2-D array z is not actually an array. It must be declared as a pointer to a pointer (e.g., float **z). This lets us use array subscript notation for clarity in the code, but the compiler automatically expands such notation to pointer offsets with a corresponding increase in speed over actual array access. Don't pass an array declared as float z[nx][ny] to surface().

Quadrants And View Angles

In the x-y plane, the coordinate axes divide the plane into four quadrants. By mathematical convention, Figure 2 shows their labeling. Also shown is the relationship between the viewer's eye and the horizontal view angle (θ_h) in the x-y plane.

I depart from the standard mathematical convention of specifying the vertical view angle as the angle rotated downward from the positive z axis. Instead I use the more natural concept of elevation angle (θ_{el}) defined as the angle above the x-y plane at the location of the viewer's eye.

Axonometric Projection

This method of rendering 3-D objects in two dimensions was invented during the dark ages (i.e., before computers). It was quick and easy then, and it's quick and easy now. You lose perspective (especially when you're on your third six-pack and have to close one eye to read code). But it is quicker than a perspective transformation (I think, don't quote me on this), and with surfaces you rarely need depth cues. If you would like a nice perspective transformation, see Reference (3).

When I developed the 3-D module, I began by defining the math transformations with the viewer's eye located in or above quadrant 1. Figures 1 and 3 show views in that quadrant.

Figure 3 shows the transformation from (x,y,z) coordinates to projected (p,q) coordinates. Careful study reveals that for any point (x,y,z):

```
p =  y*cos(φ) - x*cos(ψ)
q = -y*sin(φ) - x*sin(ψ) + z*cos(θel)
```

Traditional (pre-computer) axonometric projections do not consider an elevation angle and do not multiply z by its cosine. I do it to fake a vertical viewpoint.

So where do ψ and φ come from? If you look straight down at the x-y plane ($\theta_{el} = 90°$) as in Figure 2, then:

```
φ = θh
ψ = (90°) - θh
```

But φ and ψ are functions of θ_{el}, and of the viewer's quadrant. It's weasel time now. I would formally like to derive the following relations for you (I did it for myself), but it would take too much space, so I'll sketch an outline.

Intuitively, for $\theta_{el} = 0°$: $\phi = 0°$, and $\psi = 0°$. For other values of θ_{el}, as we rotate θ_h from 0° to 45°, then from 45° to 90°, $\phi = \psi$ at 45° and φ and ψ must reverse their respective values on opposite sides of the dividing line formed by $\theta_h = 45°$.

The resulting transformation, valid in all four quadrants, is:

```
φ' = tan⁻¹( sin(θh) / cos(θh) )
ψ' = (90°) - (φ')

sin(φ) = sin(θel) * sin(φ')
cos(φ) = cos(φ')
sin(ψ) = sin(θel) * sin(ψ')
cos(ψ) = cos(ψ')
```

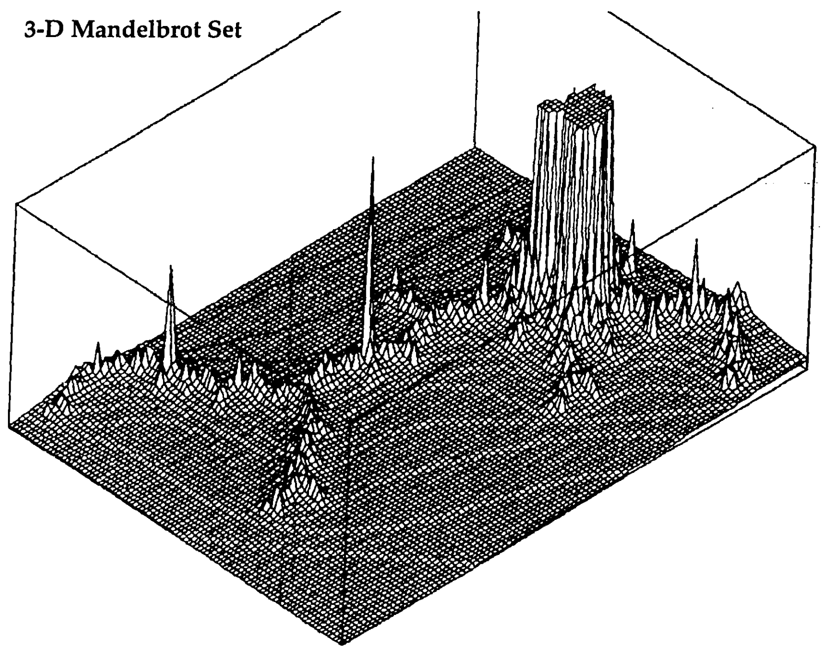
In quadrant 1, of course, $\phi' = \theta_h$. The inverse tangent function is necessary to get the transformation correct in the other three quadrants. transform_angles() performs this work for later use in axonometric().

More Coordinate Transformation—Will It Never End?

Okay. We now know how to get from 3-space to 2-space, but we're still working with real, or floating point, numbers. We want to display the surface in a window, in integer coordinates. For printer graphics, we'll need transformations in both Portrait and Landscape modes; but let's keep it simple by just considering Portrait mode. You use this mode on a CRT display (unless you do your work lying sideways).

We still have three coordinate systems (Figure 4): P-Q, our projected coordinates; IX-IY, a right-handed, rectangular, space; and DX-DY, a left-handed, rectangular space (the plot device's natural coordinate axes). IX-IY and DX-DY are integer spaces where you count by pixels on your CRT display. They map one-to-one onto each other and are non-Cartesian if the aspect ratio does not equal 1.

3-D Mandelbrot Set



You could easily skip the IX-IY space in making these transformations, but I find it a lot easier to decide where to place a window on a screen if (0,0) is in the lower left corner. IX_{max} is the pixel width of the CRT minus 1, and IY_{max} is the pixel height minus 1 (we start counting at (0,0), hence the minus 1).

You can find (p_{min}, q_{min}) and (p_{max}, q_{max}) by projecting the eight corners of a box drawn to completely enclose the surface: (x_{min}, y_{min}, z_{min}), (x_{min}, y_{max}, z_{min}), (x_{max}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{min}), (x_{min}, y_{min}, z_{max}), (x_{min}, y_{max}, z_{max}), (x_{max}, y_{min}, z_{max}), and (x_{max}, y_{max}, z_{max}).

This is done in get_max_min(), determining which of the projected values define (p_{min}, q_{min}) and (p_{max}, q_{max}) and saving the projected coordinates for later use to draw a box around the surface.

In my source code, I never use variables explicitly declared as p or q (corresponding to the description given in this text) preferring to think of them as x and y values, or reusing already declared variables. I define P-Q space in this discussion purely for illustrative purposes. p and q are, however, used as arrays in dosurf() and drawfillrectangle() to store plot device coordinates. This is just a coincidence. Remember: Consistency is the bugaboo of small minds.

You get to specify where the window will go. (x_{minw}, y_{minw}) and (x_{maxw}, y_{maxw}) define this in Figure 4. P-Q coordinates are then transformed to IX-IY coordinates by:

```
ix = xminw + (xmaxw - xminw) * (p - pmin)
     / (pmax - pmin)
iy = yminw + (ymaxw - yminw) * (q - qmin)
     / (qmax - qmin)
```

This is done by defreg() and transf(). "defreg" means "define region." It's really a window definition, but I was afraid of conflicting with someone else's window definition function name if I used "wind" as part of the name.

All you need to get into plot device pixel space (in Portrait mode) is:

```
dx = ix
dy = IXmax - iy
```

itransf() handles this. It also plots in Landscape mode. If you use surface() only in Portrait mode, you can speed things up by eliminating the Landscape transformation and making itransf() into a macro.

The Painter's Algorithm

dosurf() contains the painter's algorithm. It is conceptually simple, but confusing when drawing in all four quadrants. If we all had mainframes, we could use matrix manipulation routines and swap z values around for the proper perspectives. That would keep the loops in

Figure 6—Code for 3D Surface Drawing

```
/* header for threed.c */
int surface(
  float xmin,float xmax,
  float ymin,float ymax,
  float zmin,float zmax,
  int xminw,int xmaxw,int yminw,int ymaxw,
  int hmax,float **z,
  float horangle,float elangle,
  int nx,int ny,int box,
  int fillcolor,int edgecolor,int boxedgecol,
  void far csetfunc(int color),
  void far linefunc(int x1,int y1,int x2,int y2));

/* start of threed.c */
#include <stdlib.h>
#include <math.h>
#include "threed.h"
#include "grafprt.h"
#include "arrays.h"

#define PI 3.141592653589793238
#define BIGNUM 1.7e38

/* module threed.c, turbo c v 2.0, large code model.

released to the public domain by Gregory K. Landheim on
January 1, 1989, (c) all rights reserved. anybody is
authorized to use this code for any purpose whatsoever
on the condition that they realize I assume absolutely
no liability and give no warrantee for its use or
performance. it is distributed "as is." */

/*--------function prototypes-----------*/

void degrees_to_rads(float *horangle,float *elangle);
int get_quadrant(float horangle);
void transform_angles(float horangle,float elangle,
                      int quadrant);
void get_max_min(float xmin,float xmax,float ymin,
                 float ymax,float zmin,float zmax,
                 float *yminp,float *ymaxp,
                 float *zminp,float *zmaxp);
void axonometric(float x,float *y,float *z);
void dosurf(float xmin,float xmax,float ymin,
            float ymax,int nx,int ny,float **z,int i0,
            int i1,int inci,int j0,int j1,int incj,
            int incmode,int fillcolor,int edgecolor);
void drawfillquadrangle(int p[2][2],int q[2][2],
                        int fillcolor,int edgecolor);
void filltriangle(int x1,int y1,int x2,int y2,
                  int x3,int y3,int fillcolor);
void swapcoords(int *x1,int* y1,int* x2,int* y2);
void transformbox(void);
void drawboxbottom(int boxedgecolor);
void drawboxback(int quadrant,int boxedgecolor);
void drawboxtop(int boxedgecolor);
void drawboxfront(int quadrant,int boxedgecolor);

/*--------global declarations----------*/

int **Xa;                             /* storage for */
int **Ya;               /* filltriangle() edge lines  */

float Cosphi,Sinphi,Cospsi,Sinpsi,Cosel; /* constants
                         for axonometric projection */
float Yb[5],Zb[5],Yt[5],Zt[5]; /* arrays for box
      bottoms and tops in projected real coordinates */
int Pb[5],Qb[5],Pt[5],Qt[5]; /* arrays for box bottoms
        and tops in transformed integer coordinates */
void far (*Linef)(int x1,int y1,int x2,int y2);
     /* address of plot device line drawing function */
void far (*Csetf)(int color); /* address of plot
                             device set color func */

/* surface () - general-purpose surface plotting
routine to plot a surface described as a matrix of
gridded z-values in the xy domain.

int surface(
  float xmin,float xmax,float ymin,float ymax,
  float zmin,float zmax,
  int xminw,int xmaxw,int yminw,int ymaxw,
  int hmax,float **z,
  float horangle,float elangle,
  int nx,int ny,int box,
  int fillcolor,int edgecolor,int boxedgecol,
  void far csetfunc(int color),
  void far linefunc(int x1,int y1,int x2,int y2))
{
  extern int Maxheight;   /* declared in mod grafprt */
  float temp;
  int quadrant,bufsiz,incmode,invert;
  float ymaxp,yminp,zmaxp,zminp; /* max and mins of
                                 projected surface */

  if (abs(elangle) > 90.0) return(1);
  (elangle < 0.0) ? (invert = 1) : (invert = 0);

  /* allocate memory for the arrays:                  */
  bufsiz = ymaxw - yminw;
  if ((xmaxw - xminw) > bufsiz) bufsiz = xmaxw - xminw;
  Xa = (int **) alloc_2d_array(2,bufsiz,sizeof(int *),
                                  sizeof(int));
  if (!Xa) return(2);
  Ya = (int **) alloc_2d_array(2,bufsiz,sizeof(int *),
                                  sizeof(int));
  if (!Ya) return(3);
  /* change the angles from degrees to radians:       */
  degrees_to_rads(&horangle,&elangle);
  /* determine the viewing quadrant:                  */
  quadrant = get_quadrant(horangle);
  /* determine the global projection angles:          */
  transform_angles(horangle,elangle,quadrant);
  /* swap xmin/xmax and ymin/ymax in quads 2 and 3:   */
  switch (quadrant) {
      case 1 : break;
      case 2 :
      case 3 : temp = xmin; xmin = xmax; xmax = temp;
               temp = ymin; ymin = ymax; ymax = temp;
               break;
      case 4 : break;
   }


  /* find max and min of the real coordinate domain: */
  get_max_min(xmin,xmax,ymin,ymax,zmin,zmax,
              &yminp,&ymaxp,&zminp,&zmaxp);
  /*set max pixel height of device and def plot region*/
  Maxheight = hmax;
  defreg(yminp,ymaxp,zminp,zmaxp,
        xminw,xmaxw,yminw,ymaxw);
  /* assign color setting and line drawing function
     addresses to global variables: */
  Csetf = csetfunc;
  Linef = linefunc;
  /* put an optional box about the surface: */
  if (box) {
      transformbox();
      invert ? drawboxtop(boxedgecol) :
        drawboxbottom(boxedgecol);
      drawboxback(quadrant,boxedgecol);
   }
  /* plot the surface as a function of quadrant.
     make certain it draws from back to front: */
  switch (quadrant) {
      case 1 : (horangle > PI/4.0) ? (incmode = 0) :
                 (incmode = 1);
               dosurf(xmin,xmax,ymin,ymax,nx,ny,z,0,
                 nx-1,1, 0,ny-1,1,incmode,fillcolor,
                 edgecolor);
               break;
      case 2 : (horangle > 3.0*PI/4.0) ? (incmode = 1):
                 (incmode = 0);
               dosurf(xmin,xmax,ymin,ymax,nx,ny,z,nx-1,
                 0,-1,0,ny-1,1,incmode,fillcolor,
                 edgecolor);
               break;
      case 3 : (horangle > 5.0*PI/4.0) ? (incmode = 0):
                 (incmode = 1);
               dosurf(xmin,xmax,ymin,ymax,nx,ny,z,nx-1,
```

Continued on page 87

dosurf() simple, but it adds a lot of overhead. So we end up with a function that gives me a headache.

The painter's algorithm simply draws the projected quadrangles that form the gridded surface from back to front, filling (or painting) the quadrangle interiors with a background color as each is drawn, to cover up the quadrangle edge lines behind the current quadrangle.

Quadrant 1 serves as the basis of the whole thing. For θ_h less than 45°, you draw each quadrangle in the column nearest the y-axis (Figure 1), then move one column out in the direction of the positive x-axis and draw that column. Repeat for all columns.

For θ_h greater than 45°, you draw each quadrangle in the row nearest the x-axis, then move one row out in the direction of the positive y-axis and draw that row. And so on to the end.

Argument "incmode" of dosurf() controls whether to increment along the x- or y-axis. In either mode, an initial for loop over k and l projects, transforms to device coordinates, and fills the first quadrangle in each column (or row). After that, we only need to project and transform two new points for each quadrangle, because the other two are saved from the previous quadrangle. The quadrangle is drawn and filled by a call to drawfillrectangle(). (I know, I should have called it drawfillquadrangle(), but I wasn't thinking straight at the time.)

Drawing And Filling

I don't see any way to fill a quadrangle without breaking it up into two triangles. This is what drawfillrectangle() does. One vertex of each triangle is the one farthest from the viewer, another is the one closest to the viewer. After the triangles are filled, the quadrangle edge lines are drawn.

filltriangle() does the dirty work. It first sorts the vertices of the triangle in ascending vertical order (in device coordinates).

In order to use loops instead of if statements, I pretend the long side of the triangle consists of two lines, one of which has zero length. This trades off storage space for speed.

I used a variation of Bresenham's line-drawing algorithm to compute the coordinates of the triangle edge lines and store them in global arrays Xa and Ya. The computation is performed by dividing the triangle into two conceptual triangles with an imaginary horizontal line at the mid vertex (Figure 5). The outer loop controls whether the computations are for the upper or lower triangle. The inner loop computes the actual edge lines coordinates.

A final loop fills the triangle by drawing blank horizontal lines from (Xa[0][j],Ya[0][j]) to (Xa[1][j],Ya[1][j]). The j subscript indicates the j^{th} pixel working from the top to the bottom of the triangle. Ya[1][j] equals Ya[0][j] for all j.

To make threed.c as general as possible, I use the same line drawing function for horizontal lines as for all the other lines. However if you have a high speed horizontal line drawer, use it.

References

(1) *Advanced C Tips and Techniques*, Paul Anderson and Gail Anderson, Howard W. Sams & Co., 1988.

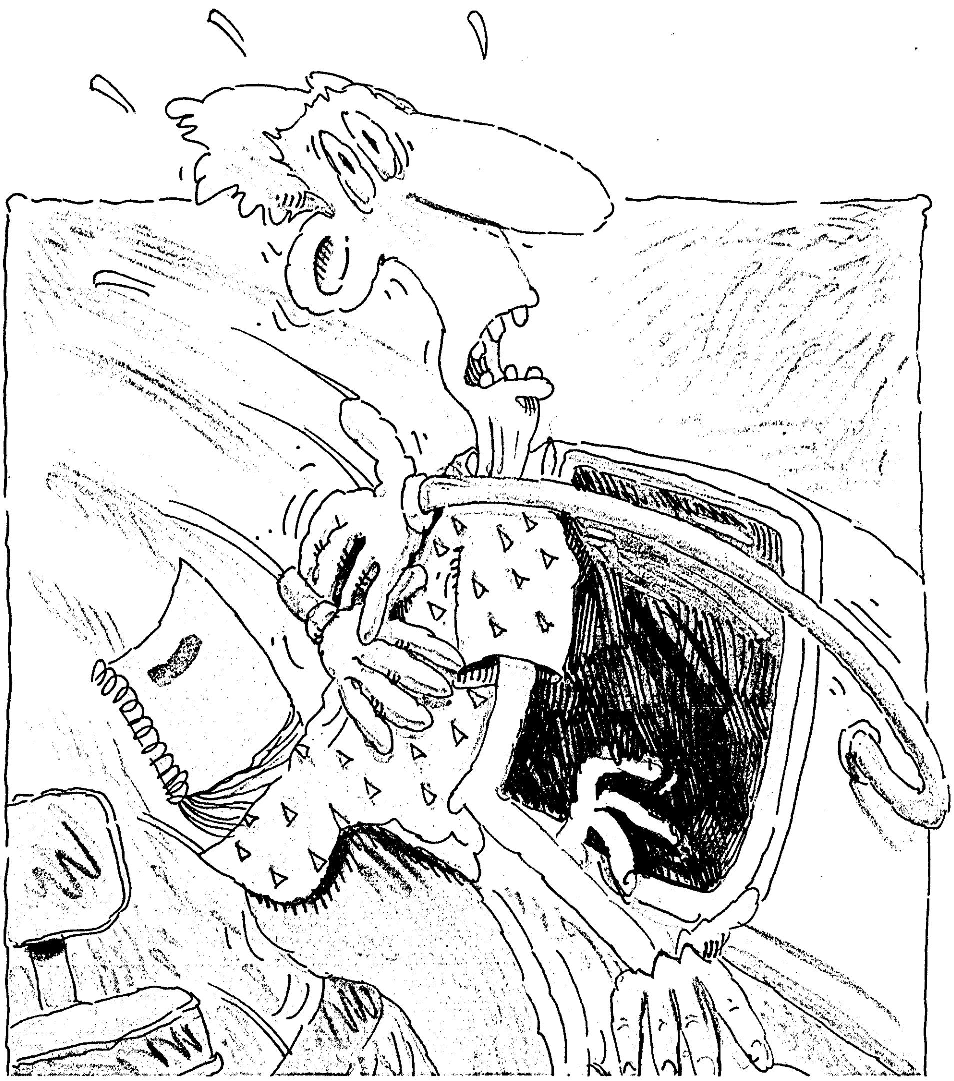
(2)"High-Resolution Printer Graphics," Mark Bridger and Mark Goresky, *BYTE*, Nov 1985, pp. 219-232.

(3) "The Painter's Algorithm," Richard Chandler and Gary Faulkner, *PC Tech Journal*, November, 1985, pp. 181-187.

♦ ♦ ♦

The PC Video Frame Grabber

Or, In Search Of The Lost A/D Converter



Desktop graphics are quite possible. But possible and easy are two different things. Here's a video frame grabber that might just shortcut the process. This is yet another way to use an A/D convertor.

There I was trying to come up with a catalog for IDEC's line of PC clone computers using one of our AT compatibles, Ventura Publisher, and our laser printer. The idea was to do as nice a job as possible on this document so that our customers could see what a desktop publishing system could do for their efforts. I wanted to let our customers *see* one of our systems. *See* it in print. Credibility. Yeah, that's the ticket.

How do I get a picture of our computer into our computer? My associate, Dr. Rao, tripping over one of the many extension cords strung out on the floor, suggested we needed a scanner to make the idea work. He was carrying a box of hard drives. "They're about a thousand dollars," he said. I wondered if he was referring to the scanners or the hard drives. Turned out it was the scanners.

Then I looked at what I had to do: take a black and white picture of the subject; develop and print the picture; copy the picture on a copy machine to reduce glare; scan the picture; suck the result into Ventura; and then print it on the laser printer. Sounded like a lot of work.

So I proceeded to draw pictures of computers and disk drives for our catalog using a paint program and a mouse, pixel by pixel. That *was* a lot of work. "There's gotta be a better way," I grumbled.

There is. The obvious solution is to hook a video camera into the computer. We immediately confiscated our little VHS/C camcorder for the project. Sorry, no more baby pictures! Using video added the dimension of being able to take pictures from tape. The situation was improving by the minute.

What Would It Take?

The two happiest days in a sailor's life are the day he buys his boat and the day he sells his boat. When it comes to engineering projects, the two happiest days are the day you design the hardware to do the job and the day you think you've fixed the last bug in the software. Some projects never have the final bug fix, some have it a great many times. This project seems to be of the latter variety.

Having decided to build the World's Greatest Low Cost Video Digitizer, there remained but a single question: How to do it? We needed some kind of A/D converter. What speed? It was going to have to put its data somewhere. PC memory? Dedicated memory?

The Specification

We wanted to digitize standard RS-170 video. Cameras, camcorders, and VCRs output this on their video output connectors. And, we wanted to do this in real time.

That's a computer buzz word if there ever was one. What is real time? Real time and beauty must be very much alike

By Gene Toner
IDEC, Inc.
P.O. Box 69
Fountainville, PA 18923
(215) 249-0673

because they both depend on the beholder. Real time in the RS-170 video world is about 1/60 of a second. This is the time it takes to paint one field of video. The rewards for being able to digitize one field of video in real time are definitely worth the effort.

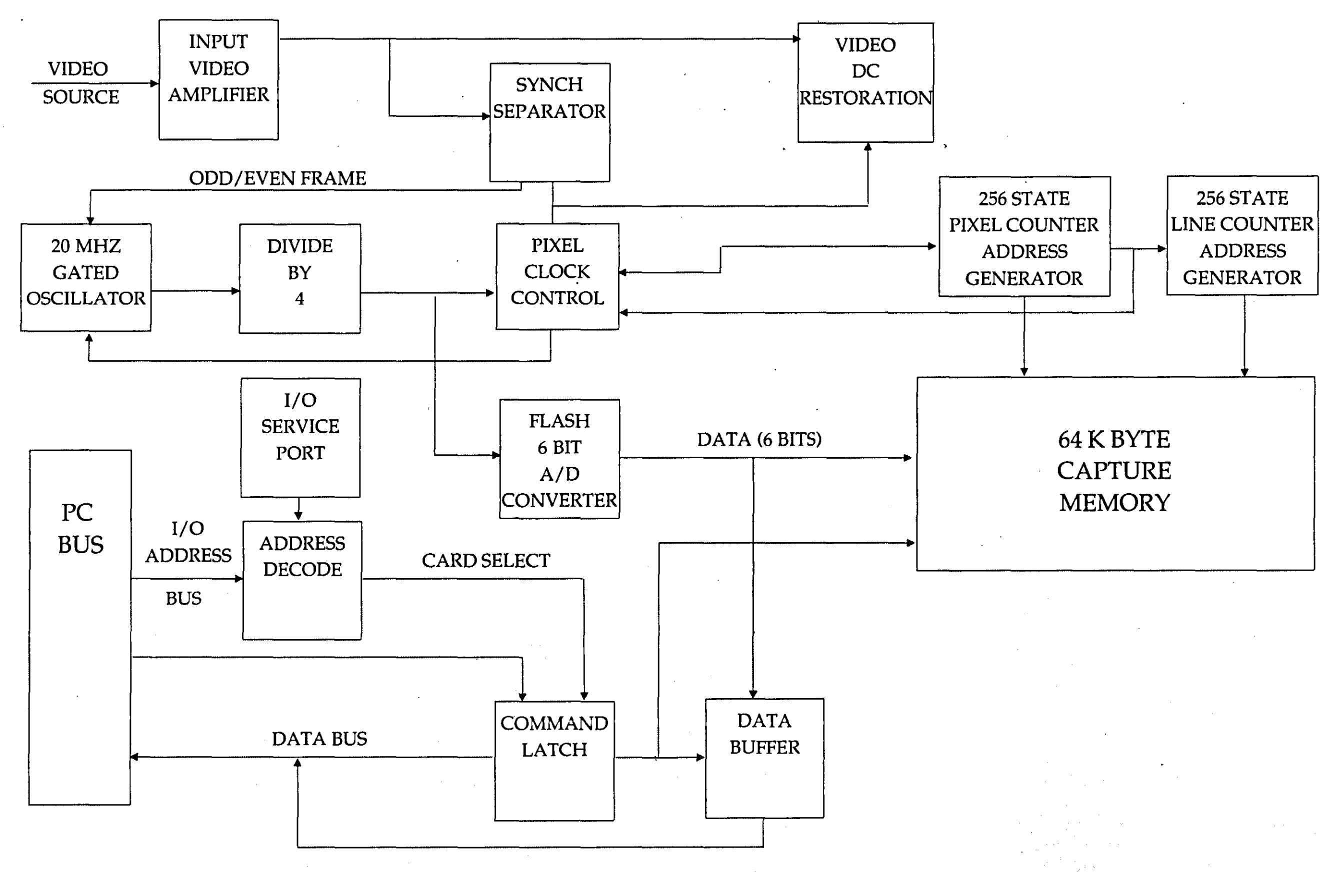
If one can digitize "on the fly," a "snapshot" can be extracted from any video source. Without this capability, the subject of the picture must remain still, or the video source frozen. Some cameras and VCRs do a credible job of this, but we (and perhaps you) don't have this feature on our video gear.

If one can digitize "on the fly," a "snapshot" can be extracted from any video source.

So we want to digitize an entire field in real time. Where do we put it? RAM is an attractive place. We could use one of the system's DMA channels for depositing the result of the A/D conversion directly into memory.

How many pixels? How many shades of grey? Well, the best PC video adapters around these days are VGA. They have a grey scale mode which allows 320 x 200 resolution with 64 shades of grey.

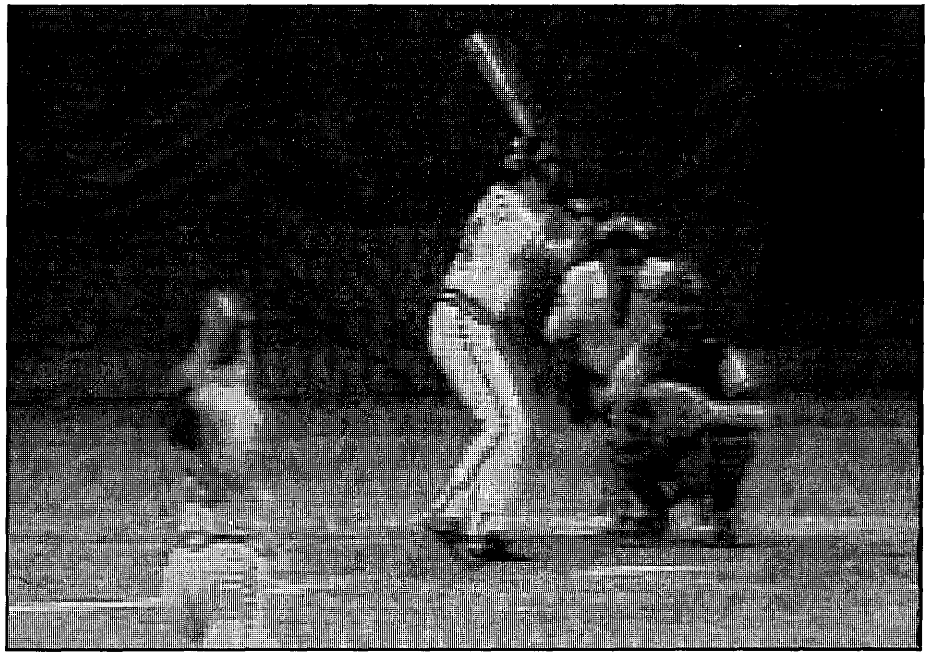
Figure 1—Grabber Block Diagram



Sounds like a good place to start, especially since a field of RS-170 video has only about 244 lines of vertical resolution available. (Actually 262½ lines, but some of these get consumed by vertical retrace.) Because we work in a digital world, the number 256 is much more appealing than 320 or 244. So we went with 256 x 256 x 64 shades of grey.

In a standard video signal, every horizontal line takes roughly 63 µseconds. Of that 63 µsec, video takes up about 53 µsec, with the remaining time devoted to horizontal synchronizing information. After the dust settles over the calculator, the result requires a sampling rate of 5 MHz to slice that 53 µsec up into 256 samples.

Another result is that with a normal PC-XT (4.77 MHz) computer, you wouldn't have enough bandwidth available to stuff all these samples into computer memory using the machine's DMA channel. You would have to use dedicated memory.



Real-time Grab. Fast!

The Lost A/D Converter

Figure 1 shows the frame grabber (actually a field grabber, each frame made up of an odd and even field) in block diagram form. Figure 2 is the schematic of the final version of the grabber. The heart of the system is the RCA3306 6-bit flash A/D converter.

This, however, was not always the case. There are other offerings in the flash A/D race, including a very nice 8-bit Samsung part. It, along with the A/D converter, has some very nice signal clamping circuits.

Samsung advertised the part heavily and, upon our request, sampled us a few parts, with the assurance that unlimited quantities would be available when we needed them. So we prototyped a system using this part.

Then came the worldwide DRAM shortage, and Samsung decided to build DRAMS instead of flash A/D converters. Can't imagine why, but they left us with a very functional video capture board and no way to get parts.

We then made a mad search to find a replacement A/D converter, preferably *American*, with at least one second source. What we found was the RCA 3306 6-bit converter chip, second sourced by Micro Power Systems. We were in business once again.

Not often does a designer have the opportunity to go back and rethink all the design tradeoffs he made during the design, but it happened here. Because of this, we were able to squeeze more performance out of the capture card.

The System

We brought the signal from the camera into the capture card as standard RS-170 video, which has a 1 V peak to peak (black to white) amplitude.

The input video amp (IC1) serves two purposes. (See Figure 1.) As a differential amplifier, it not only provides signal gain but also noise reduction. This output goes to two other modules, the sync separator (IC2), and the DC restoration block (IC3 and IC4).

The National LM1881 handles sync separation. Fed a standard video signal, it produces composite, vertical, odd/even frame and burst gate/black level timing information.

We use the odd/even signal to capture even frames (those with a complete line of video on the first line of a frame). We use the black level signal to sample the incoming video for its reference black level. The DC restoration block is the LM 398 sample and hold amplifier which holds and filters the video black level for later subtraction from the video signal.

After subtracting the detected black level from the incoming video, we have a ground referenced video signal with black at ground and white at roughly 3.3 volts—perfect for conversion by the RCA 3306 flash converter (IC13).

A gated crystal oscillator operating at 20 MHz and divided by 4 controls the conversion process. Each rising edge of this sample (pixel) clock causes the A/D to take another look at the video signal. It then shifts the previous sample to its output.

The Conversion Process

The PC bus address decoder (IC17 and IC18) decodes a write to the board, and a command to capture a video frame loads into the command latch (IC19). This command places the capture memory in a writeable condition and allows the capture of the next even field. When the sync separator senses the next even field, it allows the gated oscillator to operate for 256 samples of the pixel clock as counted by the pixel counters (IC9 and IC10).

When it reaches the count of 256, the pixel clock shuts off until the next horizontal sync pulse shows up. (Line counters, IC11 and IC12, also get incremented here.) These counters generate the addresses for the video capture memories. The falling edge of the pixel clock strobes in the data.

The process continues until the carry bit of the last line counter resets the command latch and stops the process. The computer monitors the command latch to determine the completion of the sample.

At the end of the process, video memories return to a readable condition. The computer resets the pixel counters and the line counters. Then, once a signal from the command latch has replaced the pixel clock, the computer can access video memory.

When the computer forces this signal line high and then low, the pixel counters and the line counters are incremented. This way the computer can read the contents of video memory and transfer those contents into its own memory.

The Software

Once the data resides in main memory, the software takes over. The program was written using Microsoft C 5.1 and the Zortech libraries. We found the Zortech windows libraries not only well done, but also a bargain at $50.

Everybody has their favorite C compiler, and ours is Microsoft's. Its wide, third-party support is one of the reasons, CodeView another. The intent was to produce reasonable pictures on the laser printer, and we accomplished this using a technique called dot dithering. This technique forms a macro-pixel made up of several dots and simulates grey scale by the percentage of dots printed. Here, we chose a four dot by four dot cell to simulate 16 grey levels. (This doesn't seem to be too grainy.)

We can display the video on all the standard monitors. The most impressive is the VGA, presenting the picture in its full 64 shades of grey.

We can also reproduce this picture on an IBM graphics compatible printer. Here, we use each printer dot as a cell and strike it up to four times to simulate four levels of grey.

Finally We Get To The Desktop

Recent investigation showed that both Ventura Publisher ver. 2.0 and Aldus PageMaker ver. 3.0 support grey scale images in the TIFF file format. Discussions with the technical support group at Aldus yielded a TIFF developers software kit for the meager price of $30.

After tearing into the developers kit, we discovered that the TIFF file format is well thought out and easy to use. We wrote a file format converter to take our format (IDC) and convert it to TIFF.

Once we had conquered TIFF, Ventura 2.0 accepted our image files. We now take video photographs and incorporate them into desktop publishing. Thanks to TIFF, we can also print the resulting documents on the printer while maintaining grey scale information.

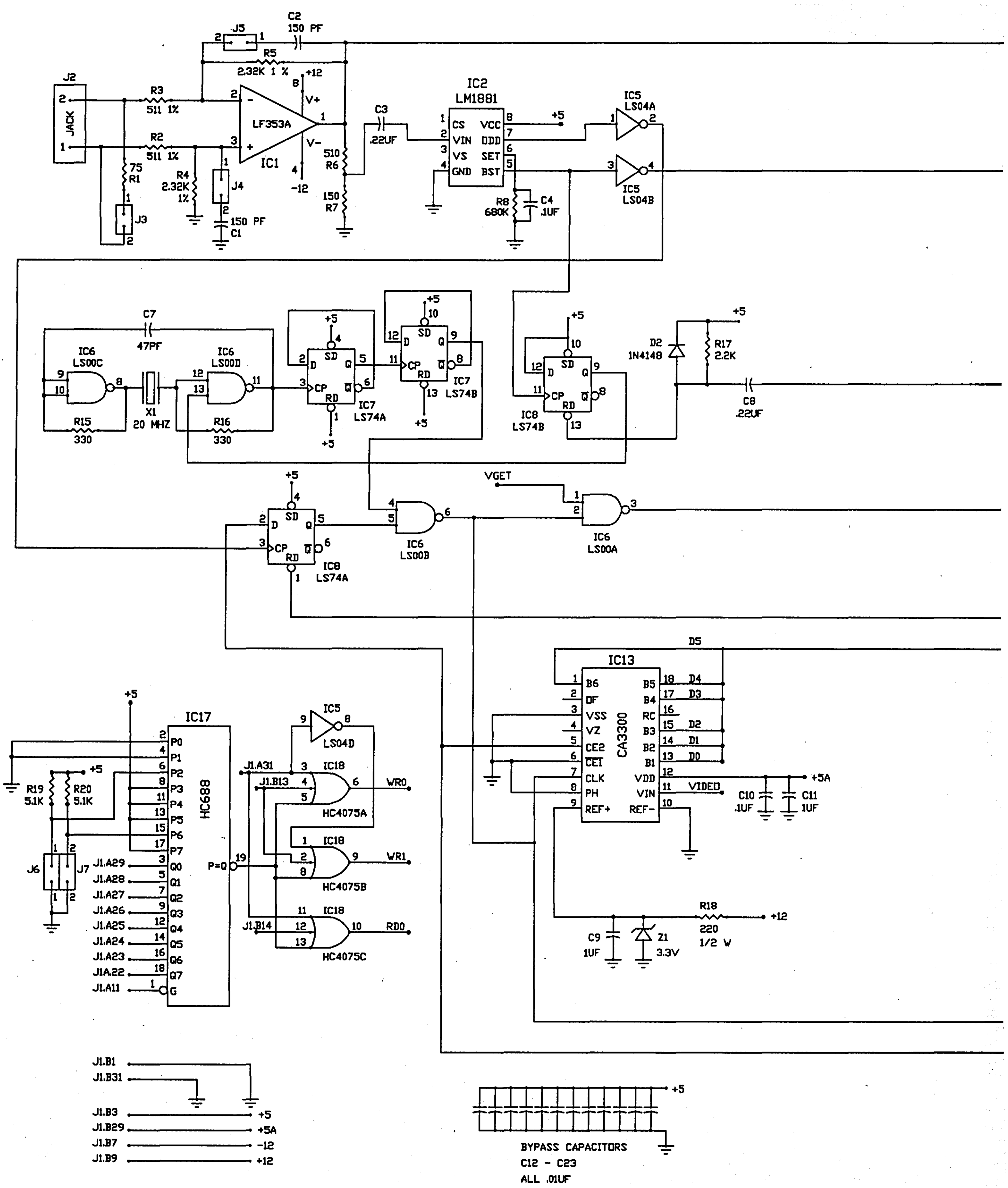
Quality has been very good. With the VGA and its 64 shades of grey, an image direct from the camera (tape tends to degrade the image slightly) comes very close to black and white broadcast quality.

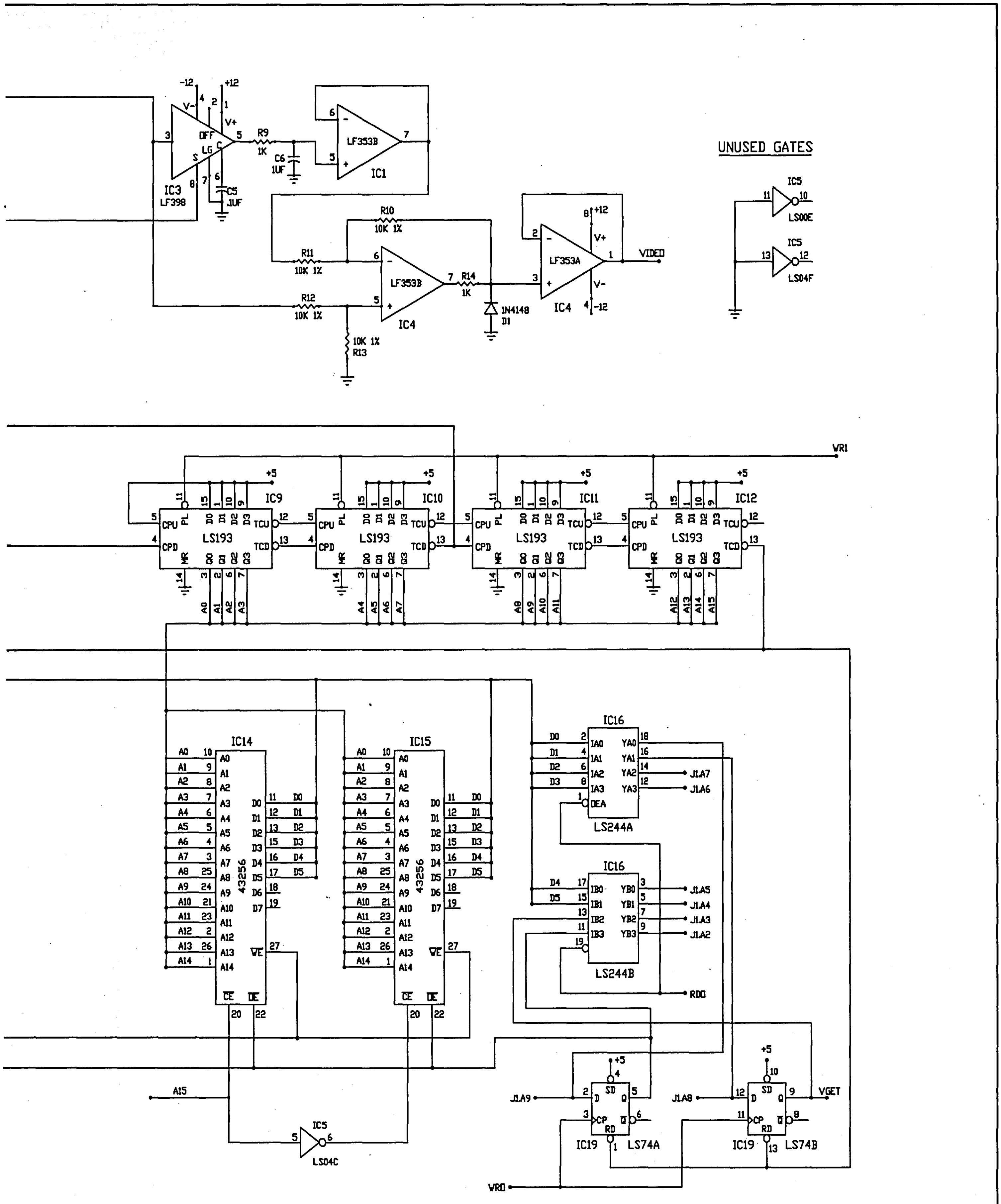
Editor's note: Idec's Supervision software and a sample captured screen are available on the Micro C BBS and the Issue #50 listings disk.

They've offered Micro C *readers a special price for the Grabber Board and software ($175 + s/h). Kits are also available. Call Idec for details.*

♦ ♦ ♦

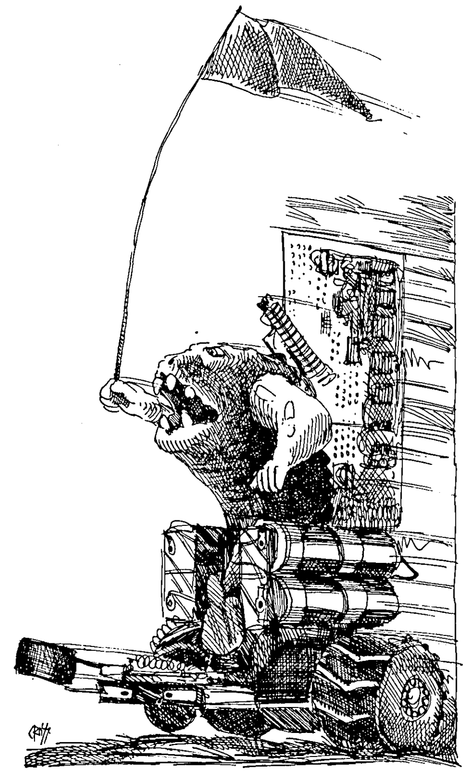
Figure 2—Grabber Schematic




UNUSED GATES
IC3
LF398
IC1
LF353B
IC4
LF353A
LF353B
R9
1K
C6
1UF
C5
.1UF
R10
10K 1%
R11
10K 1%
R12
10K 1%
R13
10K 1%
R14
1K
1N4148
D1
VIDEO
+12
-12
IC5
LS00E
LS04F
WR1
+5
IC9
IC10
IC11
IC12
LS193
CPU
CPD
TCU
TCD
PL
MR
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
IC14
IC15
43256
IC16
LS244A
LS244B
J1A7
J1A6
J1A5
J1A4
J1A3
J1A2
RD0
J1A9
J1A8
IC19
LS74A
LS74B
VGET
WR0
LS04C

LIMBO Part Three

Building The Mobility Base



It's time to get out the bailing wire and tin snips. This time Bob builds LIMBO's action fraction.

Mazebots get stuck in many ways, usually by running headlong into something.

A maze robot ain't much fun if it can't cruise through the maze without getting stuck. Mazebots get stuck in many ways, usually by running headlong into something. We can usually blame inadequate sensor coverage. Even a single one-inch gap in the bumper perimeter will be catastrophic, eventually.

Of course, there are other ways to get stuck, losing traction being a popular one. The concrete surface on which members of the Seattle Robotics Society run maze robots looks flat, but the robots know better. Mazebots with two drive wheels and two castors tend to wobble like four-legged stools in our maze. To maintain traction, the wheels need to conform to the irregularities of the running surface. That requires some kind of springy suspension.

Why not make a robot with a rigid three-point suspension instead of four-point? Small bumps and hollows wouldn't bother a three-wheeled robot because, like a three-legged stool, each wheel makes contact with the ground.

However, I find that three-wheelers are more tipsy than four-wheelers. Also, four-wheelers can have more traction because the weight can be placed right over the drive wheels, an impossibility with three-wheel designs. (Perhaps the ideal system would be a gyro stabilized two-wheeler, or even a unicycle. Imagine: a robot able to do a high wire act....)

LIMBO is round as viewed from above, with a four-point spring suspension. The drive wheel/motor combinations mount rigidly to the chassis, while the front and back castors are spring mounted. The drive motors mount inside an aluminum box.

I call the motor box, combined with the castors and spring, the undercarriage. The undercarriage bolts on underneath the bumper contact skirt. This whole assembly is known in robotics slang as a mobility base, or just base.

A superstructure, which carries the batteries, sensors, and control electronics, bolts on top of the base. Wires from the undercarriage run up to the superstructure through a central wire access hole. This time we'll build the LIMBO mobility base.

Preparing The Motors

Tools: Hacksaw, drillpress, Vee-blocks, clamps, hand reamer, metal files, 200 W soldering gun, 6″ machinist metal rule, flux brush, rubber mallet, drills.

Materials: Nesting brass tubing (9/32″ o.d. to 3/8″ o.d.), IMC stepper motors, rosin-core solder, rosin flux paste, cotter pins.

Step 1. Make drive shaft sleeve adaptor.

The IMC motor I chose has 1/4″ shafts on each end, one 0.7″ long with flats, the other 1.55″ long, no flats. The motor body has a 2.25″ square mounting flange with one ear cut off and standard 3/16″ mounting holes in the remaining three.

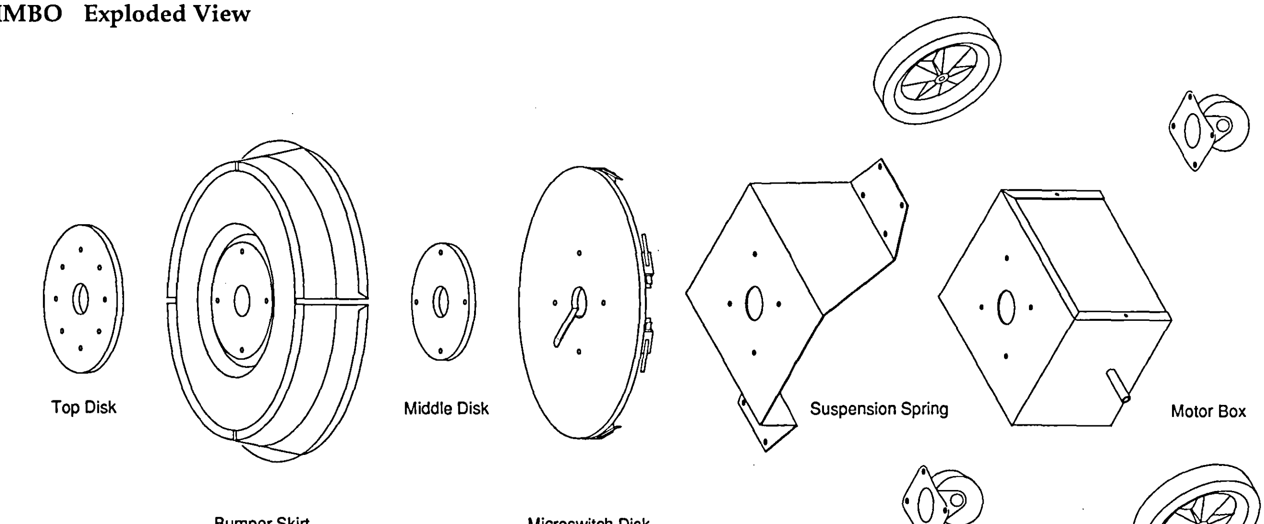
The long shaft is just right for mounting the wheel. Unfortunately, it's on the opposite end from the mounting flange, so you'll need spacers to mount the motor inside its box. Also, you need to add a sleeve to the 1/4″ shaft so it'll fit into the 3/8″ hole on the wheel.

You can accomplish this with nested brass tubing, available at any hobby shop. You need four sizes from 9/32″ to 3/8″ o.d. with 1/32″ wall thickness. It's best to take the motor and wheel you plan to use with you to make sure the sizes are right.

These brass tubes usually come in 12″ lengths, more than sufficient for two motors. Get brass tubing, not aluminum, be-

By Bob Nansel
816 North 105th St.
Seattle, WA 98133

LIMBO Exploded View



cause aluminum tubes are too soft and are very difficult to solder.

First, nest all the tubes together. They should be very nearly all the same length. If not, sand or file the tube ends (still nested) until they are. Next, telescope the tubes out on one end so that about ¼″ of each tube shows. Then brush soldering flux paste on these exposed ends (but not on the inside of the smallest tube, nor on the outside of the largest tube). Slide the tubes back together, repeat this procedure on the other end and then slide back flush.

Using minimum pressure, clamp the nested tubes in a bench vise with one end of the tubes perpendicular to the bench top. Heat up the top end with a soldering gun and sweat a small amount of solder into the flush ends. Be careful not to get solder on the inside surface of the smallest tube. Allow the tubes to cool, then flip the tubes over to solder the other end. The idea is to make the tubes a single unit for cutting and drilling.

From each soldered end, measure 1.5″. Use a hacksaw to cut a piece from each end slightly longer than the marked length. Sand or file these to 1.5″. Remove the burr inside the cut ends with a hand reamer. Ream away any excess solder on the other end, too.

Clean off any flux or brass particles with a paper towel, then slide the completed sleeves onto the stepper motor shafts. They should slide on easily. If they don't, you will either have to ream the ends some more or squeeze the tubes back to round.

You can avoid most of the work above if you can find tubing with ¼″ i.d. and ⅜″ o.d. (I couldn't find any). In this case, just cut two pieces to length, file, ream and you're done.

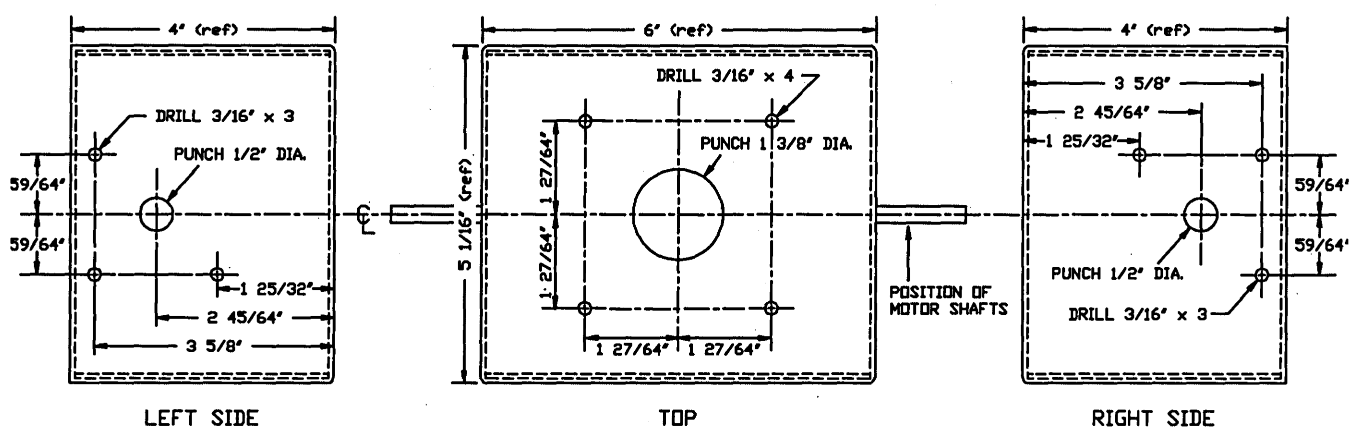
Step 2. Drill cotter pin hole.

The long shafts of the steppers don't have flats ground in them, so you can't use setscrews to secure the adaptor sleeves and wheels to the shafts. I don't trust setscrews for high torque drives anyway (they tend to come loose midway through your best run), so I use cotter pins for really positive drive. Our problem now is how to drill cotter pin holes precisely through the ¼″ hardened steel shafts of the steppers and to perfectly match holes through the sleeve adaptors.

The problem breaks down into how to hold the work-pieces and how to assure proper alignment. Holding the work-pieces is best done with machinist type Vee-blocks and clamps. Alignment will be perfect if the holes are drilled through both the sleeves and the shafts simultaneously (see photo).

Mark a line around the sleeve adaptors 7⁄32″ from the soldered ends, then slide them onto the long shafts, ends flush with the ends of the shafts. Use two Vee-blocks to support both front and back shafts. Slide scrap tubing left over

Motor Box Layout



from Step 1 onto the front (short) shaft, so the two Vee-blocks support the motor evenly between them.

Clamp the Vee-blocks to the drillpress table using C-clamps or bar clamps. Don't tighten the clamps completely yet; leave them loose enough that you can adjust the work position by tapping with a rubber mallet. Chuck up a 3/32" bit in the drillpress. Then, with the drill press still turned off, bring the bit gently down to touch the sleeve adaptor. Adjust the Vee-blocks until the bit comes down squarely on the 7/32" mark.

An old machinist trick to tell if the bit is perpendicular to the round shaft is to put a 6" metal rule on the shaft, then bring the bit down (power still off!) with enough pressure to hold the rule in place. Adjust the blocks around until the rule is perfectly level and perpendicular to the drill bit.

This will ensure that the hole is drilled through the diameter. Remove the rule and check to see where the bit touches now. If you're lucky, it will still be right on the 7/32" mark; if not, keep fiddling with it.

When everything is right, tighten the clamps, then check to make sure everything is still aligned. Also check to see that none of the clamps will interfere with the drill chuck. Remember that the chuck will come 3/8" closer to the work by the time you've drilled all the way through the shaft and sleeve. Safety tip: use masking tape to hold the wires out of the way.

Now drill the hole slowly, backing the bit out often to clear chips. Take care not to drill too far; Vee-blocks are not cheap. Does everything look centered? If not, something slipped.

The setup takes a while to get right. But once done for the first motor, you'll be able to do the second without any changes, provided everything is still clamped down tight. You should still check alignment before drilling the second shaft.

You now have two stepper motors with cotter pin holes drilled perfectly. The cotter pins should fit snugly. A little sanding or filing to clean up the drill exit burrs, and smile: you've just done precise machining by eyeball.

Constructing The Motor Box

Tools: Prick punch, metal scribe, 1 3/8" chassis punch, drills, or hand drill, machinist trisquare, machinist compass, pin vise, files, hand reamer.

Materials: 6"x5"x4" aluminum project box, modified stepper motors from Step 1, plastic training wheels, aluminum hex standoffs, #10-32 machine screws, wood scraps.

Step 1. Layout Hole Locations.

LIMBO uses a two-piece aluminum project box to protect the stepper motors and undercarriage wiring. With the cover mounted, it also serves as a rigid, lightweight mount.

Since the stepper motors are mounted directly to the box, it pays to be exact when doing the mounting hole layout so the drive shafts are accurately aligned.

Begin by locating the vertical center line of the side panels and the center of the top piece (scribing diagonals from corner to corner will do just fine). Measure all locations with respect to the center points, and use the trisquare to line up holes the same distance from center.

Or, you can tape templates to the panels and directly transfer the hole locations with a prick punch. Templates can be made by either enlarging the accompanying illustrations or ordering the full-sized plans (you can find the details in the parts list table).

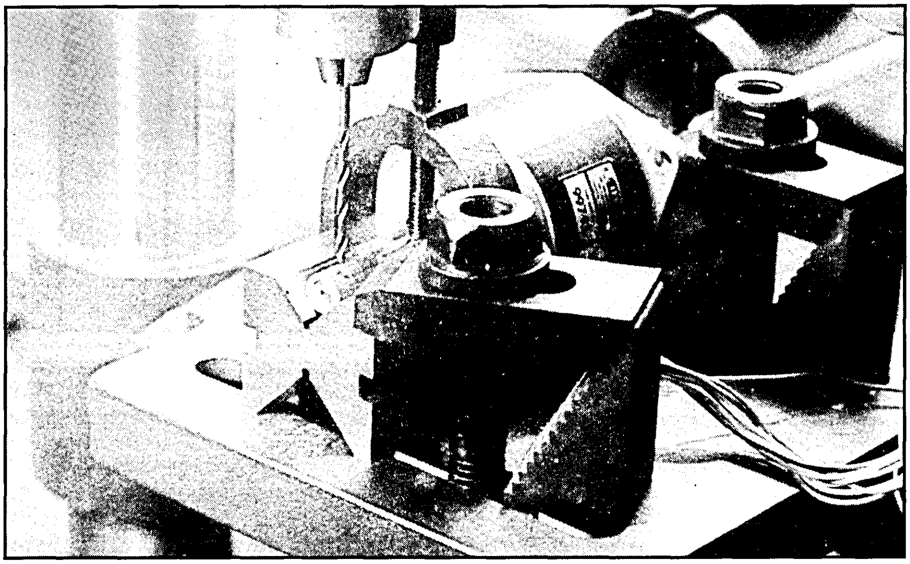
Whatever way you choose, lightly centerpunch all hole locations. Back up areas being centerpunched with scraps of wood to prevent deforming the panels.

Step 2. Drill and punch motor box holes.

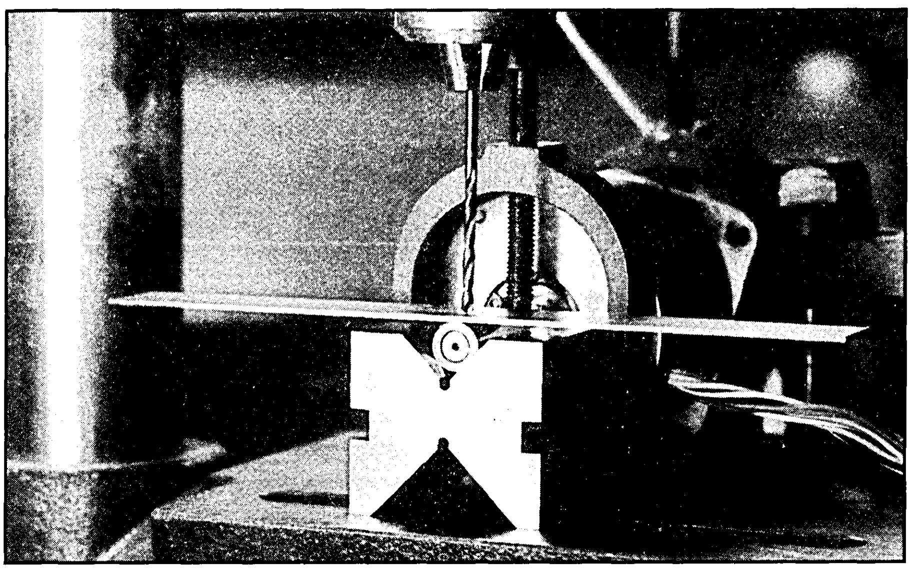
Drill 1/16" pilot holes at every marked hole location. You can use a hand-held drill for this if you're careful, but I prefer the drill press. Examine the results. Do any of the pilot holes seem off-center? If so, now's the time to correct this by nibbling away metal from the side you want the hole to move toward. Do this with the corner of a file or the prick punch. Aluminum is soft, so don't overdo it.

Once you're satisfied with your pilot holes, enlarge them with a 3/16" bit. The two shaft holes should be drilled with a 1/4" bit, then enlarged to 3/8" with the hand reamer. (Or, you can use a 1/2" chassis punch.)

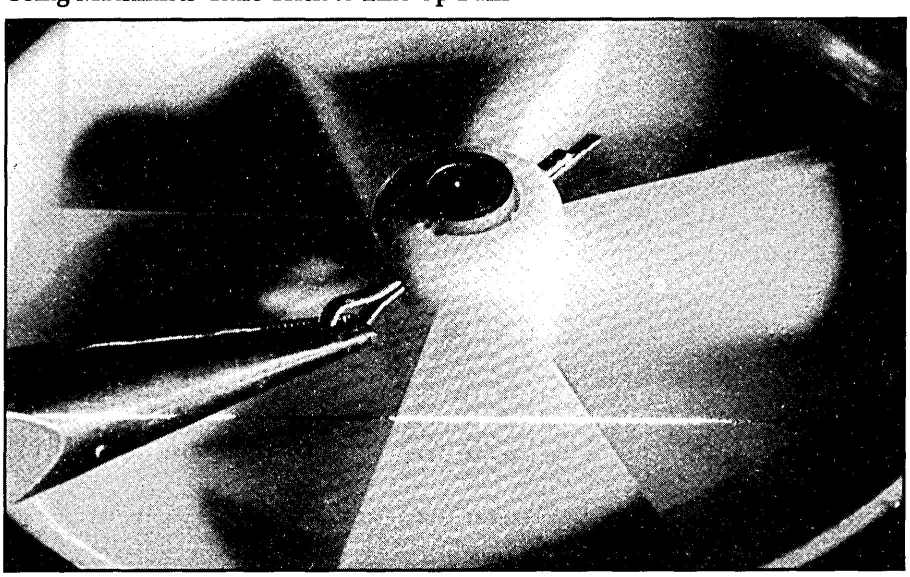
Punch the wire access hole with a 1 3/8" Greenlee chassis punch. The punch should be on the inside of the box, the die on the outside. It is much easier to turn the wrench this way, and it will make the inside edge rounded and



V-block Setup On Drill Press



Using Machinists' Rule Trick to Line Up Drill



Inserting the Cotter Pin with Needle Nose Pliers

smooth so it won't chafe the wiring. Oil the threads of the chassis punch before you begin punching; less friction will make the job easier and your punch will last longer.

If you don't have a chassis punch (they cost about $25), scribe a 1⅜" circle on the box before drilling the pilot holes. Use a nibbling tool to cut a circular slot starting from the center and spiralling out to follow the scribed circle line. You'll need to do some filing to smooth the cut. Either way, finish by deburring all the holes with a file or X-acto knife.

Step 3. Mounting the motors and wheels.

We need standoffs to mount the stepper motors. I special ordered the 1$^{13}/_{16}$" standoffs to save time. You might wish to find 2" spacers that you can file down to size, though you'll want to get the ends as square as possible.

First, loosely mount the standoffs to the stepper motors, then fit the motors inside the motor box one at a time. You'll probably need to slide the standoffs around a little to get them to line up with your mounting holes. Once you've gotten all three outside screws in, you can tighten the inside screws.

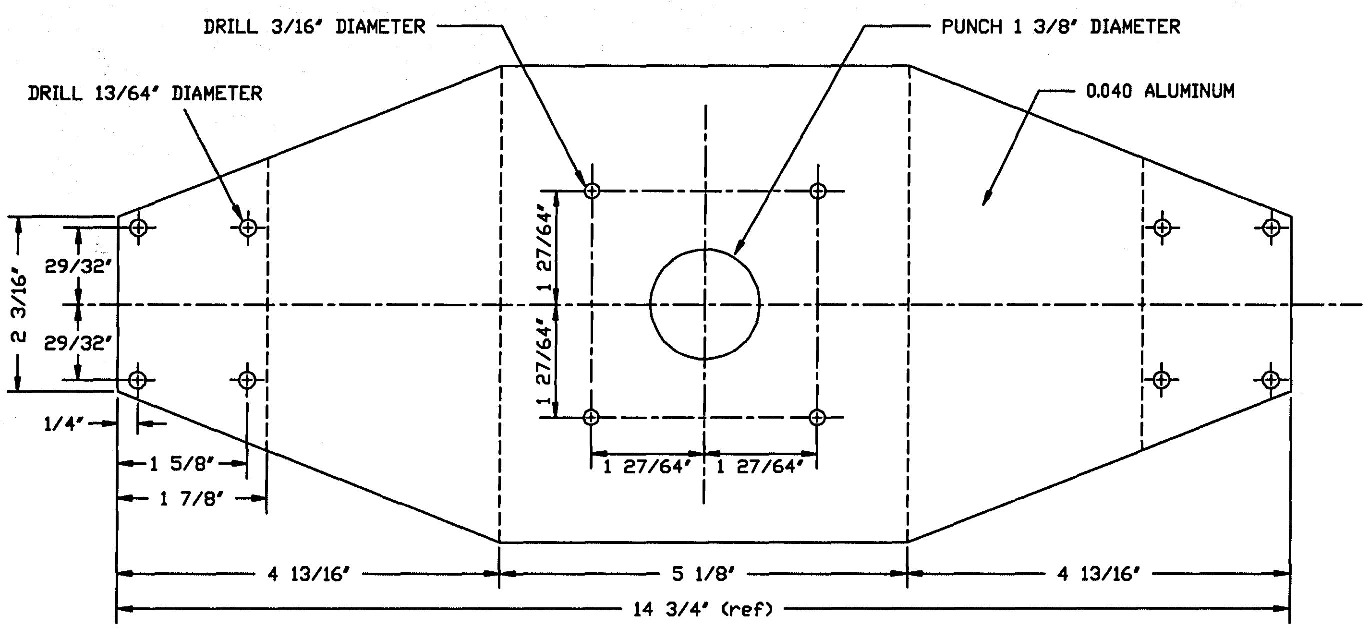
The shafts may look somewhat askew before you close the box up, but the box will flex a little with the bottom cover in place so the shafts should then appear to line up with each other. If they don't, loosen the outside screws a bit and slide the motors around until they do, then retighten.

Remove the adaptor sleeves from the motors and slide the plastic wheels onto the sleeves. Mark which sleeve went with which wheel and motor. With the cotter pin holes visible, mark the hub of each wheel adjacent to the holes. Remove the sleeves.

The hubs of the wheels don't protrude far enough to allow directly drilling the hole, so you'll have to use the $^3/_{32}$" bit in a pin vise. The holes will be angled inward slightly, so you'll need to offset them to compensate (see photo). Once you have the holes drilled, slide the sleeves back on the motors, lining up the holes with a small nail.

Without moving the shafts, slide each wheel onto its sleeve. Getting the wheels aligned can be tricky. You may have to drill the holes in the wheel hub to a larger size. Once aligned, force a new cotter pin through each hub/sleeve/shaft hole, then bend the ends of the cotter around the hub. The wheels are now

Castor Suspension Spring Diagram



solidly mounted, and the motor box is complete.

Making The Castor Suspension

Tools: Same tools as Motor Box procedure, plus aviation snips or bench shear, ball peen hammer, C-clamps, plastic protractor, bevel gauge.

Materials: 6"x14.75"x 0.040" Aluminum (5051), swivel castors, scraps of wood, #12-24x½" machine screws and nuts.

Step 1. Preparing the suspension spring blank.

The castor suspension spring for both front and back castors is a single piece unit made from a 6"x14.75" sheet of aluminum. Buy at least a few extra inches length when you have the metal shop cut a piece for you so you can discard the bent corners (metal dealers seem to save their best pieces for bigger customers). Try to get the width as close to 6" as possible to save trimming.

Using the trisquare, scribe a square line across the best end, then make all measurements relative to this line. Layout the fold lines first, then the tapered outlines, and finally centerpunch the hole locations. Use either direct measurements or transfer the locations from the motor box completed previously. Cut to the outline using either aviation snips or a bench shear, if you're lucky enough to have one. Drill and punch the mounting holes and the wire access hole as you did for the motor box.

Step 2. Bending the suspension spring and mounting the castors.

If you have access to a sheet metal brake, this step will be easy. If not, a few 5" C-clamps and some blocks of wood and a hammer are all that you need. Clamp the blank between two solid scraps of wood, straight edges lined up right on the scribed fold line. The first bends to make are the inner, wide ones. Set your bevel gauge to about 115°.

Begin the bend by pushing from one side of the metal blank with a block of wood. As the bend proceeds, you'll need to use considerable persuasion with the hammer to keep the bend crisp. Don't hit the metal directly with the hammer, but indirectly through a block of hardwood. Check the angle with the bevel gauge often. Take your time, and don't break your thumb (you'll need it later).

After you finish the four bends, mount the castors in place with #12 screws. Make sure that the castors can swivel freely over the screw heads. Take the castor/spring assembly and temporarily mount it to the motor box with #10 screws and nuts. Does it stand up straight and proud? Is it level with no wobbles? No? Bend the spring a little to make it level.

The angles given should put the castors slightly lower than the drive wheels so that the castors contact the ground before the drive wheels do. This is called preload.

The function of the preload is to provide enough tension in the suspension to prevent rocking, but not so much that the drive wheels lose traction. If the preload seems a bit much now, remember you'll have at least five more pounds of robot. Put a bag full of sugar on top (C&H granulated works best). Adjust the preload if it seems either too stiff or too wobbly.

The Bumper Contact Skirt

Tools: Same tools as above, plus nibbling tool, 1¼" wood boring auger or spade bit, sabresaw or coping saw.

Materials: 12"x24"x¼" plywood, #10-32 screws and "Tee" nuts, #4 roundhead woodscrews, microswitches, 22 ga. stranded hookup wire (9 colors), 24 ga. solid buss wire, nylon cable ties, 9-pin connector, 14" Superpot tray.

Step 1. Making the bumpers.

A robot's interactions with its environment can be only as good as its sensor data. The bumper contact skirt is the first and most basic of LIMBO's sensor suite.

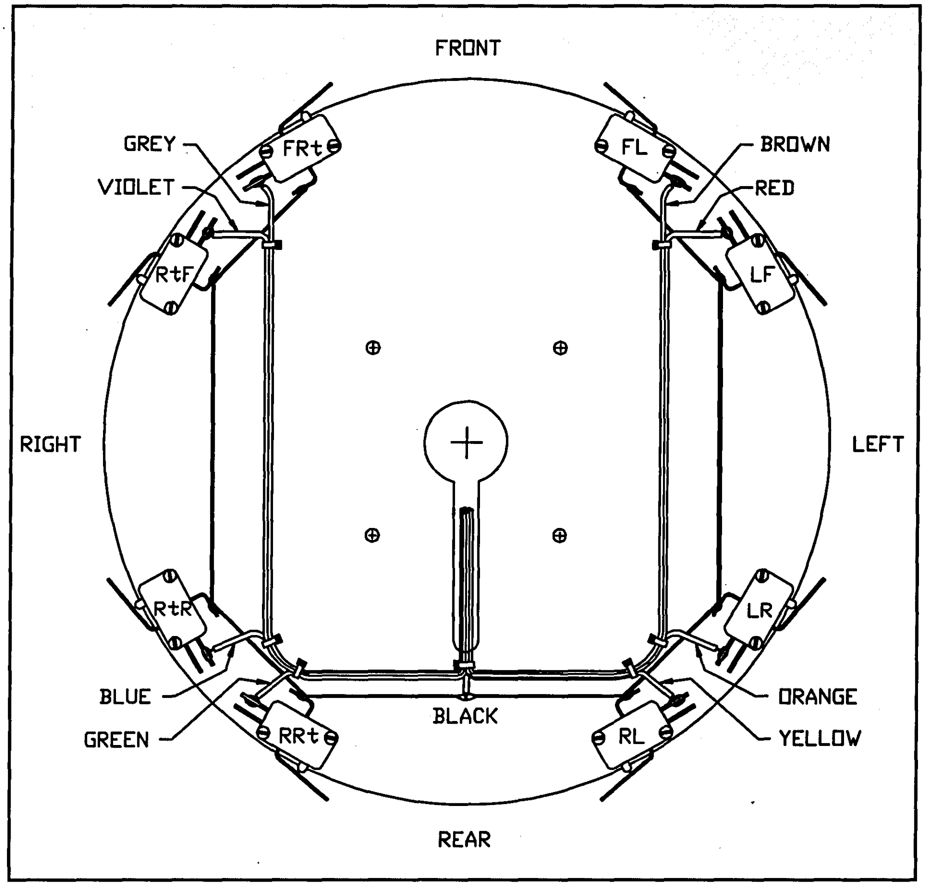
For a $2 flower pot tray to function as a super sensitive and reliable bumper contact sensor, you must make a few modifications (i.e., don't plan on using this baby to start tomatoes next year).

First mod is to create four independently moveable bumper segments by cutting ¼" slots at the 90 degree points around the perimeter of the tray. Each bumper segment actuates two microswitches, each microswitch covering about ⅔ of the segment with an overlap of ⅓ in the middle. With these three contact regions per bumper segment, the bumper skirt can detect contacts in twelve different directions.

Mark the location of the slots with an indelible marker (if you goof, use rubbing alcohol to make the marks delible). Cut the slots with a nibbling tool so they run perpendicular from the rim of the tray all the way to the first raised ring on the underside of the tray. (The underside will be the top of the bumper skirt when finished.)

Once you've cut all the slots, you'll notice that the rim of the tray will come in a little, closing up the slots. You'll need to trim a little plastic from each cut edge to maintain a constant ¼" slot width. I found my bench belt sander an excellent tool for this, but an X-acto knife will work, too.

Chamfer the rim's corners.



Bumper Contact Wiring Diagram

The bumper segments should move easily to slight finger pressure, and two adjacent bumpers should not interfere with each other for simultaneous ¼" bumper movements. If they interfere, remove more plastic. Don't go overboard because you want to keep the gaps as small as possible. Last, drill a ¼" hole in the exact center of the tray.

Step 2. Making the support structure.

Use the machinist compass to scribe three circles, 11", 5¾" and 4⅞" diameters, onto ¼" plywood. Scribe two sets of two perpendicular diameter lines on the 11" disk layout and label them 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°, proceeding counterclockwise.

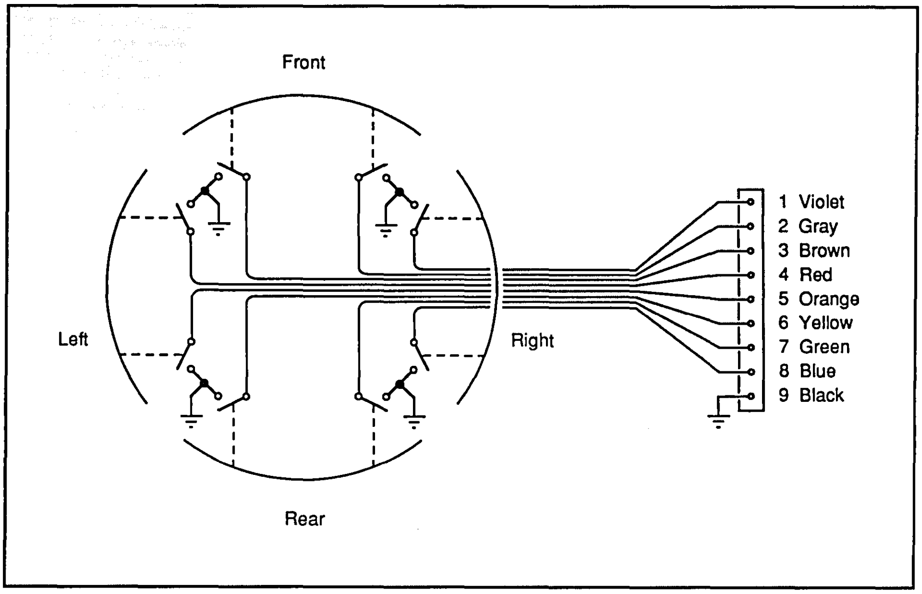
Further label the 90° position with "F" (front), the 0° with "L" (left), 180° with "Rt" (right) and 270° with "R" (rear). Right and left may seem reversed at first, but the layout will face down. Scribe a 1⅜" circle in the center.

Align the motor box wire access hole and mounting holes with the circle and the 45°/225° and 135°/315° lines. Transfer the mounting hole locations to the large disk as you did with the suspension spring. Cut the disks about 1/16" outside the line with a bandsaw, sabre saw, or coping saw. Sand the disks to final dimension. Eight microswitches will be mounted on the large disk, two for each bumper segment. Microswitch mounting hole locations are measured from the 45°/225° and 135°/315° lines and labeled as shown.

Now drill a ¼" hole in the exact center of each disk. The rest of the mounting holes are drilled with all three disks and the bumper skirt temporarily bolted together. Align the 45°/225° and 135°/315° lines of the 11" disk with the slots of the bumper skirt so the mounting holes will line up with the slots. Mark the bumper segments inside with the corresponding directions of the 11" disk (i.e., front and rear, left and right).

Drill the mounting holes with a 15/64" bit (the mounting holes are larger in the plywood than the motor box and suspension spring to accommodate tee-nuts). Put a tee-nut through each hole as it is drilled to maintain perfect alignment.

When all four holes are drilled, remove the bolt from the center and drill the wire access hole out to 1¼". This is a big hole to drill, so take it easy. The wire



Color Coded Wiring Schematic

access hole is purposely smaller here than with the motor box and suspension spring to prevent wire chafing against metal. (Also, 1¼″ was the largest spade bit I had.)

Finally, drill a ⅜″ hole 2½″ from the edge of the 11″ disk on the line marked "R." This hole forms one end of the wire access slot. The rest of the slot is made by two parallel coping saw cuts ending at the 1¼″ hole. Wires from the bumper contacts will be routed through this slot over the suspension spring and up through the wire access hole.

Step 3. Mounting and wiring the microswitches.

Drill starter holes for all the microswitches with a 1⁄16″ bit. Drill no deeper than ⅛″ (you may want to put masking tape on the drillbit for a depth gauge). Mount each microswitch as shown with two ⅝″ #4 roundhead woodscrews. When they're all mounted, temporarily assemble the bumper skirt and disks together.

With the upside-down assembly supported only in the center so nothing touches the bumper skirt plastic anywhere, press each bumper segment inward about ¼″ at several locations on each segment. Note over what arc each microswitch is activated. Ideally, each switch will be activated over ⅔ of its bumper segment. You'll probably have to bend the actuating levers into shallow S-curves in order to get the proper coverage.

If you have trouble telling whether a switch activates, hook up a continuity tester. Disassemble the disks so the 11″ disk can sit flat on the bench.

Now you can wire up the switches. I like to color-code wiring harnesses because I ultimately spend more time rebuilding/repairing my robots than in original construction. It takes an extra minute or two now, but saves hours later.

First wire the common terminals together with 24 ga. bare buss wire, starting with the "FL" switch and working clockwise. Loop the wire around each common terminal, then solder and dress square and flat on the board, making straight runs between terminals.

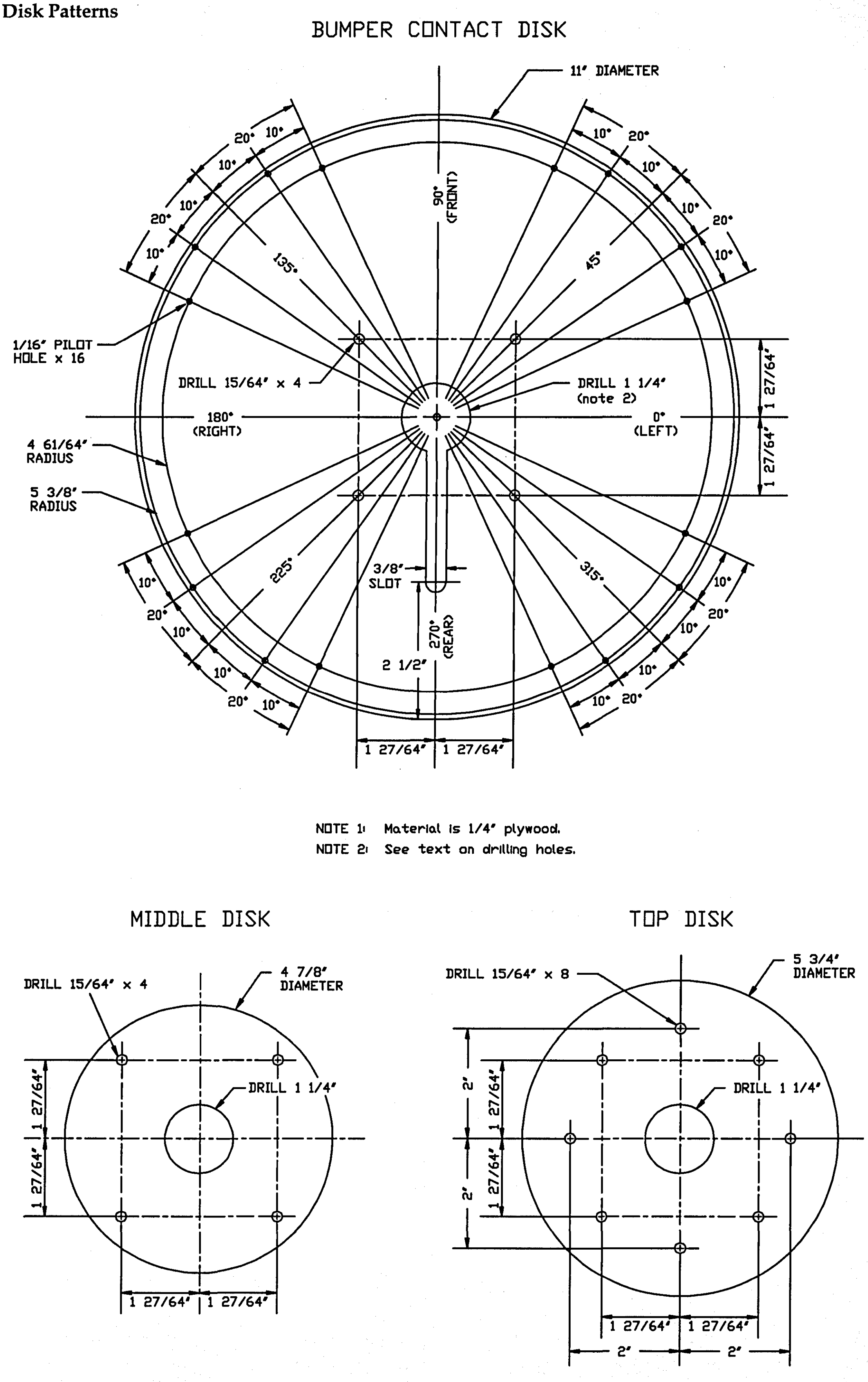
Terminate the wire at the "FR" switch. The wire should form a "U" shape with the open end pointed forward. At rear center, solder a 20″ length of black 22 ga. stranded wire directly to the bus wire and dress it to run forward to the wire access slot.

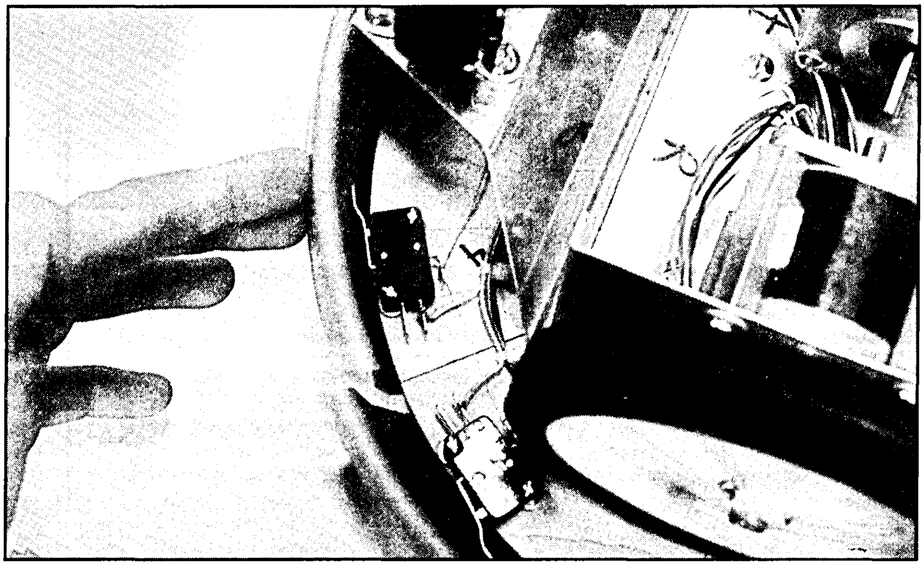
Next, cut 30″ of 22 ga. stranded hookup wire for each switch (use color coded from brown to grey). Strip and tin one end of each wire. Each switch gets its own color wire, starting with brown at the "FL" switch and working clockwise until you get to grey at "FR."

Editor's note: Standard color codes are, in order: black, brown, red, orange, yellow, green, blue, violet, grey, white. They stand for the numbers 0 (black) through 9 (white). We normally use black for ground so you'll start with brown (1) for the switches.

Now neatly route the wires along the common wire as shown. Use a nylon cable tie where each wire enters the

Plywood Disk Patterns





Actuation of Bumper Contact Microswitch

bundle. Wire bundles from the two halves of the board should meet at the black wire, then continue forward as one bundle to lie down in the slot to the wire access hole.

To finish the mobility base, assemble the motor box, suspension spring, bumper skirt, and support disks with tee-nuts in the top disk and #10-32 machine screws from the motor box.

Next time, we'll do the superstructure and (hold on to the cool end of your soldering iron) the stepper drive electronics.

♦ ♦ ♦

LIMBO Parts List

Qty.	Item	Description	Supplier
2	Stepper Motors	IMC Magnetics #023-2024-12	CH Sales
2	Bike training wheels	5 1/4" dia. plastic, 3/8" hub bore	K-Mart
6	Hex standoffs	3/8" x 1 13/16" aluminum, 10-32 thread	Olander Corp. (RAF # 2259)
2	Cotter Pins	3/32" x 1" zinc-plated	
4	Nesting brass tubes	1 ea: 9/32", 5/16", 11/32", 3/8" od 1/32" walls, 12" lengths	
12	Machine screws	10-32 x 3/8" steel binder-head	
4	Machine screws	10-32 x 3/4" steel binder-head	
6	Plain washers	#10, steel	
4	Tee-nuts	10-32 thread, steel	
8	Machine screws	12-24 x 1/2" steel round-head	
16	Wood screws	#4 x 5/8" steel round-head	
8	Lock washers	#12 steel, internal tooth	
4	Tee-nuts	10-32 thread, steel	
8	Hex nuts	12-24 steel	
1	Plywood	12" x 24" x 1/4", both sides good	
1	Sheet aluminum	6" x 14 3/4" x 0.040", 5051 alloy	
1	Project box	2-piece alum. project box 6"x 5"x 4"	Digi-Key #L105-ND
1	Flower pot tray	14" SuperPot	K-Mart, Ernst
2	Swivel castors	2" dia. wheel, ball bearing swivels, flange mount	
8	Micro-switches	Unimax/C&K #2TMT15-4 snap-action, Newark leaf actuated	
1	Connector	Waldom female 9-pin, 0.100" centers	Digi-Key #WM2007

Misc.:

9 assorted colors of # 22 ga. stranded hookup wires
24 ga. solid buss wire
Nylon cable ties
Crimp terminal for 9-pin connector
Rosin-core solder
Rosin flux paste, such as Kester's

Suppliers:

Digi-Key 701 Brook Ave. South P.O. Box 677 Thief River Falls, MN 56701-0677	CH Sales P.O. Box 5356 Pasadena, CA 91117-9988	Olander Corp. 14612 NE 91 Redmond, WA 98052

Newark Electronics is nation wide. Look in phone book for local rep.

♦ ♦ ♦

PostScript Programming, Part II

It was a dark and summery night. As I pulled in behind the Micro C international offices, I spotted a furtive figure outlined by a dimly lit window. It was Larry. Larry Fogg, ex C programmer, now grappling with something neither he nor I really understood. FORTH.

What follows, the result of that evening, is true. So true, in fact, that only one name has been changed. (Hint: PostScript.)

Surely stack-oriented programming destroys brain cells at a higher rate than any other form of mental self-abuse. At least that's my experience after spending several late nights trying to make sense of the "who's on first" game of stack tracking.

Last issue we sidestepped the meat of PostScript programming by focusing on one simple aspect: binary image creation and manipulation. This time around, I'll try to give you a feel for some of PostScript's other capabilities. I'll do it by generating (you guessed it) Yet Another Fractal.

Geometric Fractals

First, a word about geometric fractals for those of you who've yet to subject your computer to this particular form of digital torture.

A geometric fractal superimposes a pattern of line segments (called a generator) over a larger line segment. Then, for each segment of the generator, it superimposes another, smaller, copy of the generator.

And for each segment of the smaller copy ... ad infinitum (ad nauseum?). Take a look at Figure 1 for a graphic example.

Unlike fractals such as the Mandelbrot set and Julia sets, the detail at each level of a geometric fractal is *exactly* the same as at any other level. I chose to use the classic square snowflake for the example. (See Figures 1 and 2.) Take a look at "Introduction To Fractals" in Issue #33 for more information on the nature of geometric fractals.

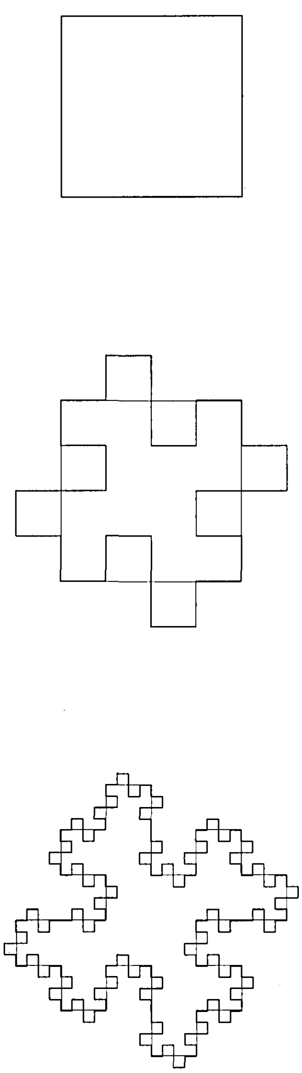


Figure 1—Geometric Fractal Construction

On To The Code

Let's take a stroll through the sample program, GEOFRAC, and pick it apart. (See Figure 3.) I'll assume you've read Part 1 from last issue and have some familiarity with the fundamentals of PostScript programming.

Ignore the header comments for the time being and begin with the variable section. The first items of interest are the array definitions. In general:

```
/ArrayName ArraySize array def
```

defines an array. PostScript arrays *always* index from 0 to ArraySize-1.

Xval and Yval together hold an array of 25 points defining an area around a line segment. (See comments in Figure 3.) The endpoints of the segment live in locations 10 and 14. The generator can easily be drawn by choosing the appropriate points in Xval and Yval.

What about str? It's also an array, but of the particular type, string. Actually, PostScript looks at dang near everything as an array, even procedures (which it thinks of as "executable arrays").

Array Access

Xval and Yval define more points than necessary for the snowflake generator. This allows for use of larger and more complex generators; you'll only need to define a new value for the number of points in the generator (GeneratorSize) and change the code in LoadGenerator.

The following line assigns a value to an array location:

By Larry Fogg
Micro C Stiff

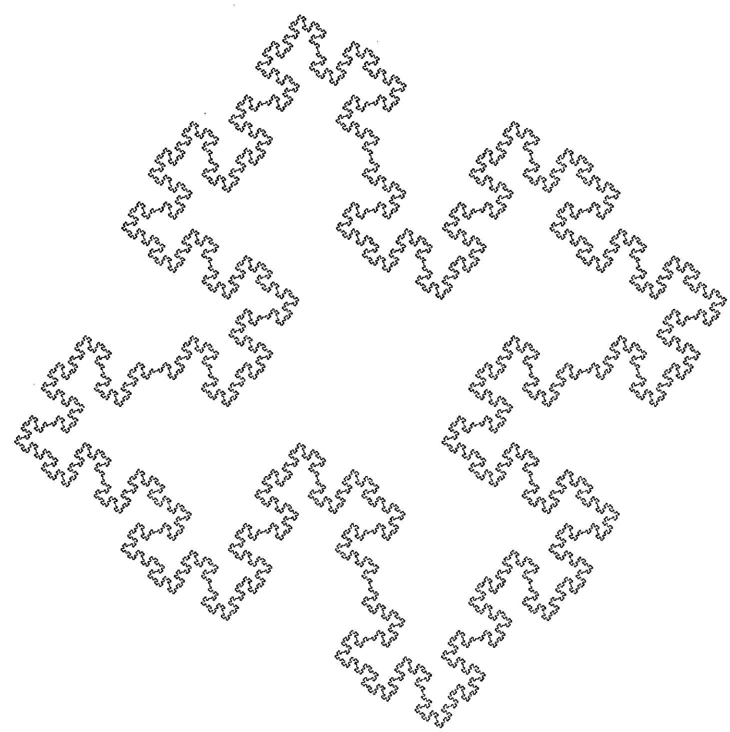


Figure 2—Square Snowflake

```
ArrayName Index Value put
```

Unlike arrays in other languages, you can stuff any mix of types into a PostScript array; witness the executable arrays. We can't index more than one dimension, but there's no law against arrays of arrays.

LoadGenerator puts a sequence of integers into Generator. Using these values to index into Xval and Yval lets you connect the dots (go from point 10 to point 11 to point 6...) and draw the generator.

To read a value from an array:

```
ArrayName Index get
```

and the interpreter will push the value onto the stack.

Procedures

The procedure LineLength accepts two points and finds the distance between them. We'll use it later to limit the smallest size line segment in our fractal. I thank Pythagoras for the math.

Notice the roll operator. It takes three objects off the stack, rolls them one location up (the top item goes to position three), and pushes all three back onto the stack. The direction of roll can be negative as well. An equivalent command would be:

```
3 -2 roll
```

I find it most helpful to draw pictures of the stack contents at each point in a procedure. Figure 4 shows how LineLength works. Each stack diagram shows the contents *after* the operation above the diagram. Any time you get into trouble with PostScript, I'd recommend this technique to see what your code is really up to.

Stack Versus Variables

If you wanted, X1, X2, Y1, and Y2 could be declared as variables rather than passed to LineLength on the stack. And you could set up some temporary variables within the procedure. This would make the code easier to read, and it would feel more like C or Pascal.

But you'd be fooling yourself and probably hurting performance. Remember, variables get buried in the current dictionary. The interpreter has to root around until it finds the variable you reference—potentially a much slower process than manipulating the top elements on the stack. "When in PostScript, do as PostScriptoids do."

The rule of thumb should be: when your code gets so confusing that even you don't have a clear idea of what's going on, throw in some variables for clarity.

The bulk of the code in GEOFRAC is dedicated to finding the point coordinates stored in Xval and Yval. It works for line segments in any orientation. I haven't shown the code for each point; you should see the pattern after looking at one or two.

dY and dX hold the Y and X displacement between adjacent points in the 25 point array. It seems odd to use dX to find the y coordinates and dY for the x coordinates, but if you draw it out it'll make sense.

Control Structures

PostScript supplies a fairly complete group of control structures and I think the first one is really slick. It performs a procedure for each value of an array and looks like this:

```
ArrayName {loop body} forall
```

forall pushes one of the array values onto the stack at the head of each loop. In our Draw procedure, the forall loop uses each value in Generator to find the next point to draw to. A very convenient way to draw the generator.

The next procedure, PushSegments, shows a more standard for loop.

```
StartCount Increment EndCount
  {loop body} for
```

At the top of each loop, for pushes the loop count value onto the stack, making it available for use within the body of the loop. More on PushSegments' reason for existence in a bit.

The last procedure, Generate, uses most of the rest of Postscript's control operators plus its conditional statements. Conditionals have these forms:

```
Value1 Value2 lt
  {body} if

Value1 Value2 lt
  {then clause}
  {else clause} ifelse
```

lt (less than) compares Value1 and Value2 and pushes a boolean value onto the stack accordingly. It works for both numeric and string comparisons. In the first case above, if Value1 is less than Value2, the interpreter executes the body. (The sad thing is, it can do it over and over again.) The second case extends the condition to if-then-else.

The full complement of comparison operators includes:

```
gt — greater than
lt — less than
eq — equal to
ne — not equal to
ge — greater than or equal to
le — less than or equal to
```

Figure 3—PostScript Code to Generate Figure 2

```
%!PS-Adobe-1.0
%%Creator: Larry Fogg
%%Title: Fractal Snowflake-2
%%CreationDate: July 20, 1989  9:00 am
%%For: Micro Cornucopia Issue #50
%%BoundingBox:230 230 572 580
%%EndComments

gsave                % save Ventura's graphics state

%%EndProlog

%%Page: ? ?

% The pattern below defines a matrix of points surrounding the
% line segment defined by points 10 and 14.

%                  *0  *1  *2  *3  *4
%                  *5  *6  *7  *8  *9
%                  *10 *11 *12 *13 *14
%                  *15 *16 *17 *18 *19
%                  *20 *21 *22 *23 *24

% Here comes the actual PostScript code

% variables *************************************

/GeneratorSize 9 def  % # points in the generator

/Generator GeneratorSize array def

/Xval 25 array def
/Yval 25 array def

/Resolution 10 def

% procedures ************************************


/inch {72 mul} def


/LoadGenerator       % connect point 10 to 11 to 6 to...
{
  Generator 0 10 put
  Generator 1 11 put
  Generator 2 6 put
  Generator 3 7 put
  Generator 4 12 put
  Generator 5 17 put
  Generator 6 18 put
  Generator 7 13 put
  Generator 8 14 put
} def  % LoadGenerator


/LineLength          % finds line length - params = (X1 X2 Y1 Y2)
{
  sub                % find delta Y
  dup                % make another copy of delta Y
  mul                % square delta Y
  3 1 roll           % rotate X parameters to top of stack
  sub                % find delta X
  dup                % make another copy of delta X
  mul                % square delta X
  add                % delta X squared + delta Y squared
  sqrt               % ((Y2 - Y1)^2 + (X2 - X1)^2)^1/2
} def  % LineLength


/FindMidPoint        % finds midpoint of segment (X1 X2 or Y1 Y2)
{
  add                % just a simple average
  2 div
} def  % FindMidPoint
```

Continued on page 36

Reader Service Number 10

The main loop in Generate is a loop loop. Let me elucidate. This structure:

```
{
  ...statements...
  boolean
    {exit} if
  ...statements...
} loop
```

will execute until boolean becomes true. You can put the exit *anywhere you want*. Put it at the top of the loop, and you have a do-while structure; put it at the bottom for a do-until; put it in the middle and...God knows what you have! Roll-your-own control structures. Imagine that.

The Dave-orithm

Let's get back to implementing geometric fractals. Dave wandered in one night while I was pounding away on GEOFRAC and offered a truly elegant algorithm. Here goes.

Push x and y values defining the starting line segment onto the stack. Now superimpose the generator on the line segment and find the length of an individual generator segment. If the length is less than some set value (Resolution), draw the generator. If the length is greater, push all segments of the generator onto the stack. That's it. Just repeat until the stack is empty.

Here's what happens. The stack grows until the segment on top becomes small enough to pass the resolution test. At that point there will be GeneratorSize-1 segments (that's one complete generator) on the stack that will pass the test. After they've all been drawn, the interpreter tests the next segment of the next largest sized generator. It can't get through the resolution test, so the interpreter pushes another set of generator segments onto the stack. They pass the test, so they get drawn. And on, and on....

Implementation Details

The first line in Generate's loop body uses the built-in PostScript operator, count, to return the number of items on the stack. When count reaches zero, we exit the loop.

The lineto operator draws a line from the current position to the coordinates it receives as parameters. So it's important to draw the fractal continuously, rather than jump from one segment to another disjointedly. PushSegments puts the last segment of the generator on the stack

Continued from page 34

```
/FindXSegValues     % loads X-vals on line seg into point array (X1 X2)
{
  2 copy            % duplicate the parameters
  Xval 14           % load X2 into position 14
  3 -1 roll         % put params in proper order for put
  put               % load value into array
  Xval 10           % load X1 into position 10
  3 -1 roll
  put

  FindMidPoint      % find x-coord of center point (12)
  Xval 12           % load it into midpoint of array
  3 -1 roll
  put

  Xval 10 get       % do point 11
  Xval 12 get
  FindMidPoint
  Xval 11
  3 -1 roll
  put

  Xval 12 get       % do point 13
  Xval 14 get
  FindMidPoint
  Xval 13
  3 -1 roll
  put

} def  % FindXSegValues


/FindYSegValues     % loads Y-vals on line seg into point array (Y1 Y2)
{
  2 copy            % duplicate the parameters
  Yval 14           % load Y2 into position 14
  3 -1 roll         % put params in proper order for put
  put               % load value into array
  Yval 10           % load Y1 into position 10
  3 -1 roll
  put

  FindMidPoint      % find y-coord of center point (12)
  Yval 12           % load it into midpoint of array
  3 -1 roll
  put

  Yval 10 get       % do point 11
  Yval 12 get
  FindMidPoint
  Yval 11
  3 -1 roll
  put

  Yval 12 get       % do point 13
  Yval 14 get
  FindMidPoint
  Yval 13
  3 -1 roll
  put
} def  % FindYSegValues


/FindDeltas         % Finds dX and dY (no params)
{
  /dX {Xval 11 get
       Xval 10 get
       sub} def
  /dY {Yval 11 get
       Yval 10 get
       sub} def
} def  % FindDeltas


/FindXValues        % Loads off axis X-vals in point array (no params)
{
  Xval 10 get       % point 0
  dY 2 mul sub
```

Continued on page 37

Continued from page 36

```
    Xval 0
    3 -1 roll
    put

    Xval 11 get       % point 1
    dY 2 mul sub
    Xval 1
    3 -1 roll
    put

    % etc., etc.

    Xval 13 get       % point 23
    dY 2 mul add
    Xval 23
    3 -1 roll
    put

    Xval 14 get       % point 24
    dY 2 mul add
    Xval 24
    3 -1 roll
    put

  } def  % FindXValues

  /FindYValues        % Loads off axis y-vals in point array (no params)
  {
    Yval 10 get       % point 0
    dX 2 mul add
    Yval 0
    3 -1 roll
    put

    Yval 11 get       % point 1
    dX 2 mul add
    Yval 1
    3 -1 roll
    put

    % etc., etc.

    Yval 13 get       % point 23
    dX 2 mul sub
    Yval 23
    3 -1 roll
    put

    Yval 14 get       % point 24
    dX 2 mul sub
    Yval 24
    3 -1 roll
    put

  } def  % FindYValues

  /Draw               % draw generator over line segment - must be at
  {                   % first position before calling Draw (see Generate)
                      % (no parameters)
    Generator
    {
      dup             % copy the generator value
      Xval exch get   % use it to index into x-coord array
      exch            % put generator value back on top of stack
      Yval exch get   % use it to get the y-coord
      lineto          % draw the line segment
    } forall          % do it for each point in the generator
  } def  % Draw

  /PushSegments     % puts endpoints of each segment of generator on stack
  {                 % (no params) leaves gen segs in reverse order (8-1)
    GeneratorSize 1 sub -1 1   % for count = GenSize-1 downto 1 do...
    {
      dup dup dup     % need one of these puppies for each parameter

      1 sub
      Generator exch get
```

Continued on page 38

first to ensure proper drawing order.

Notice that Draw only defines the path; it doesn't stroke it onto the page. This can cause problems. In theory we could wait until the entire snowflake has been defined before calling stroke. But PostScript has a limitation of 1500 points in the current paths; we'll easily exceed that with the snowflake.

> **The initiator, as well as the generator, can be changed with wonderful effects.**

So every time we Draw a generator, we should stroke it as well. But stroke does an implicit newpath and we lose track of the current drawing point. A call to currentpoint solves this problem. currentpoint pushes coordinates onto the stack for moveto to use after the stroke call, and we're back where we belong.

In main we do four calls to Generate, one for each of the four sides of a square called the initiator. The initiator, as well as the generator, can be changed with wonderful effects. For example, I've made Christmas trees and giant space snails. *Editor's note: He promised he'd spend his vacation working on a finely honed image of snail bait.*

Recursion....

You'll recognize the stack games we've played in the Dave-orithm; when you write recursive code in a high level language, the compiler generates the same kind of results. We've just done the recursion at a lower level, without the benefit and simplicity of a procedure calling itself.

I originally wrote the procedure Generate with this high level type of recursion. But I had fits trying to localize the arrays Xval and Yval. They must be local so that each call to Generate creates new array copies and none of the array contents get overwritten.

I know this can be done with judicious use of dictionaries (stacks); creation of a new current dictionary at the beginning of Generate would effectively localize the arrays. But I had no debugging facility at that time, and after a day of total confusion Dave's algorithm was a breath of fresh air. I had it running in just a few trillion microseconds.

Encapsulated PostScript

So far we've talked about standalone PostScript programming. But what if you want your graphic description to be included in a page description generated by another program? For example, I want to feed the snowflake to Ventura.

The answer is to encapsulate, or surround, the snowflake description with information that Ventura (or whomever) needs to incorporate the fractal into page 33 of this issue of *Micro C*. Now Carol can just open a frame in page 33, load GEOFRAC.EPS, et voilà: a complete page description. No nasty paste-up.

Most of the encapsulation comments speak for themselves. The initial "%!" identifies this file as EPS.

GEOFRAC.EPS doesn't actually generate any output; Ventura takes care of the printing chores. So we specify 0 Pages and comment out the showpage at the end of main. To make GEOFRAC a standalone PS file, just uncomment the call to showpage.

BoundingBox tells Ventura where the snowflake goes and how big it will be. We specify X1, Y1, X2, and Y2 in points (72 points per inch).

Often a program won't know the values for DocumentFonts, Pages, or BoundingBox until it has finished execution. For example, Ventura can't tell how many pages an article will take up until all the text has flowed in. And perhaps the last page of the article will reference an obscure font like Zapf Dingbats. So these three parameters can be deferred to the trailer comments with an entry in the header like:

```
%%Pages: atend
```

Also, BoundingBox makes sense only for files to be included in a single page, like our snowflake. If you have a multiple page EPS file, omit BoundingBox.

And don't try to explain the encapsulation comments with standard PostScript comments. As soon as Ventura (or whatever) sees a line without "%%," it assumes we're out of the encapsulation comments.

After EndComments comes the Prolog. This section includes any commands that will apply to the entire EPS file. Ven-

```
    Xval exch get  % X1
    4 1 roll

    Generator exch get
    Xval exch get  % X2
    3 1 roll

    1 sub
    Generator exch get
    Yval exch get  % Y1
    exch

    Generator exch get
    Yval exch get  % Y2

  } for
} def  % PushSegments


/Generate            % procedure to generate the curve (X1 X2 Y1 Y2)
{
  4 copy             % must moveto first position before calling Draw
  pop                % don't need Y2
  3 1 roll           % put X2 on top
  pop                % get rid of it
  exch               % put em in the right order (X1 Y1)
  moveto

  {                  % top of while loop
    count 0 eq       % count items on stack
    {exit} if        % exit loop if stack is empty
    4 copy           % dup parameters
    FindYSegValues % find point locations
    FindXSegValues
    FindDeltas
    FindXValues
    FindYValues

    LineLength Resolution lt      % LineLength < Resolution?
    {                             % yes, draw this generator
      Draw
      currentpoint                % save current location
      stroke                      % does newpath also
      moveto                      % move back to current location
    }
    {
      PushSegments % else overlay the generator
    } ifelse
  } loop             % bottom of while loop
} def  % Generate


% main program ***********************************

0.001 inch setlinewidth
LoadGenerator        % load values defining generator

4.5 inch 7 inch 4.375 inch 4.375 inch Generate
7 inch 7 inch 4.375 inch 6.875 inch Generate
7 inch 4.5 inch 6.875 inch 6.875 inch Generate
4.5 inch 4.5 inch 6.875 inch 4.375 inch Generate
stroke

%showpage            % print the page

%%Trailer
grestore             % restore Ventura's graphics state
% end
```

♦ ♦ ♦

tura probably saves its graphics state (current path, etc.) before loading the snowflake, but just to make sure, gsave saves it again. That's all we need for our Prolog.

Next comes an individual header for each page and the page's description. In a multi-page EPS file, you'd see something like:

```
%%Page vi 8
```

This might be page vi of an introduction and the 8th actual page of the EPS file. Given this information, Ventura could be told to either include pages iii through ix, or include the 5th through 11th pages. Page makes no sense for the snowflake, so we enter question marks instead of page numbers. Any fonts used by the page get listed here, too.

The final section of an EPS file contains the Trailer. All we care about for the snowflake is restoring the graphics state that we saved in the Prolog. By the way, you won't need the standard PostScript ^D EOF mark for an EPS file. At least not for Ventura; it'll accept either ^D or ^Z.

GoScript

Picture this debugging session from Hell. You send your PostScript code to the printer. Nothing happens. Make a guess as to what's wrong and fix your code. Send it off to the printer again. Nothing happens. Make another guess. Nothing happens.... No interaction here; the printer can't tell you why it died. So you're flying blind.

Enter GoScript, a PostScript interpreter. I discovered GoScript while leafing through a back issue of *Micro C.* (I really should read the magazine more often.)

LaserGo, Inc., bills GoScript as a utility to print PostScript output on a variety of non-PostScript printers. GoScript translates the PostScript code into escape sequences that your printer can understand. Most folks will probably be interested in this ability of GoScript: adding a PostScript front-end to their printer.

I was after a PostScript debugging tool and GoScript provided a good one. You can invoke GoScript in interactive mode and enter code a line at a time, just like a BASIC interpreter. Or you can feed it an entire file. I did my darnedest to make it choke, but GoScript caught every error I threw at it: misspellings, out of range array subscripts, missing "def" at the end of a procedure, too many points in a path, etc.

It does annoy me that GoScript will only say, for example, that an error has shown up "near lineto." I may have used the lineto operator 100 times in the program. Which one is it? Line numbers for the errors would be much more useful.

But I can live with the error messages. If you futz with PostScript, GoScript will make life much easier. Spendy, but definitely recommended.

GoScript Plus—$395
LaserGo, Inc.
9235 Trade Place, Suite A
San Diego, CA 92126
(619) 530-2400

That's All Folks

Creating graphic images interests me much more than manipulation of text. That being the case, I've ignored PostScript's text capabilities. But PostScript has great power over the printed word. Check out Adobe's documentation (cited last issue) for complete text on text.

Also, look into PostScript's ability to control access to files, dictionaries, and arrays, do type conversions, and twiddle bits with the best of 'em. Powerful stuff.

I hope that, with what I've shown you and a bit of experimentation on your part, you'll be well on your way to PostScript paradise. Enjoy.

♦ ♦ ♦

Figure 4—Stack Diagrams of LineLength in Operation

	sub	dup	mul	3 1 roll	sub	dup	mul	add	sqrt
Y2	dY	dY	dY^2	X2	dX	dX	dX^2	dY^2+dX^2	length
Y1	X2	dY	X2	X1	dY^2	dX	dY^2		
X2	X1	X2	X1	dY^2		dY^2			
X1		X1							

♦ ♦ ♦

UNIX Packages For The PC

A Close Look At Yet Another Clone Market

Want to run UNIX? Aren't sure which flavor to get? This look at the major players should give you a very good feel for the market.

I had a problem. I had to choose a UNIX system for the office and I couldn't find any hard information on what was good and what wasn't. I started out with a Microport System V for the 286, followed by a SCO XENIX 286. By fall of 1987, I could see that the 286 had no UNIX future, so I upgraded our AT clones with the Intel Inboard 386. Since then I've become familiar with the 386 versions of UNIX.

One thing is clear. Rapid evolution assures that you will not get stuck with a lemon. Plus, the current players are extremely competent. Add in a little cutthroat competition and you've got a buyer's market.

I've deliberately limited the scope of my evaluations to implementations of System V for the 386, which can run on 386 class machines. Thus I've excluded the Sun OS, for which the only 386 host is the 386i. I've excluded Minix and Venix, which are work-alikes, and Qunix, an independent development with different architecture.

New Releases

The process of developing a new release of AT&T UNIX begins with an implementation on an AT&T 3B minicomputer. Then AT&T lets porting contracts which are partially financed by the manufacturer of the particular CPU. After completion, the port must pass the SVID conformance suite.

Until recently, AT&T did not permit others to use the trademark "UNIX." Thus we have the proliferation of third party trademarks, such as "386/ix," "ESIX," etc. Finally a source tape of the port is for sale to all, typically for $100,000.

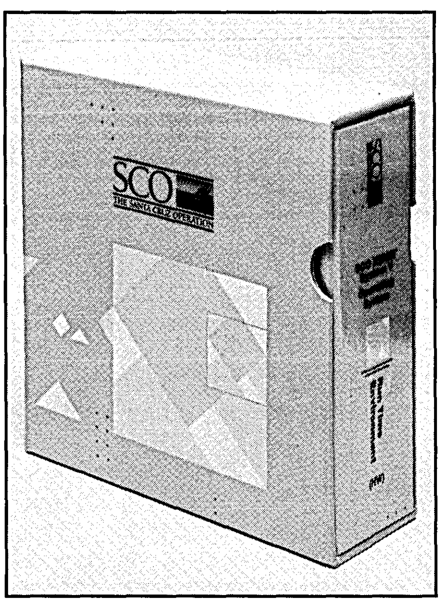
Development Threads

A workgroup at Intel did the port of System V.2 to the 286 under the project name "Microport." When Intel decided not to sell the product, a group formed under the same name to market the port.

Interactive Systems did the port of System V.3.0. Summit Computer, a joint venture of Intel and AT&T, did the System V.3.2 port.

Sun has recently appeared as the primary developer of System V.4, which will be a merge of Berkeley and AT&T UNIX.

Berkeley represents yet another thread of UNIX development. Because the Department of Defense originally financed the project, the source code is available at universities and has spawned a national treasure—a generation of brilliant hackers. The last version was 4.3, after which funding was cut.



SCO Software Package

Until recently, Berkeley had a technical edge over System V. It still does in two areas: the file system and the keyboard input. But the primary allure is the wealth of free software. We System V folks must do some serious adaptation to use this code (at least until we get V.4).

Although Berkeley has been cut, we're continuing to see a stream of new UNIX systems. For instance, Carnegie Mellon built the MACH system, used by both the NeXT workstation and the Evans and Sutherland supercomputers.

Many years ago, IBM purchased a license to System V.2 and has independently developed it into AIX (Advanced Interactive Executive). This system undeniably has many merits. For example, you can selectively swap out the kernel, or resident portion of the OS, to make memory use more efficient.

UNIX On Small Machines

Microsoft developed the first small system implementation for the PDP-11. Its name: XENIX. The system was ported to the 68000 and all the Intel processors, including the 8088.

SCO, the Santa Cruz Operation, took over much of the responsibility for maintenance and sales, and they've turned XENIX into a super system. At one point 70% of the UNIX licenses world wide were XENIX. AT&T made accommodation to that fact by merging the capabilities of XENIX into V.3.2.

Thus, current versions of the AT&T product and XENIX can run application programs written for each other. Furthermore, XENIX has passed the SVID conformance specification, and it is SCO's position that XENIX is therefore System V. This interchangeability does not extend to device drivers or programs dependent on device drivers.

By Bob Morein

Automata Design Associates
1570 Arran Way
Dresher, PA 19025
(215) 646-4894

Speaking Of UNIX Drivers

DOS has three major components: the BIOS, the BDOS, and COMMAND.COM. Under DOS you can add BIOS extensions at boot time to handle nonstandard devices.

Lowly DOS has an advantage this way. The part of UNIX that resides in memory is called the kernel. The kernel is composed entirely of device drivers and the "core." Each driver has entry points (a jump table) established by whoever wrote it.

Device drivers are a weak point when it comes to transporting UNIX applications. These drivers are supplied as linkable object code. Every time you add a device driver, you rebuild the kernel.

After the driver writer has satisfied the basic requirements, he can add calls to his new driver. Thus we have a loose class of programs which may (or may not) work when you switch between different brands of UNIX. This includes:

- Windowing systems;
- Floppy disk utilities;
- Any program which directly accesses hardware.

Furthermore, programs which access kernel data structures, such as some system utilities, are not interchangeable between XENIX and UNIX.

Drivers are not interchangeable between XENIX and the AT&T product because the kernels are not the same code. But under System V for the 386, the program "sysadm" frequently installs and updates software automatically. There are two installation formats: AT&T and Interactive Systems. Thus when purchasing software installed by "sysadm," be sure to purchase the correct format. Incidentally, Everex ESIX understands both.

XENIX uses Microsoft C, which produces an object format much like DOS, called OMF (Intel Object Module Format). AT&T established a universal, though less compact, format entitled COFF (Common Object File Format) which you can use with any processor. Both systems can execute programs in either format. But, software development tools aren't portable because the link and debugging formats are different.

> **One thing is clear. Rapid evolution assures that you will not get stuck with a lemon. Plus, the current players are extremely competent.**

Timesharing Vs. Workstations

You can put together two basic types of systems. One is a timesharing machine, to which you connect multiple serial terminals. Office automation uses this when there's no need for fancy graphics. The principal advantage of UNIX over a LAN is that you have large amounts of program memory for databases. Plus you don't have the administration problems of a LAN.

The other class is the workstation. This humble name refers to a system with a large screen running a windowing system, probably X. It's a productivity tool. With a large screen you can turn out more code in less time. Plus, you can run CAD tools, though there are precious few for System V now. A workstation seldom has more than one user, but multiple users can use it if it's networked.

Two Users/Many Users

This brings us to the two types of UNIX licenses. A logon refers to either a user logged into the machine or a network connection. The unlimited user license has no limit to the number of logons. The two-user license is considerably cheaper.

Since background communications (such as a uucp license) count as a logon, a two-user license is really a single user system. The license agreement states the two-user restriction. However, there are differences in enforcement. I know of at least one system where compliance is completely voluntary.

I use three basic criteria for selecting a system. First is performance. Second, does the system support your hardware? Some systems support mainly proprietary hardware, while others try to support everyone's. You must also consider the importance of advanced features and add-ons.

Advanced Features

MULTISCREENS: All these systems contain Multiscreens, pioneered by SCO XENIX. Each of the 8 to 12 screens is a virtual terminal which you log onto separately.

FILE SYSTEMS: File System Switch lets different disk types run simultaneously . If you have a XENIX 386 disk, it can be mounted by V.3.2. The reverse isn't true, since XENIX doesn't have FSS. All AT&T implementations offer the 1K and 2K file systems. At least one vendor plans to offer the Berkeley fast file system.

Remote File System (RFS), the AT&T authored software, allows mounting another file system over a network in a transparent way. The de facto standard in this area is NFS (Network File System), repackaged for the 386 by Lachman Associates and the Wollongong Group.

WINDOWING: MIT's X Window System is more important to UNIX than Microsoft Windows is to DOS. Programs for DOS "know" about most of the displays, and frequently bypass the BIOS. In contrast, the UNIX kernel uses the hardware memory mapping and protection of modern CPUs to isolate application programs from the hardware.

The only way to support advanced video hardware easily is via a windowing system, which provides a device-independent interface. All vendors provide an X implementation. Be sure that your video board is supported, however.

X (for extensible) can operate over networks via TCP/IP transport. But, not all implementations of X support this.

DOS ENVIRONMENTS: It's possible to run DOS under UNIX. AT&T's Simultask, Phoenix Technology's VP/IX, and Locus Dosmerge give you multiple virtual DOS machines by running the 386 as an 8086 emulator. Simultask and VP/IX run on a modified V.3.2.0 kernel. Dosmerge runs on V.3.2.2, not yet in common use.

You can hotkey between DOS and UNIX sessions. The construction of a virtual machine under UNIX is quite difficult and, until recently, the overhead was ridiculous. I've measured Norton SI performance figures of 2 on a 16 MHz 386. There has also been a tendency for DOS programs to break the system.

Under 386/ix version 2.01, I've measured overhead of only 25%, about that of Windows 386. Bear in mind that programs can only use expanded memory. You can't use extended memory because the 386 cannot run virtual 286 machines. So 286 and 386 mode programs will not work.

VP/IX supplies DOS service to serial terminal users by emulating the IBM Monochrome Adaptor. This is text-mode only, probably unacceptable except for the simplest application.

SERVER: On the other hand, you might make your UNIX machine a file server for DOS. SCO and Interactive offer this as an option. Because these servers run in 386 mode, performance should be better than any system except Novelle Netware 386.

If you run a PC, you can toggle between UNIX and DOS applications. You can also run UNIX applications from the DOS prompt, in background mode only.

While the 386/ix product provides only a UNIX server to DOS client connection, SCO XENIX-NET lets you run multiple servers and mixed server/clients. It's compatible with IBM-PC-Network and MS-NET.

How UNIX Is Packaged

Until recently, the system was split into three parts which could be purchased separately. These are the Operating System, the Software Development System (SDS), and the Documenter's Workbench.

I captured the output of the XENIX "custom" utility by using the "tee" util-

Figure 1—Typical XENIX Manifest

Name	Inst	Size	Operating System packages
ALL	Part	16346	Entire Operating System set
LINK	Yes	2138	The link kit
RTS	Yes	4690	XENIX run time system
BASE	Yes	1176	Basic extended utility set
BACKUP	Yes	310	System backup and recovery tools
SYSADM	Yes	1492	System administration tools
FILE	Yes	486	File manipulation tools
LPR	Yes	554	Multiple line printer spooler
IMAGEN	No	228	Imagen Laser Printer Support
MAIL	Yes	648	Electronic mail and local area networking
CSH	Yes	140	The C-shell
DOS	Yes	364	DOS utilities
VSH	Yes	266	The visual shell
EX	Yes	332	The ex and vi editors
UUCP	Yes	2026	Uucp and cu communications utilities
INITTAB	Yes	10	Terminal initialization
MAPCHAN	No	152	International character set mapping
TERMINF	Yes	508	Terminfo Database
HELP	Yes	498	Help utility and related files
MOUSE	Yes	142	Mouse and graphic input devices files

♦ ♦ ♦

ity. Because the XENIX custom utility manages the system software with very fine granularity, it gives you an excellent view of the composition of a UNIX system. (See Figure 1.)

The SDS, even at the highest prices, costs less than a corresponding collection of DOS tools. It includes a source code debugger (SDB), and a wealth (okay, cornucopia) of development tools. (See Figure 2.)

Since the text processing is not WYSIWYG, it has not been popular lately. AT&T recently increased the royalty fees for the package, which seems to be killing it off entirely. (See Figure 3.)

The UNIX documentation is voluminous. Although the online manual page option is available to all vendors, most have chosen not to pay the license fee to AT&T. (See Figure 4.)

The face of the UNIX user is changing, becoming less technical. In recognition of that, many new packaging schemes are appearing. In particular, some firms omit the documentation.

Prentice Hall reprints the AT&T manual set, which you can order from any bookstore. But you must take into account the $300 price. Should X or networks interest you, you'll need additional books. Particularly noteworthy are the O'Reilly books, *Xlib Programming Manual* and *Xlib Reference Manual*.

The Companies—The Santa Cruz Operation

XENIX has been around for quite some time. It's widely used for office automation, primarily due to the excellent technical support.

It does not offer support for all the advanced features of System V.3.2, such as RFS (remote file system) and File System Switch. However, XENIX has the best support for DOS. It's also the only system that has online manual pages available. (I find the manual very useful.) Benchmarks show that SCO has by far the fastest console driver.

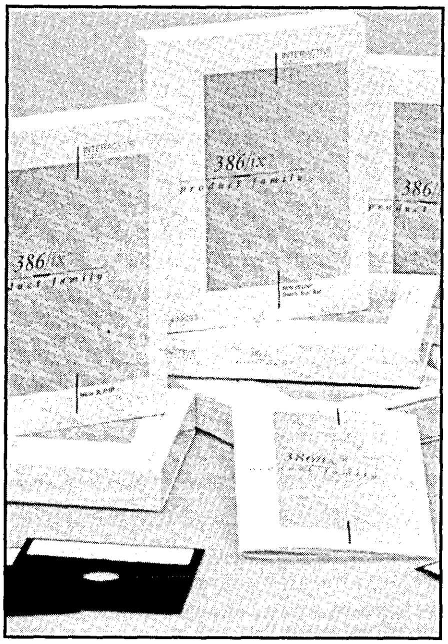
SCO will shortly introduce SCO UNIX, an implementation based upon V.3.2. They've promised that it will support DOS applications under X Windows, a feature not currently supported by any other PC-based UNIX. Sun has accomplished this with the 386i, but unfortunately their display is unsatisfactorily slow.

Interactive Systems

Recently acquired by Kodak, this

company has always been a pivotal player. Interactive executed the port of V.3.0 to the 386 and has acquired a reputation for hole-in-one performance. Nowhere is this clearer than in their X Windows port. Version 1.0 appears bugless.

The X port also features a new video board device driver interface designed to support smart boards and bit plane VGA and EGA. As a result, it supports more video boards than any other port I have seen.



Interactive Software Package

Interactive has packaged the product for the end user and the corporate market. Rather than supply the AT&T documentation for system administration, they've authored new short form books. These books are very clear, though they are not complete.

Their Software Development System, however, includes the *AT&T Programmer's Guide, Reference Manual,* and the *ISDG,* which covers device drivers and packages installable by the sysadm utility. The O'Reilly books, including the new *X Windows Users Guide,* completely document the X Windows System.

Interactive's file system is the fastest I've tested. A standard UNIX file system maintains a set of "inodes" (initial nodes), disk-based pointers to file system blocks. Interactive reads the list into RAM, where bitmap à la Berkeley replaces it. This, along with the algorithm to manipulate the bitmap, reduces file system fragmentation while retaining

Figure 2—Software Development System

Name	Inst	Size	Development System packages
ALL	Yes	14922	Entire Development System set
SOFT	Yes	8428	Basic software development tools
LEX	Yes	114	Generates programs for lexical analysis
YACC	Yes	106	Yet another compiler-compiler
CREF	Yes	466	Cross reference programs
CFLOW	Yes	114	Generates C flow graphs
LINT	Yes	418	Syntax and usage check files and tools
SMALL	Yes	508	Small Model 8086/286 Library Routines
MEDIUM	Yes	530	Medium Model 8086/286 Library Routines
COMPACT	Yes	544	Compact Model 8086/286 Library Routines
LARGE	Yes	564	Large Model 8086/286 Library Routines
SCCS	Yes	664	Source code control system
DOSDEV	Yes	2272	DOS cross development libraries and utilities
HELP	Yes	138	Help utility and related files

♦ ♦ ♦

Figure 3—Documenter's Workbench

Name	Inst	Size	Text Processing System packages
ALL	Yes	2794	Entire Text Processing set
TEXT	Yes	828	Basic text processing commands
EQN	Yes	154	Math equation formatter
MANMAC	Yes	172	Man macro package
SPELL	Yes	372	Spelling checker
NROFF	Yes	206	Nroff formatting tools and tables
TROFF	Yes	232	Troff formatting tools, tables and fonts
TBL	Yes	82	Tbl table formatter
MS	Yes	78	Ms macro package
MM	Yes	650	Mm macro package

♦ ♦ ♦

Figure 4—Documentation

Name	Inst	Size	SCO On Line Manual Pages packages
ALL	Yes	3218	Entire on line manual and utilities
MAN	Yes	116	Basic manual page utilities (required)
C	Yes	1166	Manual pages for basic XENIX commands
CP	Yes	296	Manual pages for programming commands
CT	Yes	128	Manual pages for text processing commands
DOS	Yes	128	Manual pages for DOS commands
F	Yes	128	Manual pages for file formats
HW	Yes	142	Manual pages for hardware dependent info
M	Yes	116	Manual pages for misc commands
S	Yes	694	Manual pages for system services

Name	Inst	Size	VP/ix packages
VPIX	Yes	2358	VP/ix

♦ ♦ ♦

compatibility with the System V file system data structures.

Unfortunately, perfection has a high price. The list price of the complete system (including networking) is over $3,000. No doubt corporations will find it worth the price. If you can convince Interactive that you are a UNIX developer, however, they'll knock 75% off the price.

Interactive takes DOS connectivity seriously. The PC-Interface Server works in conjunction with the DOS Bridge Module to provide a powerful server. It supports both Ethernet and RS-232.

Intel

Intel has recently acquired Bell Technologies. Bell built the first PC-based UNIX workstations (they were later enhanced with extraordinary video hardware). A Bell MPE produced the benchmark figures in this article.

VGA and EGA cards for DOS have a character mode and a graphics mode. Because these boards do not have an onboard processor, graphics mode requires the CPU to draw each character dot by dot. Character mode, on the other hand, uses a character generator so you can't display graphics.

The BLIT Express is an example of the kind of video hardware that DOS users only dream about. The screen has a raster of 1200x1664 and is capable of displaying 35,000 characters with different fonts in each window. This video system is based on the Intel 82768 blitter chip.

Running under X Windows, the BLIT dispenses with the separation of character and graphics modes. If you don't have a BLIT, this port also supports EGA, VGA, and Hercules.

Bell's UNIX does not diverge in any way from the AT&T product. Performance is excellent. The documentation philosophy is old-fashioned, simply rebound editions of the AT&T manuals.

Docs for the optional packages include *everything* (even the O'Reilly X books). The NFS and TCP/IP docs contain "deep throat" info that is a godsend for the developer, for whom it was intended. While other vendors separate developer and runtime packages for such things as TCP/IP and X Windows, this port provides (almost) everything.

You will still have to purchase a few odds and ends, such as a utility to read DOS disks, but this package is so reasonably priced, you shouldn't mind.

Everex ESIX

A newcomer to this field, Everex has, fortunately, attracted a group with a lot of Berkeley experience.

Their marketing scheme is shockingly simple: charge ¼ to ½ of what the competition charges, even though they include all the options. They've kept the packaging simple and are passing the savings on to the consumer.

Thus, instead of a myriad of different add ons, Everex has one product. It includes RFS, TCP/IP, X Windows, and the Software Development System. You get a slight discount if you don't want the SDS, but think carefully—it's not available separately.

The first release, rev A, was remarkable—it worked. I did, however, find bugs in the X implementation. (I didn't try the networking.) Rev B added networking to X, and rev C, of which I have a beta, doubles the disk throughput.

Everex is using a high-speed (Interactive) file system. Should the system find a bad sector, it attempts to recover and move the data, then adds the sector to the bad sector map.

Everex will soon add the FSS (file system switch) so they can support the Berkeley Fast File System. They say this system will be faster than Interactive's.

The FFS uses very large disk blocks, either 4K or 8K, compared to the 1K or 2K of System V. The end of each track contains an odd length block which contains "buffer fragments."

The remainder of a file smaller than the block size stores as a fragment, so the large block size does increase file size.

Documentation comes in two parts. "Minimal documentation" comes with the software. This minimum part explains how to install the system and gives you details on the current release.

The second part comes from the bookstore (i.e., the Prentice-Hall manual editions, the O'Reilly X books, etc.)

Everex is also writing its own version of the second part (probably to avoid

Figure 5—Base Package Contents

	386/ix	ESIX	Intel	XENIX	Dell
System Administration	y	y	y	y	y
Editing Package	y	y	y	y	y
Software Development System	n	y	n	n	n
Security Adminstration	y	y	y	y	y
Extended Terminal Interface	y	y	y	y	y
Remote Terminal Package	y	y	y	y	y
Form and Menu Language Interpreter	n	y	n	n	n
Framed Access Command Env.	n	y	n	n	n
2-k File System	y	y	y	na	y
Xenix File System	y	y	y	y	y
Networking Support Util.	y	y	y	y	y
Remote File Sharing (RFS)	na	y	y	na	n
TCP/IP Network Transport	n	y	n	n	n
Sendmail (a new usenet protocol)	y	y	n	n	y
X Window System	n	y	n	n	y
Network File System (NFS)	n	n	n	n	n
Documenter's Workbench	n	na	n	n	n
VP/IX DOS environment	n	na	na	n	y

♦ ♦ ♦

Figure 6 —Benchmark Summary

	Operating System			
	XENIX	386/ix	Intel	ESIX
Dhrystone2 NO REG	7923.2	7698		7698.2
Dhrystone REG	7874.0	7942	7803	7942.8
Iostone	2380	7142	5633	5402
C&L	2:50.3	2:26.7	2:43	2:49.4
L	2.8s	2.1s	2.6	2.8
cat to screen	18.7s	1:12	1:16	1:08

C&L = ESPRESSO compile and link
L = link only
cat = send all ESPRESSO source files to the screen

♦ ♦ ♦

AT&T royalties). This is an enormous project and will take some time, but when they finish the documentation should be quite a bargain.

Dell UNIX

This is an enhanced repackaging of the Interactive port. They've included some nifty XENIX utilities, notably "diskcp," for copying floppy disks. And, the Bourne Shell, the equivalent to DOS' COMMAND.COM, now shows the current path.

The packaging is spiffier than 386/ix. The binders, the paper, and the typesetting are beautiful. However, the X Window documentation is not complete since it doesn't include the O'Reilly books. This is a very reasonable package if you purchase it installed on a Dell computer. Performance is identical with 386/ix.

The Base Distribution

The "base" refers to the smallest usable package you can purchase. Each vendor has included a different mix. (See Figure 5.)

The abbreviation "na" indicates that the vendor does not sell the product. However, in most cases except XENIX, you can purchase the product from another source if the "sysadm" install format is compatible. ESIX can install software in both the AT&T and Interactive formats. Some potential incompatibility persists with X and networking at the device driver level, so you should check with the vendor.

The Intel distribution is compatible with all AT&T products since Intel's done no modification.

Conclusion

I hope this information will help you select a system. Any of these packages will give you two big advantages over DOS: multitasking and superior development tools. Switching from DOS to UNIX is a little like a cool dip in the ocean; it's great once you're in!

Editor's note: Bob also sent along a very substantial amount of benchmarking information. Though there isn't room to run it with the article, we're putting it on the issue #50 disk. A brief summary appears in Figure 6.

Suppliers Mentioned

Automata Design Associates
1570 Arran Way
Dresher, PA 19025
(215) 646-4894

Dell Computer Corporation
P.O. Box 203818
Austin, TX 78720-3818
(800) 426-5150

Everex Systems
48431 Milmont Drive
Fremont, CA 94538
(800) 821-0806

Intel Corporation
(formerly Bell Technologies)
330 Warren Avenue
Fremont, CA 94539
(800) FOR-UNIX

Interactive Systems Distribution
P.O. Box 906
Hollis, NH 03049
(800) 537-5324

The Santa Cruz Operation
400 Encinal Street
P.O. Box 1900
Santa Cruz, CA 95061
(800) 726-8649

♦♦♦

By Linda Lunt
2133 186th Place SE
Bothell, WA 98012

Life, Bliss, And Rocky Mountain SOG

I'm not sure whether your best SOG is your first or your last. I'm not even sure you can get enough SOG (I'll be attending four this year). However, there certainly isn't a SOG without people. Here's a look at the Rocky Mountain SOG from the eyes of new initiates.

"Do you want to go to the Rocky Mountain SOG with me?" "Yeah, I guess so." Karl, my husband, had recently begun writing for *Micro C* and was about to experience his first SOG.

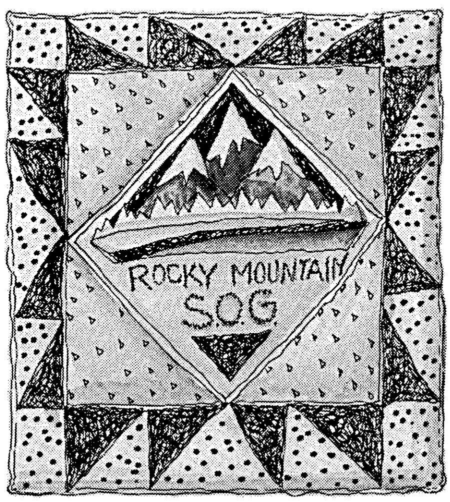
Something had been missing in my life for a long time, so we were quitting our jobs, leaving Phoenix, and moving to Seattle. Sandwiched in was the Gunnison SOG. I'd go, do a little shopping, check out the local quilt store, while Karl talked computers. Then we'd return home and move to Seattle. Finally (maybe) I would figure out what was missing in my life and get on with it.

The journey started innocently enough—14 hours on the road. The pine trees of Flagstaff gave way to the stark beauty of the Ute Mountain Indian Reservation. North of Durango, the real mountains started, cool fresh air and greenery everywhere. The awesome ruggedness, the exhilaration of 11,000-foot Red Mountain pass, and the 10 mph hairpin turns began working on my soul.

In Gunnison the following morning, I helped Maria Ladd with registration. Great fun. The participants were a happy, friendly group, each unique. Maria had arranged a craft class for the next day. Good; I would get to know some of the wives and have some fun.

That evening we went to the Cattlemen Inn for the first SIG. You know, a Special Interest Group meeting. Remember those all-night sessions in college? Add a diversity of experiences and a wide range of expertise, grounded by a common interest in computers.

> Remember those all-night sessions in college? Add a diversity of experiences and a wide range of expertise, grounded by a common interest in computers.



The next day dawned. Was it only Friday? So much had happened. After a while two of us went to the craft class. The E and P Sewing Emporium in downtown Gunnison is a wonderful place. We walked in and became part of the family. While my cohort began an appliqued shirt, I started a pillow quilt. It was an intriguing new pattern for a quilt that folds up on itself, fits into its own pocket, and becomes a pillow.

You have to understand something—I love making quilts. The combination of detail and precision, fabric color and texture, and spatial relationships—what an experience. So I played all day, completely missing the fractal presentation by Roger Stevens and the talks which followed.

That night we had the Jolt SIG at the Aspinall-Wilson Conference Center. I was finishing my pillow-quilt when Richard, one of the participants, wandered by. "Would you make one of those quilts for me?" "Yes, I suppose so."

As he went to get his checkbook, I tried to recover from shock. Someone actually wanted to buy one of my quilts. The following morning I was back at the center with fabric samples.

Larry Fogg, in his article on creativity, ("Problem Solving and Creativity" in *Micro C*, May-June 1989) talks about distracting the conscious mind long enough for the subconscious to get involved. Something was happening inside me, only I didn't realize it. I just woke up Saturday morning convinced that I could sell a quilt to every attendee.

As I showed off my quilt and began taking orders, I realized something: I'd found the solution. I'd found what was missing in my life. I had found my bliss (David J. Thompson, "Around the Bend" in *Micro C*, May-June 1989).

Now tell me, does this always happen at SOG? I wonder what we'll do next year? Anyone know how to print a full color fractal on fabric?

♦ ♦ ♦

A View From The SOG

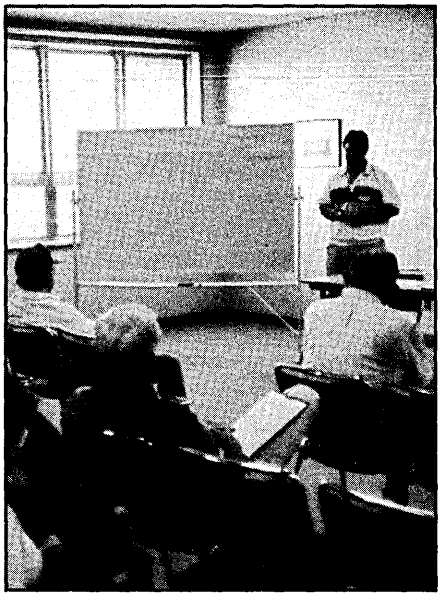
The first Rocky Mountain SOG ended yesterday. Though I sit amidst stacks of packing boxes (we're preparing for this week's move to Seattle), I have something more important to do. I must tell you about SOG.

RM SOG-I took place July 27-29 at the Aspinall-Wilson conference center in Gunnison, Colorado. Due largely to superhuman work by Scott and Maria Ladd, everyone had a great time.

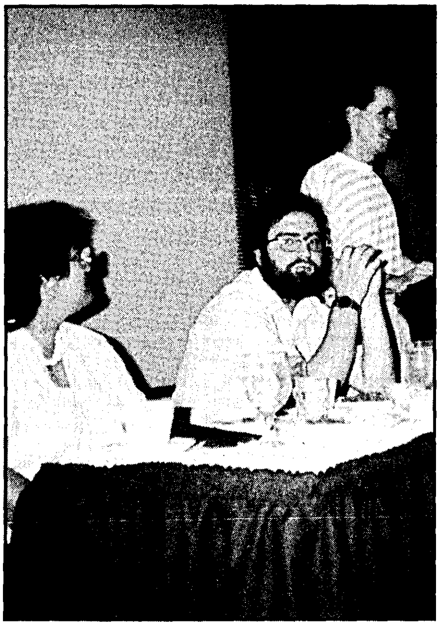
The Raft Trip

SOG opened on Thursday with a white-water raft trip. Twenty-eight SOGgers (SOGgies?) and five guides made a five-hour run down a river just outside Gunnison. During the lunch break, those who signed up for only half way caught a van back to town; the rest (23) climbed back into the inflatables.

If you have ever attended a SOG (this was my first), white-water rafting seems



Micheal Hunt's Talk on Pascal and Modula Two



Scott Ladd, RMSOG's Director

to be a necessary event. As this was also my first rafting trip, I learned the importance of packing a *full* change of clothes and taking plenty of sun block. Expect cold, wet feet for the duration.

Our river guide, John, showed a great sense of humor and lots of patience. Apparently, he doesn't normally run the river with a boatload of hackers; he didn't contribute much to the arguments over high-level languages and the jokes about FORTH and C went right past him.

Editor's note: The uninitiated usually lose the thread quickly.

He became irritated only once; the whole group got into a discussion of MAKE files, ignored his instructions, and we hung the boat on some rocks.

Although the first official session didn't begin until Friday, custom apparently demanded an immediate Jolt SIG. For the uninitiated, this technical free-for-all begins after dinner and continues until the next to the last person loses consciousness. Since the conference center lobby wasn't available that evening, about 20 of us headed for a local restaurant and started the late-nighter in the lounge. Linda and I left around 10:30, two of the first to go.

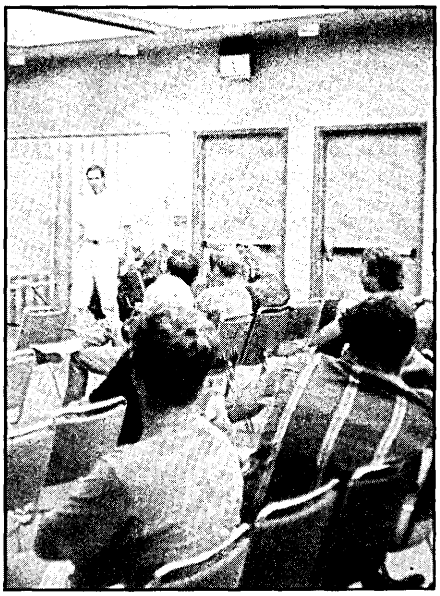
Let The Real SOG Begin

Sometime Friday morning I began to feel the magic. Roger Stevens presented "Fractals for Fun and Knowledge," a super discussion of C programs for generating Mandelbrot, Julia, Dragon, and Phoenix curves. He left a PC running a slide show of his fractals in the lobby of the conference center. The display always had a crowd around it.

I wasn't able to attend Don Jindra's opposing talk on his $25 PC-based Local Area Network, but others who did told me it was excellent. Don sold several

Karl Holds Forth at SOG Break

By Karl Lunt
2133 186th Place SE
Bothell, WA 98012



Walter Bright Lectures About Optimizing Compilers

Gary Entsminger and Larry Fogg at SOG.

copies of his program at the SOG, and it was the center of attention during Saturday evening's Jolt SIG.

Dave Thompson's talk on starting a high-tech business drew quite a crowd, as did Jim Nutt's opposing session on Fidonet technology. I opted for Dave's presentation, but with some regrets; I also wanted to hear Jim's talk. I found out later it was great.

The whole SOG continued the same theme; two opposing high-quality presentations. Often it was a difficult choice, but I seldom heard any complaints.

The Jolt SIG

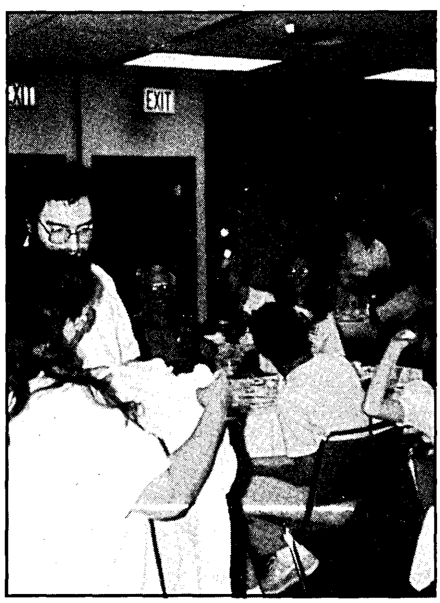
Nothing captures the essence of the SOG like the late-night sessions. Starting at an hour when "real" conferences have broken up, the Jolt SIG isn't even rolling until the caffeine kicks in. You have to be there to appreciate the decibel level generated when you mix 50 hard-core hackers with several cases of high-octane soda.

Many times during the Jolt SIGs, I found myself standing outside the crowd, simply watching. These people had traveled hundreds (sometimes thousands) of miles, at considerable expense, to spend their nights amidst pure Pandemonium.

One group would be talking about the latest C compiler, overhear another gathering talking about optimizers, and suddenly the two groups would merge. Like a low-resolution Game of Life, the crowds formed and reformed seemingly at random, always seeking new energy and ideas.

The People

Those who attended RM SOG-I came from two general groups. The old-timers knew what the SOG meant, and understood the energy generated by a three-day gathering of hard-core hackers. Others (like myself) came out of curiosity and quickly found that this was something special.



Scott gets marshmallowed!

I have gone to conferences sponsored by other organizations; most are mind-bendingly dull, presided over by a distant god selected by an elite few. I usually learned little and consider most of them a waste of time.

But a SOG marches to a very different beat. The interplay, the exchange of information and ideas, occurs constantly and on a very high level. It does so without regard to status or station.

I know of at least two Ph.D.s who attended RM SOG-I; without knowing in advance who they were, you could not have picked them out. All were equals, sharing freely.

It wasn't until late Friday that I realized something unique about this crowd; alcohol was not part of the scene. Even at the Thursday Jolt SIG, held in a restaurant lounge, the interest was more on talking than drinking. Apparently, many felt booze just got in the way of the energy; this also was different from other conferences, where drinking relieved the boredom.

The Individuals

Many people stood out from the crowd for one reason or another; they added to the flavor of the SOG through their unique personality.

Chief among these was Maria Ladd. I don't know where she got her energy; she seemed to be going constantly. Organizing road trips, running registration, setting up the Jolt SIGs, doing errands, making posters...it exhausted me just to watch her. I know of at least one occasion when Maria went all night without sleep.

RM SOG-I gave me my first chance to meet the Boys from Bend. I had talked often with Dave Thompson and Larry Fogg over the phone, but hadn't met them (or Gary Entsminger) until the SOG. It's hard to write anything serious about these guys, knowing that at least two of them will edit it before it gets printed.

I will only say that the panel presentation they gave on the direction of *Micro C* drastically changed my concept of "magazine editor."

Hey Dave, what's he saying here, exactly?—Larry

I'm not sure. I suppose we can leave it in until we figure it out.—Dave

Of all the people I met at the SOG, none affected me more than Christy Quinn, Debee Norling, and Debee's seeing-eye dog, Duchess. This triafeminate (an old Latin word I made up) brought me face to face with many misconceptions I had about "handicapped" people.

Subconsciously, I had always felt sorry for the blind; I compared them to sighted people and felt somehow obliged to pity them. Let me tell you, three days with that crew forced some hard second thoughts.

For instance, seeing Debee in the white-water rafting group Thursday morning caught me by surprise. I didn't expect her to go rafting and I certainly didn't expect her to take Duchess. But with Debee going, I certainly expected her sighted compatriot, Christy, to be in the boat. Wrong, bucko; Christy informed me that she was not crazy enough to go rafting, but that Debee went every chance she could.

Then there was the time I saw Debee setting up her talking Toshiba laptop at one of the Jolt SIGs. Feeling like I ought to help, I started arranging the cords only to receive a firm "thank you!" Taking the hint, I stepped back and watched her quickly put the machine together.

Unfortunately, I could not attend Debee's session on TSRs in C, but I quickly realized that Debee did not fit my stereotype of the blind. Together she and Christy have run a successful consulting firm for several years. They've based the business on Debee's software talents and Christy's hardware expertise.

The Sights And Sounds Of The SOG

So much of RM SOG-I consisted of details. I remember talking with Michael Hunt, who writes the *Micro C* column "Units and Modules." He wanted to tell me about a project he and Dave Thompson had cooked up. Michael was so excited about the idea that he went an entire minute without getting both feet on the ground at the same time.

At the Saturday lunch break, about 15 of us headed off to the cafeteria. We got so caught up in our conversations that we ended up lost in the middle of the campus.

I also remember standing in the middle of the Saturday Jolt SIG, watching all the activity, and noticed Scott Hurlbert next to me doing the same thing. Scott arrived from Anchorage, Alaska, for his first SOG; I don't remember ever seeing him without an L.A. Laker cap or T-shirt. Scott kept staring at the crowd and murmuring, "This is incredible, this is just incredible!"

The Friday Jolt SIG began without Tadas Osmolskis (from Maryland). Half an hour later, Tadas and several others arrived, triumphantly carrying Gunnison's entire supply of Jolt cola (three six-packs). They dumped these into a tub of ice (supplied by Maria Ladd) and the SIG was official.

Linda and I sat at the front table of the Epicurean restaurant on Saturday morning, eating a delicious breakfast of homemade sausage and ebelskiver (small baked pastries topped with apple chunks). At a table behind us, Rick Hollinbeck quietly prepared notes for an impromptu session he had volunteered to do as a replacement for a flu-stricken Scott Ladd. He and his wife, Marty, run a software company called Western Wares; I understand his presentation went well.

I set up Bertha, my 68000 system, at the Friday Jolt SIG. It wasn't long before Debee asked to see how SK*DOS worked. We cabled her to Bertha and soon her talking Toshiba started calling out SK*DOS commands. When she asked to see (listen to) some 68000 assembly language code, I told her how to access my files; soon, she was hearing the source for my multitasking kernel.

Joseph Chui and his brother, Kenneth, showed up at the same SIG, arguing what the odds were that a computer program could correctly choose the rank of a randomly-selected playing card. I left Joseph with Bertha for a while; soon, he and Kenneth had the program running and were close to determining the odds (by observation).

Conclusion

This event changed everyone who attended. The old-timers got their annual fix, but the first-timers—we felt the magic most of all. We came out of curiosity and left ready to get on with new projects. Each of us plan to attend RM SOG-II, and to bring something more to contribute to the energy.

For me, the SOG crystallized into one special moment. I stood in the lobby of the conference center, well into the Saturday Jolt SIG. At a group of tables across the room, Debee and Don Jindra had set up a couple of PCs and Debee's talking Toshiba, forming a three-system LAN. Debee sat on the floor, her arms nearly over her head to reach the keyboard on the table in front of her. Duchess lay curled around her, an island of calm within the buzz.

When I close my eyes I can still see that picture. Somehow, the confusion and serenity, the darkness and swirling action, the concentrated silence and vocal uproar, all came together at that instant.

It has only been two days since SOG, but already some of the memories aren't as vivid. I recall meeting someone named Carl who wanted to talk with me about surplus 68000 systems. I asked him to find me during the Jolt SIG that night, but I don't believe I saw him again. I was drained, exhausted from long days and little sleep. I didn't get his information, but I appreciated his enthusiasm.

Scott and Maria announced at the final breakfast that RM SOG-II will take place mid-June, 1990.

I can't wait.

♦ ♦ ♦

Capturing & Graphing A Voice In Real-Time: Part 1

Analog Signal-Conditioning

This isn't just an analog project, but here's the analog portion: as complete a discussion of analog filter design as you'll see in any magazine.

Last issue, I looked at several commercially available I/O boards for the PC. All those boards included analog-to-digital conversion and usually some kind of digital I/O and counter/timers. One had a D/A converter as well.

This issue, I'll use one of the boards (the Real-Time Devices AD1000) and step through a complete interfacing project. The project will capture a voice and display it on the screen using Turbo C and the Borland Graphics Interface (BGI), which works with any common display adapter.

Although I'm using the Real-Time Devices AD1000, you can also use the Advantech PCL-812 or 712E board from Rapid Systems or Halted Specialties for this project. To breadboard the amplifier/filter circuit and connect it to the A/D board, I use Real-Time Device's XB40 prototype board and cable (the Advantech has a similar prototype kit).

The Circuit

It's one thing to understand a circuit; it's something else to design one. My friend Brinkley Barr breathes electronics. This circuit took him about five minutes to design; it would have taken me days. When you're trying to accomplish something, it makes a lot of sense to go to an expert. You pay for the expert's experience, but you'd pay for the experience one way or another.

Figure 1 shows the voice signal-conditioning circuit. It consists of a microphone, a two-stage amplifier, and a 3-pole anti-aliasing filter to remove frequencies higher than 6 KHz (most of the information in your voice is below 6 KHz). Aliasing occurs when the frequency of your input hits half the sampling rate of your A/D converter—the converter doesn't get enough information about the waveform, so the information is garbled.

The circuit's three op-amps reside on a single chip, so the parts count is quite low: the chip, a microphone, and some resistors, diodes, and capacitors. All the parts are inexpensive and easy to mail-order (details later).

Let's walk through each section of the circuit so you understand what's going on.

The Microphone

Several different types of microphones are available: condenser, crystal, magnetic, ceramic, etc. Brink selected an electret microphone because it's simple (it doesn't need a high-voltage bias like condenser microphones).

You'll be amazed at how tiny these little guys are. They look like spy microphones, but they contain a field-effect transistor (FET) so the signal is already boosted by the time it gets to your amplifier.

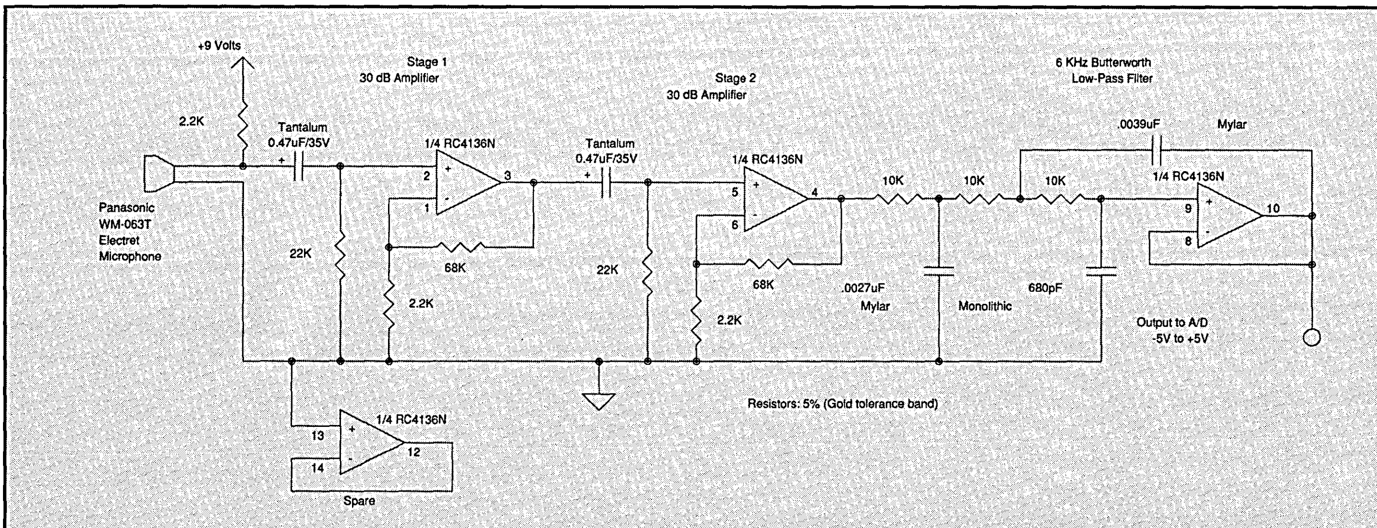


Figure 1—Microphone Preamplifier and Anti-Aliasing Filter

By Bruce Eckel

Revolution2
308 Meredith St.
Kennett Square, PA 19348
BIX: beckel
Compuserve: 72072,3256

The Amplifier

The amplification stage consists of two identical sections. The input to each stage passes through a high-pass filter consisting of a series 0.47 µF capacitor and a 22K resistor to ground.

This serves two purposes—it removes 60 Hz hum and it keeps DC (called an offset) from reaching the inputs of the first and second stages. The output of any amplifier contains DC, which, if passed on to the next stage, is magnified along with the signal. Which isn't what we want.

The amplifier stages are simply op-amps. The 68K and 2.2K resistors provide a gain of 30 for each stage. The gain for this op-amp configuration is calculated as 1+(68K/2.2K). The combined gain (calculated by multiplying the individual gains) is 900.

I've written about op-amps in my first book, *Computer Interfacing with Pascal & C* (available from Micro Cornucopia). *Electric Circuit Analysis* by Johnson, Hilburn, and Johnson is also worth checking out.

You may wonder why the amplifier is in two stages instead of just one (i.e., why not just change the values of the 68K and 2.2K resistors to create a gain of 1000 with a single stage?) Good question. Answer: because op-amps are introduced to novices as "ideal components," and in many ways they *are* ideal, but this circuit displays one of their limitations.

Gain-Bandwidth Product

If you're carrying a heavy weight, you can't run fast. If you force op-amps to provide a lot of gain, their frequency output (bandwidth) becomes limited. The relationship between gain and bandwidth is called the gain-bandwidth product.

If we ask for a gain of 1000 out of a single 4136 stage, we'll limit its output to 1 KHz, maximum. But our filter cuts off at 6 KHz. By limiting the gain of each stage to 30, we get a bandwidth of about 30 KHz. The amplifiers will pass all the relevant information, and the filter will do the work of removing the high-frequency components.

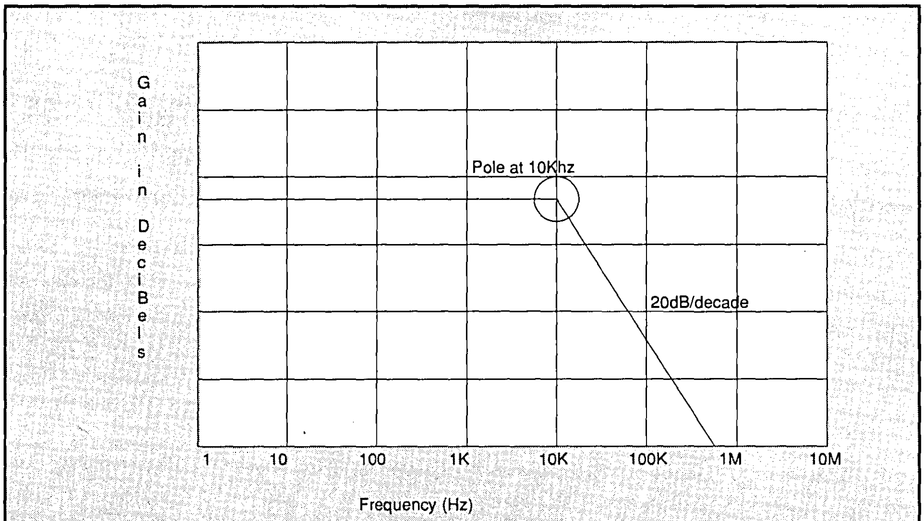


Figure 2— Bode Frequency Plot

Filters

Usually, we describe a filter, or the filtering properties of any electronic device (even if it isn't specifically designed to be a filter), with a frequency plot. A frequency plot (also called "Bode plot") shows the size (amplitude) of the signal (usually as a ratio of output size to input size) on the vertical axis and its frequency in a logarithmic scale on the horizontal axis.

In a Bode plot, the vertical axis is 20*log(base 10) of the output amplitude over the input amplitude (i.e., the gain in deciBels). Although there's an interesting history to these numbers, you shouldn't worry too much about why you use 20*log(base 10)(output/input) instead of just (output/input).

Figure 2 shows an example Bode plot, similar to what you'll see in electronic data books when you're looking for parts. Where the line is straight, signals of that frequency (shown on the x-axis) pass uniformly. Where the line bends, signals fall off. Figure 2 shows the signals passing up to 10 KHz, where the bend begins.

You can create any waveform by combining simple sine waves of different amplitudes and frequencies (we call this Fourier analysis). Thus, your voice contains components of 20 Hz, 500 Hz, 5 KHz, etc.

If your filter has a bend at 2.5 KHz and you pass a voice signal through it, all the frequency components of your voice above 2.5 KHz will be reduced or eliminated. There might still be enough information in the lower frequencies for you to hear the words, but they'll sound different.

The straight-line part of the plot (called the passband) shows which frequencies pass. After the bend, you enter the stopband, because the filter doesn't pass the signals anymore. The sharpness of the bend shows how quickly the filter switches from passing signals to stopping them.

Here the poles of the filter come in—the poles determine how quickly the filter switches from passband to stopband. The more poles at the cutoff frequency (10 KHz in Figure 2), the sharper the cutoff.

The quality factor Q, which you'll see mentioned in the databooks, measures the sharpness of a filter. The ideal filter would be a perfect step transition between passband and stopband. If you're wondering why you can't just add a bunch of poles until this happens, it turns out to be a multi-way tradeoff.

First, more poles require more components, which costs money. If you don't need a 12-pole filter, don't buy one. Also, adding poles can cause other side-effects, like "rippling" in the passband (which causes distortion). Thus, you won't often see a filter that is more than 4 poles (the one used in this project has 3).

Why Call Them Poles?

To design a filter, you represent it mathematically. The mathematical equation for a filter has a numerator and denominator, which have points passing through zero. When the denominator goes to zero, the expression becomes infinite. If you look at the equation in a three-dimensional space, a pole looks like a tent-pole under the plane (the canopy) representing the function.

Each pole of the function causes the Bode plot to bend and start decreasing at 20 deciBels (dB) per decade (change in the frequency by 10 times). A deciBel is the unit on the vertical axis of the Bode plot equal to 20*log(base 10) of the output/input. If you add a pole at the same cutoff frequency, the Bode plot will decrease at 40 dB/decade; a third pole will cause a decrease of 60 dB/decade (which is what our filter does at 6 KHz).

Zeros of the function (points where the numerator goes to zero) have the opposite effect—they make the Bode plot *increase* by 20 dB/decade. Thus, designing a filter is a matter of figuring out how to manipulate the poles and zeros of the filter function to generate the right Bode plot.

There are several ways to manipulate these poles and zeros, and these ways are named after their inventors. In this circuit, I used Mr. Butterworth's method ($5 says Dave inserts a comment about syrup here).

Editor's note: $10 says I don't. Just proves you can't sweet-talk me into adding editorial comments to your articles.

Usually you don't have to think about poles and zeros, since most reference books have the equations worked out for you; all you need to know is the cutoff and the quality factor you want.

If you want to know more about filters, an excellent reference is *An Introduction to Filter Theory*, by David Johnson, Prentice-Hall, 1976.

Power Supply

Figure 3 shows the power supply circuit. To achieve a voltage swing between -5 V and +5 V at the input of the A/D converter, we need to reduce the ±12 volt supplies brought out from the PC on the XB40 prototype board. To do this, use a 9.1 V zener diode on each line along with an electrolytic capacitor to smooth out the noise caused by the diode.

The diode will turn on whenever the voltage at its cathode exceeds the voltage at its anode by 9.1 V. Thus, the top diode will maintain its cathode at +9.1 volts, and the bottom diode will maintain its anode at -9.1 volts.

The 150 ohm resistors in Figure 3 are essential. Without them, the zener diodes would suck current until they fried something. These resistors also determine the maximum current the circuit will provide:

```
(12V - 9V)/150 Ohms = 600 milliAmps
```

which is more than enough.

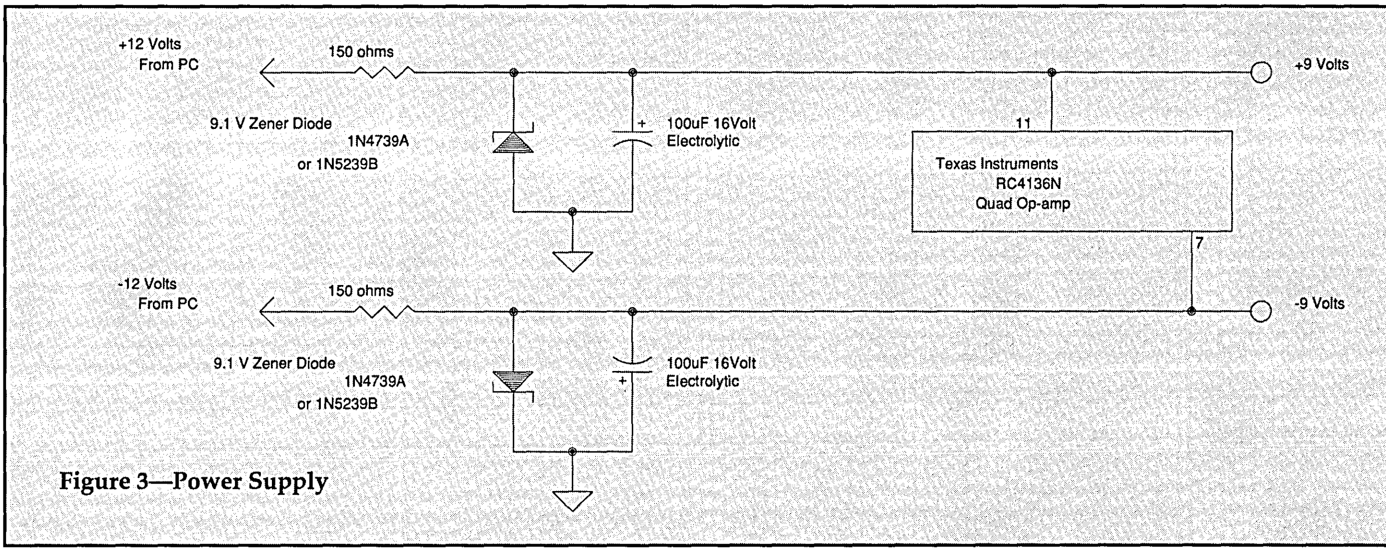


Figure 3—Power Supply

Buying Parts

I needed to shop at two outlets to get the parts. Digi-key has the electret microphone and the 1N4739A zener diodes. (Digi-key also has an extensive line of resistors and capacitors. If you do projects like this a lot, you should look into their prepackaged assortments.) You can get their catalog by calling (800) 344-4539.

JDR Microdevices ((800) 538-5000) and Jameco ((415) 592-8097) both have the RC4136N quad op-amp (four amplifiers on a chip).

Types Of Capacitors

I ordered parts (from Digi-key) before I realized I didn't have the right capacitors in my parts cabinets. JDR (where I bought the op-amp) didn't have a great selection of capacitors. I asked Brink if I couldn't substitute ceramic disk capacitors for some of the values I couldn't find in tantalum or monolithic.

He said that ceramic disk caps are the scuzziest type around and should only be used as bypass caps. Their values can change by as much as 80%. Once, as an experiment, he hooked one up in a filter circuit connected to an oscilloscope. He could squeeze the capacitor with his fingers and see a radical change!

There are many types of capacitors: electrolytic, tantalum, monolithic, silver mica, ceramic disk, polyester, polypropylene, metalized film, etc. They vary in production cost and properties. Electrolytic capacitors, for example, come in large values and some can handle high voltages (they're also physically large), so they're ideal for smoothing the ripples in power supplies. But they're polarized; you get the plus lead hooked to the minus side of the supply, and poof!

Ceramic disk capacitors are cheap and good for bypassing chips (routing supply line noise to ground). But their capacitance isn't stable (they drift with time and temperature) so you can't use them in critical applications, such as filters.

Silver mica capacitors (they *sound* expensive, don't they) are pricey but they don't drift. JDR didn't have .0027 µF or .0039 µF in mylar, nor 680 pF in monolithic, but Brink said silver mica would work just fine. JDR didn't have the right ones in stock, so I got smaller values and paralleled them to generate approximately the right values.

Editor's note: Ceramic and electrolytic capacitors are relatively cheap. However both are unstable, capacitance-wise, and considered lossy. That means that some DC current leaks through the insulation between the plates. Electrolytics also have a fairly high impedance so they don't pass high frequencies.

If you need a high capacitance (>1 µF) polarized capacitor that can pass high frequencies, use a tantalum capacitor. If you need a capacitor for frequency (e.g., filter) applications, you can use just about anything other than a ceramic, electrolytic, or tantalum.

Next Time

Next issue I'll show you the code to make this all work. We'll display graphics in windows, allocate data dynamically, and talk to the A/D board.

♦ ♦ ♦

Products Mentioned

Prom Kit - $179
Annabooks
12145 Alta Carmel Ct., Suite 250-262
San Diego, CA 92128
(619) 271-9526

AD1000 - $295
Real-Time Devices
P.O. Box 906
State College, PA 16804
(814) 234-8087 FAX 234-6864

Advantech PCL-812 or 712E
Rapid Systems
Seattle, WA 98103
(206) 547-8311
or
Halted Specialties
3500 Ryder Street
Santa Clara, CA 95051

Schematics prepared with OrCAD SDT
OrCAD Systems Corp.
1049 SW Baseline Street, Suite 500
Hillsboro, OR 97123
(503) 640-9007

Digi-Key
P.O. Box 677
Thief River Falls, MN 56701
(800) 344-4539

JDR Microdevices
2233 Branham Lane
San Jose, CA 95124
(800) 538-5000

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
(415) 592-8097 FAX 592-2503

C'ING CLEARLY

Tuning CRITTERS

By Scott Robert Ladd
705 West Virginia
Gunnison, CO 81230
(303) 641-6438

Scott recovers from SOG just soon enough to dash out this issue's column. Stay tuned as he discusses: Bull, CRITTERS, and a graphics character generator.

It's over—but it isn't. Rocky Mountain SOG. It was a big success. We didn't do it alone; I'd like to thank everyone who helped us with making RM SOG a success. Special thanks go to Jim Nutt and Linda Lunt for their above-the-call-of-duty assistance.

The raft trip was, as always, an adventure. I kept losing my paddle; but then again, I've never had my oars in the water. One boat lost its guide; another boat was swamped by high waves. The Taylor River provided one of the most exciting raft trips of my life. Fortunately, I remembered to bring dry shoes for the ride home this year.

Will we hold another SOG? You bet! Maria and I can't conceive of a SOGless summer. So, we're scheduled again for 1990: June 14-16, here in Gunnison. The theme of Rocky Mountain SOG-II will be graphics and animation, with (we hope) a special focus on robotics. If you couldn't come this year, maybe you can try it next year. There's nothing like cool mountain air and friendly people.

In fact, because of Rocky Mountain SOG, I'm going to make a change. From now on, I'm releasing all program code published in this column into the public domain. Copyrighting it is both a waste of time and a nuisance. So go to it, folks—this code's for you.

I'm writing this column during the first days of August, just a few days after our SOG. While SOG was great fun, it was also a considerable amount of work for Maria and me.

So, things that should have been done for this column weren't and I'm making a minor course correction. While the fractal landscape generator is almost complete, it isn't finished. Look for it in issue #51, along with a speedy auto-sensing graphics library.

Fortunately, I received a lot of questions and comments about the column. So this is going to be a potpourri of answers. We'll begin by looking at some new hardware that wandered in the door. (Fast hardware can be a bane as well as a boon.)

Bull

Bull is the name of my latest computer. It's from BFM Computing of Rhinelander, Wisconsin. A 20 MHz 80386 with 4 megabytes of memory, a 64K static RAM cache, 28 millisecond 65 megabyte Mitsubishi hard drive, and 16-bit VGA, Bull certainly has improved my computing environment. He benchmarks out at 23.0 on the Norton SI scale, considerably better than my "old" 16 MHz 386 box.

BFM offers an amazing 3-year warrantee on the motherboard, and so far, this machine is impressive. I don't want this to be an advertisement, but if you'd like to contact the company, you can reach them at:

BFM Computing
701 Washington St.
Rhinelander, WI 54501
(715) 362-4712

Inefficient Code

High-powered hardware is great but those of us who write programs may find that our super machines are hiding a problem: inefficient code. Even a sloppy program will be fast on a 386. However, put those programs on a lowly 8088, V-20, or even 80286, and they crawl. Such is the case with the CRITTERS program introduced last issue.

I've rewritten CRITTERS. The self-extracting file CRITTER4.COM contains everything you need. You can find it on the Micro C BBS, on the listing disk for this issue ($6 plus $2 S/H to foreign countries) from Micro C, or from me for a disk and postage. It's just too big to publish in the magazine. If you get a copy, feel free to distribute it via BBSs and users groups.

I had several goals when I rewrote CRITTERS. First, I wanted it to be faster—much faster. Second, I needed to make the program

High-powered hardware is great but those of us who write programs may find that our super machines are hiding a problem: inefficient code.

more generic so people with non-Microsoft compilers could compile it. Third, I wanted to add some new features. I broke out my tool kit and got to work.

Profilers are among the least used and most ignored tools for programmers. A profiler analyzes your program's execution, creating a report showing how much time it spends in which functions. It can even track variable accesses to tell you which data gets used most.

I use the MMC AD Systems profiler called PMon. It works very simply. You compile your program to create a .MAP file, which is then fed to a processing program called MapVar. Then, PMon works as a shell to run your program; at regular intervals, PMon checks to see where your program is using the output from MapVar to locate different parts of your program.

When your program is done, PMon outputs a report which tells how many times each function in your program was "hit." The more hits, the more often that section was executed.

Figure 1—CSG (Character Set Generator)

```
/*   Language:   Zortech C v1.07e
     Purpose:    Creates bit-mapped character sets for use in graphics
                 programs. 256 characters per set are available. */

#include "disp.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

typedef unsigned char CHAR_DEF[8];  /* matrix to hold one character */
typedef CHAR_DEF CHAR_SET[256];      /* array of all chars in a set */

CHAR_DEF grid_char;         /* create working character set variables */
CHAR_SET cset;
char setname[64]; /* variable to hold name of current character set */
const char * version = "1.10";

/* function prototypes */
void main(int argc, char *argv[]);
void show_screen(void);               /* display screen */
int  edit_set(void);        /* modify current char set */
void norm_block(int l, int c); /* set block to "off" */
void blink_block(int l, int c);              /* cursor */
void reverse_block(int l, int c); /*set block to "on"*/
void save_char(unsigned char);  /* write char to set */
unsigned char select_char(void);/*choose char to edit*/

void main(int argc, char *argv[])
    {
    FILE *csfile; /* character set file */
    char reply;   /* user input */
    int i, l;     /* loop variables */
    int save_set;

    printf("\nCSG (Char Set Generator) v%s %s %s\n",
             version,__DATE__,__TIME__);
    printf("Written by Scott Robert Ladd.\n\n");
    if (argc < 2) {
        printf("Enter a character set name: ");
        gets(setname); putchar('\n');
        if (!strlen(setname)) {
            printf("\7No set name entered. Exit.\n");
            exit(1);
            }
        }
    else
        strcpy(setname,argv[1]);
    if (NULL == (csfile = fopen(setname,"rb"))) {
        printf("Char set %s not found. Make it (Y/N)?",
               setname);
        while (!kbhit()) ; reply = (char)getche();
        reply = toupper(reply); putchar('\n');
        if (reply == 'Y')
            for (i = 0; i < 256; ++ i)
                for (l = 0; l < 8; ++l)
                    cset[i][l] = 0;
        else {
```

Continued on page 60

```
            printf("Program terminated.\n"); exit(2);
            }
        }
    else
        fread(cset,sizeof(cset),1,csfile);
    fclose(csfile); disp_open();
    disp_setattr(7); show_screen();
    save_set = edit_set(); disp_move(0,0);
    disp_eeop(); disp_close();
    disp_move(0,0);
    if (save_set)
        if (NULL == (csfile = fopen(setname,"w+b"))) {
            printf("Char set %s cannot be written.\
                    Saving as CHARSET.DAT.\n");
            csfile = fopen("CHARSET.DAT","w+b");
            }
        else
            fwrite(cset,sizeof(cset),1,csfile);
    fclose(csfile);
    }

void show_screen()
    {
    int i;

    disp_move(0,0); disp_eeop();
    disp_move(1,1); disp_printf("");
    disp_move(17,1); disp_printf("");
    for (i = 2; i < 17; i = i + 2) {
        disp_move(i,1);
        disp_printf("                    ");
        }
    for (i = 3; i < 16; i = i + 2) {
        disp_move(i,1);
        disp_printf("");
        }
    disp_move(3,31);
    disp_printf("Cursor Keys move cursor");
    disp_move(5,31);
    disp_printf("X = Exit and Save");
    disp_move(6,31);
    disp_printf("Q = Quit (without Saving)");
    disp_move(8,31);
    disp_printf("Space Bar = Reverse Current Block");
    disp_move(10,31);
    disp_printf("S = Select Character to Edit");
    }

int edit_set()
    {
    unsigned char key, fkey, curch;
    int stop = 0; int l, c; int ret_val;

    l = 0; c = 0;
    curch = select_char();
    blink_block(l,c);
    while (!stop) {
        while (!kbhit()) ;
        key = (char)getch();
        switch(toupper(key)) {
            case 0   :
                if (kbhit()) {
                    fkey = (char)getch();
                    switch (fkey) {
                        case 72 : /* UP */
                            if (l > 0) {
                                norm_block(l,c);
                                --l;
                                blink_block(l,c);
                                }
                            break;
                        case 75 : /* LEFT */
                            if (c > 0) {
                                norm_block(l,c);
                                --c;
                                blink_block(l,c);
                                }
                            break;
                        case 77 : /* RIGHT */
                            if (c < 7) {
                                norm_block(l,c);
                                ++c;
                                blink_block(l,c);
                                }
                            break;
                        case 80 : /* DOWN */
                            if (l < 7) {
                                norm_block(l,c);
                                ++l;
                                blink_block(l,c);
                                }
                            break;
                    }
                }
                break;
            case 'X' :
                save_char(curch);
                stop = 1; ret_val = 1;
                break;
            case 'Q' :
                stop = 1; ret_val = 0;
                break;
            case 32  :
                reverse_block(l,c);
                break;
            case 'S' :
                save_char(curch);
                curch = select_char();
                l = 0; c = 0;
                blink_block(l,c);
                break;
            }
        }
    return ret_val;
    }

void norm_block(int l, int c)
    {
    unsigned int val, cl, cc;

    cl = (l * 2) + 2; cc = (c * 3) + 2;
    val = disp_peekw(cl,cc);
    val = val & 0x7FFF;
    disp_pokew(cl,cc,val); ++cc;
    val = disp_peekw(cl,cc);
    val = val & 0x7FFF;
    disp_pokew(cl,cc,val);
    }

void blink_block(int l,int c)
    {
    unsigned int val, cl, cc;

    cl = (l * 2) + 2; cc = (c * 3) + 2;
    val = disp_peekw(cl,cc);
    val = val | 0x8000;
    disp_pokew(cl,cc,val); ++cc;
    val = disp_peekw(cl,cc);
    val = val | 0x8000;
    disp_pokew(cl,cc,val);
    }

void reverse_block(int l, int c)
    {
    unsigned char bit;
    int cl, cc;

    cl = (l * 2) + 2; cc = (c * 3) + 2;
    bit = 1 << (char)c;
    if (grid_char[l] & bit) {
        grid_char[l] &= ~bit;
        disp_pokew(cl,cc,0x87B0);
        disp_pokew(cl,cc+1,0x87B0);
        }
    else {
        grid_char[l] |= bit;
        disp_pokew(cl,cc,0x87DB);
        disp_pokew(cl,cc+1,0x87DB);
        }
    }

void save_char(unsigned char no)
    {
    memcpy(cset[no],grid_char,sizeof(grid_char));
```

Continued on page 62

```
    }

unsigned char select_char(void)
    {
    unsigned char sel, bit;
    int l, c, cl, cc;

    disp_move(19,31);
    disp_printf("Enter a character to be defined: ");
    disp_move(19,64);
    while(!kbhit()) ;
    sel = (char)getch();
    disp_move(19,31);
    disp_printf("                                          ");
    memcpy(grid_char,cset[sel],sizeof(grid_char));
    for (l = 0; l < 8; ++l)
        for (c = 0; c < 8; ++c) {
            cl = (l * 2) + 2; cc = (c * 3) + 2;
            bit = 1 << (char)c;
            if (grid_char[l] & bit) {
                disp_pokew(cl,cc,0x07DB);
                disp_pokew(cl,cc+1,0x07DB);
                }
              else {
                disp_pokew(cl,cc,0x07B0);
                disp_pokew(cl,cc+1,0x07B0);
                }
            }
    return sel;
    }
```

♦ ♦ ♦

Figure 2—CS_Disp (Character Set Display Routines)

```
/*
     Version:     1.10      11-Aug-1989
     Language:    Zortech C v1.07e
     Environ:     MS-DOS, IBM-PC compatible
     Purpose:     Load and display character sets
                  created with CSG.
     Written by:  Scott Robert Ladd
*/

#include "cs_disp.h"
#include "zipgraph.h"
#include "stdio.h"

typedef unsigned char CHAR_DEF[8];
typedef CHAR_DEF CHAR_SET[256];

CHAR_SET cset;

int cs_load(char * cset_name)
    {
    FILE * cset_file;

    cset_file = fopen(cset_name,"r");

    if (cset_file == NULL)
        return 1;

    fread(cset, sizeof(CHAR_SET), 1, cset_file);

    fclose(cset_file);

    return 0;
    }

void cs_putch(int line, int col, unsigned char ch,
       unsigned char fcolor, unsigned char bcolor)
    {
    int x, y;
    unsigned char bit;

    for (x = 0; x < 8; ++x)
        {
        for (y = 0; y < 8; ++y)
            {
            bit = (unsigned char) (1 << y);

            if (cset[ch][x] & bit)
                plotpixel(col+y,line+x,fcolor);
            else
                plotpixel(col+y,line+x,bcolor);
            }
        }
    }

void cs_puts(int line, int col, char * str,
       unsigned char fcolor, unsigned char bcolor)
    {
    char * ch;

    ch = str;

    while (ch)
        {
        cs_putch(line, col, *ch, fcolor, bcolor);
        ++ch;
        col += 8;
        }
    }
```

♦ ♦ ♦

It turned out that CRITTERS spent 80% of its time in Microsoft's graphics routines! Even worse, the two routines using 60% of the program's time were setting and resetting the EGA card! Yuck! This becomes even more significant when you realize that Microsoft's graphics library is faster than most of the others (including Borland's).

The obvious solution is to get a faster graphics library. While I have several libraries around the house, I decided to roll my own.

So I wrote Fast_EGA and Fast_HGC, two small assembly language graphics modules for EGA and Hercules Graphics cards. These modules do three things: turn on graphics, turn off graphics, and plot pixels. That's all that CRITTERS needs. The CRITTER4.COM file includes both modules.

Many people purchase commercial libraries rather than write their own (usually because the commercial versions are already debugged), but there are significant advantages to building your own. First, your function libraries are yours (i.e., you know them from top to bottom, and they work exactly the way you want them to). Most importantly, you can tailor your library, as I have done here, to fit the application.

Commercial packages are also designed to handle a variety of situations, so they tend to be large. You end up linking-in all sorts of things you don't need. And, when performance is an issue, it's best to build your own.

Another problem reared its ugly head when I started to use Hercules graphics. The CRITTERS display has a short status line at the bottom, indicating the number of moves elapsed and the current number of living critters. The IBM-PC ROM BIOS can display characters on a graphics screen (albeit slowly)—but the Hercules card doesn't use the IBM BIOS!

In order to write characters to the screen on a Hercules adapter, you have to put the characters there pixel-by-pixel. So I wrote a quick program to generate 8x8 bit-mapped character sets and a set of functions to display them. This turned out to be a good thing, since my bit-mapped characters are displayed many times faster than those put out by the BIOS. The C EXPLORATIONS section below describes the character set generator and display functions.

The original version of CRITTERS read screen pixels to determine if they were food. This is slow and tedious, taking about 10% of the program's time. So I created a bit array which indicates the location of food. When food is plotted on the screen, a bit is set in the array. When a critter moves, it checks the bit array rather than having to read the screen.

At this point, I began to add new features to CRITTERS. The little guys now mutate their color, maximum energy level, reproduction cycle, and life-span. Critters now move in eight directions instead of four.

In addition, I have added a mutation that can "sense" food. This is implemented as a radial search, the distance controlled by the critter's sense gene. It turns out that senses are a very positive mutation; within a few hundred moves, the critters that have gained senses wipe out the "senseless" ones.

More additions are on the way. Carnivores and omnivores will be interesting. How about disease and disaster? Different species of food? Different food in different parts of the "world"? And, of course, what about bisexual reproduction? *(Editor's note: What about it? This is a family magazine.)* The possibilities are endless.

Of course, I tightened code here and there and removed some redundant loops. By the time I was through, CRITTERS had gone from 2,000 moves per hour to over 20,000! With the new graphics library, the program now compiles under the Microsoft, Borland, and Zortech compilers. Using Zortech C in integer-only tiny model produced a new CRITTERS program 40% smaller than the first.

C EXPLORATIONS

Now let's look at the character set generator, called CSG. CSG, shown in Figure 1, is a Zortech C program which lets you design complete 8x8 bit characters, visually. Figure 2 shows CS_Disp, a module containing the functions to load and display characters on the screen.

Let's look at CSG first. The idea behind CSG is simple. An array of eight 8-bit bytes holds each character. The array CHAR_DEF defines the structure of a character. CHAR_SET is an array of 256 CHAR_DEFs and represents a complete character set.

main() begins by checking to see if you've entered the name of a character set on the command-line. If not, it asks for a file name. If the file exists, the program lets you edit the character set. When you've finished, it saves the changes. All very neat and simple, huh?

It's the editor which is tricky. When a character is displayed for editing, it is shown in an 8x8 matrix on the left side of the screen. Each block in the matrix represents one pixel. One block blinks; this is the current cursor position. Pressing the keypad arrow keys moves the cursor around in the matrix.

Background cells are grey, foreground cells are white. To toggle the state of a cell between foreground and background, just press the space bar.

Press S to select the character to be edited. CSG will then ask which of the 256 characters you want to edit. You can enter the character directly (e.g., by pressing the "g" key to edit the lower case g), or you can hold the Alt key and enter the decimal value of the character on the keypad.

Once you've selected a character, the matrix displays its current representation for editing. Pressing X will exit the program and save the character set; pressing Q will exit the program without saving. All in all, a very simple process.

As cells are toggled, the CHAR_DEF matrix changes bits for the current character. The routines for blinking, setting, and unsetting cells contain some interesting video-display algorithms. There is no reason CSG could not handle different size characters. I made it simple to handle some simple graphics text problems.

CS_Disp is the module which makes the character sets useful. The function cs_load() copies a character set from disk into memory; only one character set can be loaded at a time. cs_putch() displays a single character, and cs_puts outputs a complete line. A specific pixel location on the screen displays characters. These are fairly simple routines.

I've found that having my own character generation and display routines has made graphics programming much easier and more pleasant. My routines run considerably faster than those provided by most vendors, and they use far less disk and memory space. Use them and abuse them; after all, that's why they were published. As always, I appreciate comments.

NEWS AND REVIEWS

Sorry folks! There aren't many new products this time around. Microsoft C 6.0 should have been out by now, but the development of a new "power programmer's" environment and CodeView 3.0 has delayed it. Lattice 6.0, which is supposed to get them back into the race, should also be out, but I haven't seen that either. Such is life....

RESOURCES

One book you must have if you use PC-compatible video displays is called *Programmer's Guide to PC & PS/2 Video Systems*, by Richard Wilton (Microsoft, 1987, ISBN 1-55615-103-9). This is the best nuts-and-bolts volume I've ever seen on every standard video adapter from Hercules to VGA. Wilton's book is written for assembly language programmers, which can be a problem for people unfamiliar with that art.

If you're a C programmer (which I assume you are if you read this column), another good book is *Graphics Programming in C*, by Roger Stevens (M&T Books, 1988, ISBN 1-558-51018-4). This book contains virtually no assembly, and the code in it works with Borland's Turbo C and others. This book isn't as deep as Wilton's, but it covers the subject well.

NEXT TIME

I've got more things to talk about than can possibly fit in the next few (years of) issues—but I'm going to try.

The next issue will feature my quick minimalist auto-sensing graphics library, ZIPGRAPH, a cornerstone of most of the graphics work I do. It's needed for the FRACLAND program, which generates fractal landscapes.

If possible, I also intend to do some work with ray-tracing in 640x400 by 256 color mode. Then there's the star-chart generator and the C++ library and the planetary system generator and more CRITTERS and some interesting new applications for fractal geometry and the 32-bit fixed-point math library and....

An author's work is never done, especially if he's having fun.

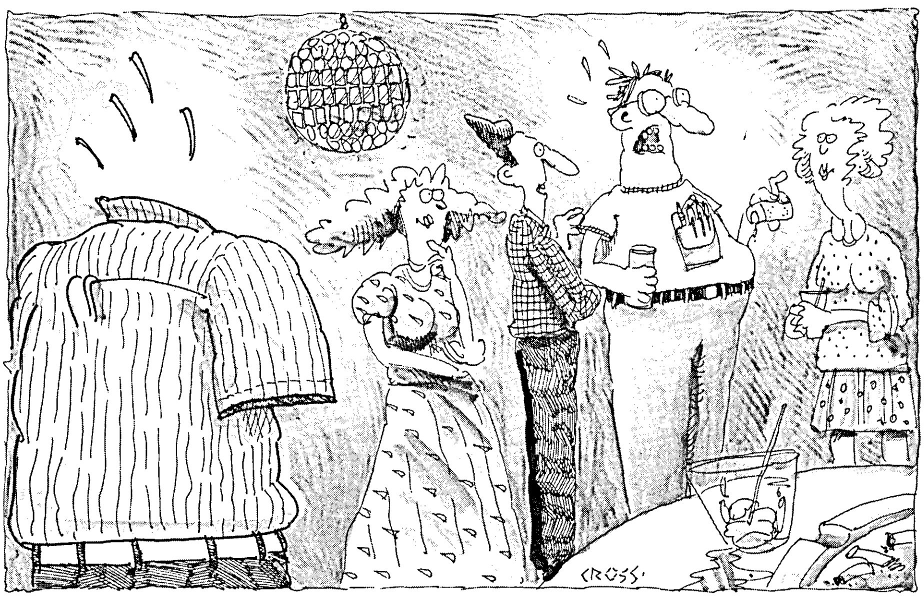
MMC AD Systems
Box 360845
Milpitas, CA 95035
(408) 263-0781

♦ ♦ ♦

CULTURE CORNER

Do You Feel Invisible?

By David Thompson
Micro C Staff



You know the problem. It's Friday night and you're attending yet another fantastic party. But, for all intents and purposes, you might as well be invisible.

You notice others who naturally attract crowds, who are the centers of attention as they move about the room. People cling to them and to their every word.

Who are these popular people? What makes them so interesting? There's one now, let's see what we can discover.

Immediately you notice he's gripping a can of Jolt in one hand, a half-eaten twinkie oozes slowly from the other. It's obvious that the color of his skin precisely matches the dingy whites of his eyes. Check out the ink-stained pocket protector safety-pinned to his T-shirt. Notice the thick, dirty glasses.

We approach within earshot: "Are the clones really compatible?" "Who designed the original IBM?" "What year did Lotus invent the spreadsheet?" "Does Adam Osborne really paint his toenails?" Notice that he answers only in carefully phrased monosyllables.

This man is clearly no lightweight. This man must be: AN ENGINEER.

Of course he is. And his suave party dress and studied manner shows he's a USED engineer, a graduate of Universal Systems Engineering Development's new program: VaporTraining. From the company that introduced VaporWare, VaporTraining is the most successful program for prospective engineers and other party animals. Plus, unlike universities, USED is no ordinary diploma mill.

Nowhere else will you get the training you really need to be the complete engineer. USED classes include:

- Surviving On Caffeine & Sugar
- Getting Technical Work Without Technical Training
- Pocket Protectors
- Advanced Pocket Protectors
- Fiction Writing
- Creating A Work History (prerequisite: Fiction Writing)
- Moving Into Management
- Dressing For Distress
- Cashing Your Paycheck First
- 100 Most Asked Party Questions
- Advanced Mumbling

Order any two classes and get the third class free. Complete four classes for a USED Bachelor's Degree. Complete six for a USED Master's, or all twelve for a USED Ph.D. (For a limited time, we offer special discounts to already USED party animals.)

And, our diplomas are so beautiful you'll want to display all of them in your party room. So send in your money ($5,000 per class) to:

USED VaporTraining
1001 Federal Penn Way
Lompoc, CA 93400

FINE PRINT: Include payment in small bills or other untraceable funds only. Foreign funds O.K. from countries with which the U.S. has no extradition treaty. We make no warrantee on usability of USED information or on your ability to understand it. From time to time we may actually ship USED products; such an accident in no way requires us to ship a product to any other purchasers. FINER PRINT: If you can read this, you're ready for our new course, Squinting Without Eyestrain.

♦♦♦

86 WORLD

Generating Z80 Controller Code On An AT Clone

By Laine Stump
% Redhouse Press
Merkez PK 142
34432 Sirkeci
Istanbul, Turkey

Laine finds that his fancy 18 MHz 80286 machine doesn't generate Z80 code. Herein he talks about some alternatives to buying a cross assembler. (And he talks about Turkey, and character sets, and....)

I remember it as vividly as last night's dinner. The long talks about why I was going off alone. The last few moments of being together, trying to tie up all those loose ends that you never think about until it's time to go. The final, tearful farewell as I headed off across the ocean and into the future, not knowing when or if we would see each other again.

Yes, I have missed my Big Board. In the four years since I left it sitting dejectedly in the X-ray tube box in my brother's garage, I can't count the number of times I wished I had this or that program. The stack of 80 or so 8" single density CP/M floppies is stashed in the closet in Woodinville, Washington. From time to time I have even considered sending for my old companion. ("I won't work for you unless you buy a first class seat for my CP/M machine.")

But the desire to have a CP/M machine has faded with the passage of time. After all, I do all my work on MS-DOS and XENIX these days. And I definitely don't have a shortage of programming tools anymore (C, C++, Modula, Pascal, MASM, CodeView, Zortech Debugger, numerous database and display libraries, countless editors...).

Turking It Up

After arriving in Turkey in 1985, I saw the first hints of a niche market that lately has been the source of most of my income. It has to do with the six characters in the Turkish alphabet which don't appear in either the IBM PC Graphics character set or in any ISO or ANSI character set. These characters are simply normal characters with the addition of an accent (like ö or ç).

When computers first arrived in Turkey, people were so glad just to have computers that they ignored the missing accents. This made computerized bank statements look quite juvenile (half the words were misspelled). That, they were told, was the price they paid for computerizing.

Around the time I arrived here, things started to get better. A few companies figured out how to change the character generator EPROMs on their video cards and write a TSR that changed the keyboard mapping.

Suddenly everybody had to have Turkish. "Does it do Turkish?" became such a common question that most companies began listing "Turkish EPROM" as a feature of their equipment.

These days, the first unit of a new model printer or video card goes straight to the "R & E" (Reverse Engineering) department to figure out where the character font is, how it's organized, and how to install the Turkish characters.

The Process

Lately I have been acting as the R & E department for several companies in Istanbul and Ankara. To aid me in this work, I have written (actually, "caused to evolve" is a better term) a program called BITED, a sort of visual version of DEBUG.

When I'm converting a machine, I usually search around in the machine's EPROM with BITED until I see something that looks like character bitmaps. Then I massage the display format a bit until it looks understandable. I replace the original characters with the Turkish characters and burn a new EPROM.

Nowhere in the process do I ever need to pay any attention to what processor is in the machine, or worry about any assembly language. I'm just modifying data.

For example, most of today's dot matrix printers use an NEC integrated 8-bit MPU chip called the uPD78C10. I have Turkified probably ten different printers which use this chip. Although I have a data book for the uPD78C10, I've never needed to look at it.

Character Set Standards

All the equipment that I Turkify plugs into IBM compatible micros. Almost all equipment that plugs into IBM compatible micros supports the IBM PC Graphics character set (which includes international (accented) characters at the codes between 80h and A7h).

It happens that the Turkish standard for this character set puts the special Turkish characters in place of already existing characters not used in Turkish. There is already a bitmap corresponding to the Turkish character code. When the printer receives that code, it knows it should display the bitmap.

A Bird Of A Different Feather

I recently had to deal with something slightly different. I was adding Turkish to the display and keyboard of the Olivetti WS-685 terminal (a private label of an Ampex product), which is a clone of a DEC VT-220 terminal.

The VT-220 (and, hence, the WS-685) uses an ANSI character set instead of the IBM PC Graphics set. This character set contains more international characters than the IBM set, but they are in a different range (A0h - FEh). The codes from 80h to 9Fh are used as control commands (similar to the familiar ASCII codes between 00h and 1Fh).

There were plenty of empty spaces in this ANSI set to add Turkish characters without displacing existing characters. But after I added them to the bitmap, the terminal refused to display them. That meant I needed to dig into the control program in the terminal's EPROM.

Luckily, the WS-685 is based on the Z80 processor. ("AHA!" you say. "Now I know why you babbled on about your Big Board!") Unluckily, I have absolutely no Z80 development tools that run under DOS. A quick search through my list of acquaintances came up with zero who had a Z80-based CP/M machine.

After searching through four or five dusty boxes full of books, I found my well-thumbed Mostek Z80 Reference Manual (from the days of EE 325) and began to disassemble by hand. This disassembling led me to a table with one byte for each character code, telling what action to take with the code. I changed the table and, violà, it worked.

Capitalizing

Now that I could display Turkish, it was time to modify the keyboard so I could type Turkish. I searched through a dump of the program EPROM until I saw the familiar "QWERTY" (strangely, only in upper case). I changed a few bytes.

After burning a new EPROM, I could type Turkish, but it always came out in upper case. Another search through the EPROM revealed no lower case table for the keyboard. That must mean that there is a tolower() subroutine somewhere that is called when shift or caps lock isn't down. Back to the Z80 reference and the notebook scratched full of ones and zeros....

After some time, I had managed to deduce my way into the tolower() subroutine. Assuming that tolower() would be entered with the character in the A register, and a compare would be done with the upper and lower limits of the capital letters, I hand assembled those few instructions and searched for them with SYMDEB.

Amazingly, I found them. (Lucky for me, since the program EPROM is 32Kbytes and it's full! I think I'd rather eat moss than try disassembling 32K of Z80 code by hand....) I modified tolower() for the special cases of the Turkish characters. The new EPROM worked, and I was done (for the moment).

Never Again

This whole experience started me thinking again about my Big Board. Just think if it had been here, sitting next to my X24. I could have easily finished the job in one-fourth the time (it took me two days). What about next time? Surely there will be another terminal based on the Z80 that needs to be Turkified.

I thought again how my Big Board uses 8" diskettes. I remembered that, with drives, it weighs something slightly less than a 1963 VW Beetle. I realized that, no matter how much loving care we put into its construction, it almost surely wouldn't survive the bashings it would receive on a flight from Seattle to Istanbul.

A Solution

Then I thought of something else. What about that little Micro Solutions Z80 Coprocessor card that Emerald Microware is selling? That doesn't even weigh as much as the left front fender of a '63 Beetle. Besides, it's faster than my Big Board (probably faster than a '63 Beetle, too). And it will eliminate all the problems of transferring data between two incompatible machines.

A few fax messages later, I had a Z80 coprocessor card and UniDOS winging my way. Ten days later I got the note declaring that it was in customs. I should come down to the docks and find out which body parts they wanted to secure its release.

(Don't ask me why the airmail customs office is on the waterfront. Probably for the same reason it took them seven days to notify me that I had an Overnight Express Mail package waiting for me.)

I hiked down the hill from my perch in Cihangir, overlooking the Bosphorus, and walked the short way to the Customs Post Office.

Luckily, while waiting in line for the third time to get a piece of paper stamped, I ran into a familiar customs official. He'd been around a few weeks before when I came down to pick up a suspicious looking package from PC Tech. He remembered me (actually, he remembered I had promised to look for an American wife for him) and mentioned to the agent clearing my package that I wasn't such a bad kid.

"Look here!," said the agent, pointing at the line on my paper that said "Duty to be Paid." "You should be paying duty on this, but I'm not charging you because you can speak Turkish." (So few foreigners take the time to learn Turkish that anyone who can utter anything resembling a complete sentence impresses them.) I thanked him, grabbed my new toy, and left.

Micro Solutions Z80 Coprocessor

The Micro Solutions Z80 Coprocessor card is a half-length card that can plug into any empty slot on a PC or AT compatible. The card contains a Z80H (8 MHz), 64K of RAM (8 x 4164), and a few TTL glue chips. Other than one address jumper used to set the location of the Z80 <—> 8086 communications port, there is nothing about the card that needs mentioning. It's small and cute; you plug it in and forget about it.

Included with the Coprocessor card are two pieces of software: UniDOS and UniForm. UniDOS is a TSR program that enables you to run any CP/M-80 program on a PC with the Z80 Coprocessor installed. UniForm is a combination device driver and TSR that allows reading, writing, and formatting of dozens of different CP/M format diskettes in your PC's drives.

UniDOS

UniDOS is the software interface which manages communication and coordination between the Z80 and the 8086. After installation, UniDOS runs CP/M programs while making itself almost completely invisible to the user. It accomplishes this by trapping all DOS calls to execute a program and checking if the program you're requesting is a Z80 CP/M program. If so, it loads the program into the Coprocessor memory and starts up the Z80.

While the CP/M-80 program is running, UniDOS also handles all I/O requests, translating display, keyboard, disk, and printer commands from CP/M to DOS. It handles both BDOS and BIOS requests properly, even translating some of the important fields in FCBs (file control blocks) for those programs that make use of the information.

UniDOS decides a program is CP/M-80 if one of the following is true: the file has a .COM extension and is on a CP/M format disk (accessed by UniForm); the file has a .COM extension and is in one of the directories specified by the "CPMDIR" command; or the file has an extension of .CPM.

CP/M Compatibility

UniDOS seemed near perfect in its emulation of CP/M. I didn't encounter problems with any of the CP/M programs I ran (including ZSID, DDT, ZZSOURCE, Turbo Pascal 2.0, M80, L80, VEDIT, EXPRESS), although I avoided some programs on purpose.

UniDOS has one problem with programs that do direct I/O (e.g., MODEM730). There are provisions for sending hardware I/O requests to the 8086, though. These programs could be patched to make these requests (MODEM730 must be patched for each different type of machine anyway).

Both CP/M and DOS programs can access both CP/M and DOS diskettes. I even ran DU77 (a CP/M disk sector editor) to look at MSDOS disks (floppy and hard) and it worked perfectly. UniDOS even cooked up some disk parameters for DU to display (number of tracks, sectors, allocation groups, etc.).

Integration of CP/M and MS-DOS

I also became curious about how well UniDOS integrates the running of CP/M programs in with MS-DOS. To test this, I tried running CP/M programs from within other programs (i.e., somewhere else than the DOS prompt). First I went into XTREE and told it to execute a CP/M program. It worked perfectly (although only if it was a CP/M .COM file in a directory marked by the CPMDIR command). Then I went into EXPRESS and gave a command to execute an external CP/M command. This also worked.

About the only thing you must avoid is losing track of which .COM files are MS-DOS and which are CP/M. Don't mix them together. Either keep all the CP/M programs in their own directory and mark it with CPMDIR, or rename all CP/M .COM files to .CPM the second you copy them onto your hard drive.

UniForm

UniForm is a program that creates a new virtual disk drive. This makes a CP/M diskette inserted in your floppy drive appear to DOS programs as a DOS diskette. Uniform version 2.13 (data version 1.12) contains 204 different diskette formats from ABC to Zorba, including Kaypro (and Pro-8) and Morrow. Most are CP/M formats, although there are a few strange MS-DOS formats, such as 96 tpi DEC Rainbow.

I expected UniForm would only allow me to copy to and from CP/M diskettes with special, internal commands. I was favorably surprised when I discovered that the "Uniform Disk" works with just about any DOS command. Not only can I use COPY, DIR, and ERASE, but I can also use XTREE, Brief, Zortech C++, Turbo Pascal, or any other program.

People using the same word processing program on incompatible systems and sharing files can just edit the file right there on the floppy diskette; no need to transfer it first. Again, Micro Solutions has integrated their software into DOS almost seamlessly.

The Seams

A few seams appear in other areas, though. My first complaint is that UniForm doesn't support CP/M user areas. All files come up as user 0. This doesn't seem to be much of a problem (all the files are still there) until you have two files in different user areas with the same name. Although I haven't tried this, the UniForm manual states that "unpredictable results may occur."

My second complaint is that they did not include the formats for Slicer diskettes. That would be okay, except there is no documentation on adding your own formats to the UniForm data file.

I remember getting a letter from a *Micro C* reader a while back telling me how to do this, but it has become impossibly lost in my famous cardboard box file system. Until I find it, I just have to live without all those programs I carefully packed onto 788K Slicer diskettes. (I deported the Slicer two years ago for lack of proper documentation.)

Something slightly related to this: it would be wonderful if UniForm would attempt to narrow down the drive type selections for me. It could look at the diskette and throw out all formats that didn't match the diskette's physical format (e.g., only display disk types that are 1024 bytes per sector, double sided). I have some unlabeled disks, and it takes a long time to discover what format they are by trial and error.

DOS 4.0 Incompatibility

My last complaint about UniForm is that it doesn't work with DOS 4.0. The resident part of UniForm is in a device driver. When I try to boot DOS 4.0 with UniForm, the system locks as soon as it loads UNIFORM.SYS.

I have two versions of UniDOS, the Coprocessor version and the Interpreter version (discussed below). The Coprocessor version (1.25) locks up under DOS 4.0, while the Interpreter version (1.11) runs just fine.

Fortunately I repartitioned my hard drive recently to make the DOS partition less than 32 Mbytes (making space for XENIX). Because of this, I can still read the drive with DOS 3.3 (which *is* compatible with UniForm). If you use DOS 4.0 for the large drive partitions, though, you'll have to use UniForm and UniDOS with a floppy-only system.

These problems are just slight annoyances. Mostly I am quite impressed with UniForm and UniDOS.

Interpreter Version Of UniDOS

The version of UniDOS I have been talking about is specially tailored to work with the Z80 Coprocessor card. There is another version, though, that uses software to interpret the Z80 machine code. I tried this version and was mildly surprised.

Memory dumps with DDT scrolled past the screen at nearly the same speed as SYMDEB. VEDIT and EXPRESS even edited text at a reasonable speed (except wordwrap). Very acceptable. ZZSOURCE did start to bog down a bit when its symbol table got large, but it was still workable.

Of course, I'm running a 12 MHz, 0 wait state 286. On an 8 MHz 8088, it might be a dog. However, the interpreter version has a mode that runs 8080 (a subset of Z80) programs on a V20 chip. Although not all programs contain only 8080 instructions, the ones that do will run just as fast on the V20 as on the Coprocessor board.

You may think now that you should forget about the Coprocessor card and get the interpreter version of UniDOS. I suggest thinking a bit more. The Coprocessor board is $169 and includes UniDOS and UniForm. The price of the Interpreter UniDOS with UniForm is $135 (including a V20 chip). Unless you need that slot for something else, I think it's worthwhile to spend $34 more and get the Coprocessor board.

Note, however, that the UniDOS included with the Coprocessor board does not support the interpreter or the V20 mode.

Another Suggestion

Don't even think of getting UniDOS without UniForm, although the opposite would be useful if you just needed to exchange data files. UniDOS would have been worthless to me if I hadn't been able to read the programs from the pile of Kaypro disks I've hoarded these last four years.

Related Products

Micro Solutions sells several other interface cards for use with UniForm. These cards allow you to read and write diskettes for Apple II, MacIntosh, North Star hard sector, and most 8" single and double sided formats.

Why?

You may now be wondering why "Laine the DOS Jock" has spent so much time talking about a "mere Z80." What good is it to you? Why should you waste your money on it?

If you only work with DOS, and never do any work with embedded controllers or anything else that might contain a Z80, UniDOS is slightly less than worthless. You may as well spend the money on flea collars for your armadillos.

If you're doing development work on anything that has a Z80, you can save yourself a considerable sum of money by buying the Z80 Coprocessor. Consider that a Z80 cross assembler running under DOS can easily cost $500. Alternatively, you can pay $169 for the Z80 card and UniDOS and buy the Z80 Macro Assembler from Micro C (disk K25) for $6 (plus $2 S/H to foreign countries). I've even seen copies of CP/M, complete with all utilities, for $25.

Of course, if you're running something like a publishing house (or a magazine) where you get in writers' work on anything from papyrus to ½" mag tape, UniForm could do wonders for your sanity.

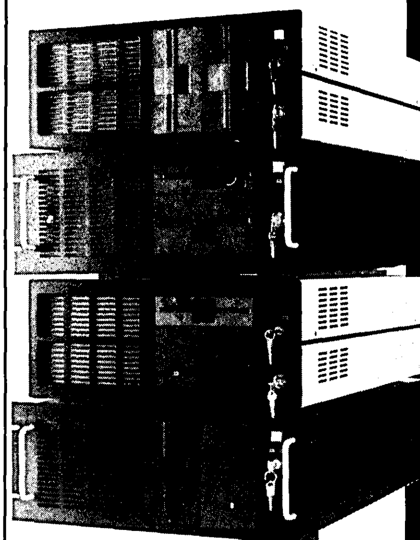
All you newcomers to the Micro C clan could use the Z80 Coprocessor to see for yourselves what it was like in the old days.

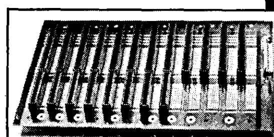
Conclusion

I have several other "Turkification" projects waiting. I'm sure that at least one of the machines in question will have a Z80 processor. When it's time to work on that machine, I'll be ready.

Emerald Microware
P.O. Box 1726
Beaverton, OR 97075
(503) 641-8088

♦ ♦ ♦





ON YOUR OWN

The Mercenaries: Silicon Valley Contractors

By Todd Hoff
1075 W. Remington Dr.
Sunnyvale, CA 94087
(408) 730-8410

Want to turn your C experience into $25 to $125 per hour? Want to work 80+ hours a week? Just print up some business cards and catch the next flight to Silicon Valley. Of course, there are a few things that money won't buy, but you'll find out about those after you arrive.

I've become something my mama warned me against—a Silicon Valley contractor, a software mercenary. Constantly chased by headhunters, scurrying from company to company, I lead a dangerous but interesting life in the Valley.

Silicon Valley, everyone's heard of it. Many dream of coming here. What will you find when you arrive? What's it like to be a contractor in the fabled Valley? Read on.

Four years ago I packed my bags, leaving my beautiful but job-starved emerald isle of Eugene, Oregon, for the techy promised land—Silicon Valley. At the time, all I knew about Silicon Valley lay printed in the *San Jose Mercury News* help wanted section; literally page after page of engineering jobs stared back at me, beckoning me with opportunity, challenging me to come. So I did.

Here I, along with the thousands of other immigrants from Russia, India, Vietnam, Taiwan, Sweden, Germany, Ohio, and Texas, found both opportunity and challenge in this former land of oak trees, apricot orchards, and old tractors.

Getting To Know The Territory

As a mercenary, the first thing you want to know is the lay of the land. Every place in this area is specified in relation to the San Francisco Bay. Silicon Valley, although existing on no map, is the little kingdom on the western shore of the Bay, usually referred to as the South Bay.

East Bay lies to the east. This area is expanding exponentially and includes Bezerkly (Berkeley of BSD fame), Oakland, and Concord. The North Bay includes the politically correct cities of Marin and Sausalito. Last, but certainly not least, is the Peninsula, home of The City, northern California's favorite son, San Francisco.

The whole area is one frenetic job producing machine. Producing, as in war, perfect conditions for the software mercenary. Contracting in the Valley is perhaps unlike contracting in any other place in the world.

There's a tangible energy in the Valley, flowing from the people and companies continuously pummelling and pushing at technology's frontiers. There's a fever here, causing normal people to work 80 hour weeks for months on end. Always omnipresent, like a golden carrot, lies the hope that your stock options will become more valuable than your toilet paper (although never as comfortable).

Power Brokers

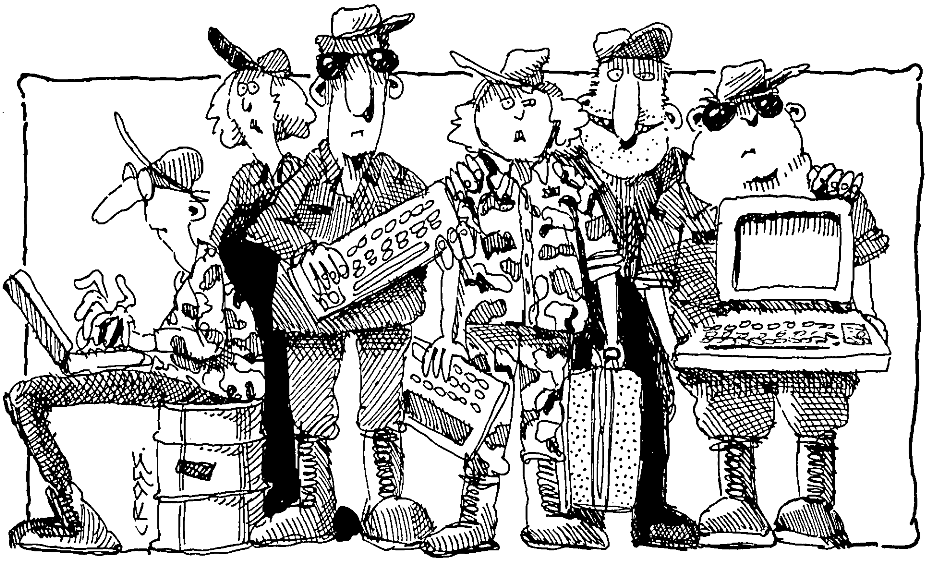
Next, the veteran mercenary must know the major players. Everybody's here. All the biggest companies and all the wanna-be-biggest companies have active development staffs in the Valley.

Opportunities are nearly endless—during good times, of course. The mercenary can take advantage of strained super power relations (DEC, IBM, Amdahl), or border skirmishes (SUN, MIPS, Apple), or foreign powers (NEC, Sony), or countless other third world countries (start ups).

Rarely, however, do you contact a client directly (as you would if you were an independent contractor). Normally, introductions are made through an intermediary (job shop). Job shops are like floating countries, hiring the mercenary for an hourly fee, finding a needy company, and then reselling your services at a higher rate. Sometimes a considerably higher rate.

Independent contracting is good if you can get it. You get to keep the full fee and your hours are very flexible. But (and there are lots of buts to independent contracting), the marketing, the problems of running a small business, and the responsibilities to clients often push folks into job shops.

"Tom Jones," a local independent contractor



As a mercenary, the first thing you want to know is the lay of the land...The whole area is one frenetic job producing machine.

who wishes to remain anonymous, got in via a career switch. "I started 15 years ago," Tom says, "with no contacts, no nothing. Like most independents, I got all my jobs through referrals. It was tough at first, but over the years I've built a steady and loyal clientele."

Now Tom is in the envious position of being picky. "I look for a synergy when accepting clients," says Tom. "I've upped my value by specializing in one area. I look for clients who fit into my specialty so I can make my jobs work together. One drawback is that I'm on call 24 hours a day. If there's a problem, I have to fix it and fast because many of my programs perform vital functions."

For those potential mercenaries who don't want to build clientele, but still want to play, job shops offer attractive rates, some benefits, and extreme ease of use. Job shops do all your marketing. They contact all the clients. They set up the interviews. They negotiate rates. All you have to do is interview...and work.

"I have a few customers I've built relationships with," says Mark Lewis, a local contractor with over ten years experience, "but I go primarily through job shops. My long term goal is to become an independent contractor. The longest I've been without a contract is seven days, and another guy I know was without a contract for a total of three days in three years."

Tour Of Duty

"UNIX porting is popular," says Anna Osborne, a local contractor. "If you just mention porting you got it made. Of course, everybody wants X Windows. Databases, networking, graphics, and surprisingly, firmware. I was originally a compiler person, but there weren't very many compiler jobs."

UNIX and C are hot. Almost any experience with either will get your foot in the door. I knew someone, don't quote me, who just read a UNIX manual, had programmed in C on MS-DOS, and got a UNIX job. Perhaps not completely ethical, but he was sure he could handle it—and he did. A mercenary must be honest enough about his or her abilities to make these judgments.

Not surprisingly, this area offers many contracts for the Macintosh if you're comfortable with MPW or C. Also in demand are testing and technical support positions. It's difficult to find permanent people who'll take these dirty jobs so companies often hand them out to mercenaries.

The Timid Need Not Apply

No matter what the job, a mercenary position is not for the timid. Many people need the patronage of a large company. Why would anybody leave the comfort and security of a large company for the fetid jungles and night marches of contracting? The answers may surprise you.

Mark found contracting attractive for having "...more freedom, less responsibility, and more pay. The less responsibility aspect is something I think most people don't consider. You have a project. You don't have to worry about the long term implications of anything. Working for a couple of start ups, I carried a lot of responsibility. I got tired of worrying about it."

Anna left the fold of a large company because "after working at a company for a while, technology passes you by. I was so caught up in the bureaucracy, I wasn't free to do anything. As an employee it doesn't look good if you jump from company to company. But as a contractor, you can jump around and stay up to date on the latest stuff."

For many, contracting is predestined. The Valley's environment almost encourages you to become a contractor. Very few people I know ever stay at a job more than two or three years. Job turnover in the Valley averages 50% every two years. Explaining the process Mark says, "I moved every two years, anyway. It was never my intention. That's just how it worked out. After having six or seven jobs in ten years, I decided to make it permanent."

War Is Hell

Risk is inherent in the mercenary's work. The mercenary must always be weary of surprise attacks. Tom has many gruesome stories. "One guy I know has worked for three companies that closed down with no notice. At one place, he showed up for work Monday morning to find the doors locked."

Company loyalty—at least in the Valley—is dead. Mercenaries merely recognize the reality. "Many people I know," says Anna, "say that once they've learned all they can learn at their company, they'll leave. Nobody cares because the companies don't care."

Lack of loyalty stems from the cycles of innovation and obsolescence. On the upward portion of the cycle, people are worth more than gold, vital assets to be treated royally. On the down side, people are dust, liabilities that must be exorcised from the ledgers. I've heard that many companies turn to contractors for public relation reasons: firing contractors doesn't make headlines.

Another problem for the mercenary is getting paid, sometimes a battle. "One of the beauties of working for a job shop," Tom says, "is that you always get paid. I've never worked for free, but it's been close. One tactic companies use for getting out of paying is complaining about your work. No matter how detailed a spec you have, there are always misunderstandings. Sometimes a client won't pay because 'I didn't do what they wanted.' This usually means the company doesn't need the software anymore so they don't want to pay me for it."

Several times people have asked me to work for free. Here's the deal: I work my butt off, day and night. I forego even the semblance of a family life. For what? For some stock options and promise of payment when the product sells "to this long list of customers just begging for it."

Another deal used by start ups is to pay up front for three months' work. The contractor then works six months assuming there's payment and bonus ready when the product is done. If the startup happens to go under, as many do, you're out of luck. Although contractors are one of the first in line to be paid from bankruptcy proceedings, chances are you'll never see a cent.

Cleanup

Latrine duty is very common. Industry often brings in contractors to fix poorly designed, badly implemented projects. Somehow it never occurs to management that they should pay for the expensive people up front, not when it's almost too late.

The "almost" means that contractors are often seen as cannon fodder. Because a person is only a contractor, asking them to work 7 days and 90 hours a week is okay. Oh well, mercenaries work for money, not glory.

Always remember: as a contractor you are just meat satisfying a temporary hunger. Eventually you will leave, expectedly or unexpectedly, but you will leave.

As a mercenary you have no ties, no loyalty. None is asked, none is offered. For money, you give someone the right any day, any time, to say good-bye.

Wages Of Sin

Software mercenaries, however, eat well. As a beginning contractor, say with 2-3 years solid technical experience, you could start at $35 per hour. With a couple of years of contracting experience, you've proven yourself so you can command $40-$50 per hour.

A medium level specialist, say in X Windows, device drivers, kernel work, or networking, can ask $50-$70 per hour. For high level specialists, with lots of pertinent experience, the sky's the limit. I know several people who get $100-$125 per hour and one who gets $250. Granted, contracts at this level may be shorter term—but who cares!

The Field Is Bloody

Lest you think these rates too high, remember: Silicon Valley workers deserve both hazardous duty pay and housing allowances. The Bay Area is beautiful, but all is not well in Oz.

Dennis Hayes quotes some startling statistics in his book, *Behind the Silicon Curtain*. It's estimated 60% of high-tech workers are seeing psychiatrists. This area has more divorces than marriages. In 1987, over $500 was spent on drugs per man, woman, and child in San Jose, although both Anna and Mark say they don't see much drug use. Anna jokes, "Do caffeine, sugar, and cigarettes count as drugs?"

Also, Silicon Valley has more Superfund sites than any other area in the nation. And the hills become a little harder to see every year.

Housing is atrocious. Only 9% of the people in California can afford a house. A house in Palo Alto, 1300 square feet on ⅛ of an acre, sold recently for more than $600,000. More reasonable areas offer cheaper houses but tack on an hour to your commute. By the year 2000, the state's average house is supposed to reach a cost of $470,000. The average commute, now 45 minutes, is projected to be 2 hours by 2010. Lovely.

People also forget the cost of little things they take for granted at a company. Health insurance is expensive. Retirement plans are nonexistent. And all those paid holidays for employees are unpaid for mercenaries.

I hear you saying that as a mercenary you get all the benefits of being a small business. Not so. New rules have made it much more difficult for contractors to qualify as businesses. The mercenaries' mercenary—the IRS—now takes a dim view of W-2 contractors saying they are a business. And because of IRS pressure, most job shops won't 1099 you anymore. You are, in effect, an employee of the job shop. In reality, of course, this is total BS.

Why Do They Keep On Keeping On?

When the dust has settled, the battle ended, wounds bandaged, the mercenary moves on. Mark, Anna, and Tom all say that they like contracting and won't stop. To them the advantages outweigh the perils. Most people wouldn't agree. But as Anna says, "Contractors are obsessed with independence. Disillusioned with large business, they want to do it all themselves."

♦ ♦ ♦

Coming Up

The whole idea of picking topics for issues happens to be a bit controversial around here.

"Don't set topics, let readers be surprised."

"Writers will see the topics and assume we're not interested in anything else."

"It locks us in for a whole year."

It's also nice, however, to have something to work on. Some direction. So here goes.

- January—Embedded Systems
- March—Object Oriented Programming
- May—Micro Controllers
- July—Operating Systems
- September—Data Collection
- November—Graphics

You'll notice that the major topics alternate between hardware and software. I'd also considered running contrasting subjects, such as running Exposed Systems following the issue on Embedded Systems. Running Micro Supporters after the issue on Micro Controllers. Running Nonoperating Systems (should be a great issue) after Operating Systems. And running Data Dispersal after Data Collection. (By this time you should have a pretty graphic idea how we pick the special subjects.)

Port Alberni SOG

You can take the SOG out of the country, but you can't take the country out of SOG. Port Alberni, Canada, was definitely country and the event was definitely SOG.

A solid (even solider after the salmon barbeque) 50 SOGgy families showed up at the local college and made it an event that reminded me of the very early SOGs. Good companionship, good food, good weather, good camping (and Bed & Breakfasting), and great scenery. The local community went out of its way to make us feel welcome, and our hosts (David Stern and Randy Young) had a great time despite their initial nervousness. (Neither had even attended a SOG before.)

The only down sides of the whole event were customs and gasoline prices. Don Jindra and wife (and sons) drove all the way from Denton, Texas, to show off their $25 network package. (It really works.)

Canadian customs wanted $1,200 duty for his three demo computers. No $1,200, no entry. After turning around and heading back toward Texas (I understand he first exchanged a few words with the customs officer—something about backward countries remaining backward because of petty officials), he decided to try another entry point.

Under the eagle eye of a new customs agent, they dug out the computers and again explained why they were taking them into Canada. This time the officer wished them luck and let them through.

This was just one of their trials. Their first Hertz van broke down two hours from home. Then their keys got locked in the second van.

Fortunately it happened during the barbeque and we had a handy hardware engineer and a handy hardware debugger (often mistaken for a coat hangar). Made me wonder how many software folks it would have taken to unlock the van—assuming of course they couldn't just disassemble it.

Gunnison SOG

If the Port Alberni SOG featured food and scenery, the Gunnison SOG featured high mountain thunderstorms and energy. Human energy. Check out Karl's write up of this fine SOG and you'll understand what I'm talking about. Thanks for a great SOG, Scott and Maria.

On Your Own

I led a group discussion at the BC SOG about my favorite subject, being on your own. We shared tales of failures: unpopular T-shirts, unmarketed hardware, and a dishonest distributor. And successes: the $25 network (again), a hospital package, and a kit robot.

When I suggested the possibility of turning a product over to marketing folks, there were hisses from the audience. (I might have mentioned used cars in the same breath.)

However, after the dinner, David Stern took my arm and suggested that I (we) might be a bit mistaken.

"Technical types design the product and then they look for a market. Marketing types find the market first. Then they see if someone can build the product."

He's right.

We can all point to situations where the design came first and the market followed: CP/M, Apple, *Byte*, VisiCalc, C, Osborne. (Osborne? That was the company which created the market for Kaypros.)

However, the computer marketplace has become a lot more mature and a lot bigger. Owning a small niche can be as lucrative as owning the whole shootin' match six years ago. You just need a lot sharper aim to hit the niche. That's where the marketing type, a real marketing type, can make himself invaluable.

Interested Folks at Robotics Demonstration, Port Alberni

Now, finding a real, knowledgeable marketing person and making sense of the information he generates ... well, that's another story.

You And Micro C?

Speaking of marketing, it's definitely time to expand Micro C. We should be exposing ourselves to more people, contacting more potential advertisers, even possibly going monthly. (Did I really say that?)

So, we're looking for a partner. A go-for-it person or group who wants to invest time, money, ideas, and energy in making Micro C the significant player it can be. We're ready to take a look at focus, market, everything. (The only things that have to stay are the light style, the informality, the readability, and the editorial. Not necessarily in that order.)

If you feel you and Micro C might be a good fit and you're willing to take a very significant (and demanding) role in a more and more significant magazine, then let's talk (503-382-5060). (Bend is a wonderful place to raise a family.)

A Surplus Idea

One of the best suggestions to come out of this year's regional SOGs so far: "Have readers send in information about their sources for surplus parts. Then print the list in *Micro C.*"

Wow! What an idea. Part of my startup time on a new project gets spent locating sources of parts. As difficult as it is finding normal retail sources, it's nearly impossible to find surplus dealers who have what I need. But, surplus prices can make the search worthwhile.

I remember purchasing 25 lb. grab boxes at the Tektronix surplus store. They wouldn't let me open them on the premises, so I didn't have the slightest idea what was inside (usually just a tangle of small parts swept off the floor at a board assembly area).

On occasion I found brand new power cords, reed relays, transformers, ICs in their original tubes, used soldering stations, and, well, other wonderful surprises. As for the floor sweepings, I'd hate to guess how many hours I spent sorting resistors and capacitors. At $7.50 per box, I was hooked.

Like the Tektronix outlet, some surplus stores are connected with corporations. Though many are for employees only, the rest are quite happy to sell to the public.

Then there are the independent outlets. They often purchase overstock by the pound (often for the value of the silver, gold, or tin). Then these shops try to sell as much as they can to hobbyists and start-ups before stripping the rest of their precious metals. Many of these stores haven't the slightest idea what it is they are selling. If it's big or impressive, they ask more. After something has blocked an aisle for six months, they ask less.

Shopping in person is by far the most interesting (and the safest). Many stores have a plethora of high-tech orphans. Since orphans almost never show up in catalogs, they're only available to browsers. They're also often just what you need (though you wouldn't have realized it if you hadn't seen them).

Anyway, if you're a junk aficionado, or just have regular dungeons where you stir up dust with the buddies on Saturdays, send us the scoop on them. (Just include those that deal with the public.) Include:

1) Name, address, phone number;

2) Person to talk to;

3) Days/hours they're open;

4) Whether they sell by mail/phone, to walk-in customers, or both;

5) Whether they have a catalog. What does it cost?

6) Their specialty (if they have one);

7) Whether they also sell non-surplus parts. (Give a general description);

8) Whether they have a return policy. What is it?

9) Prices (high/low/variable/wildly variable/chaotic). You might mention two or three examples of products and prices;

10) Include a paragraph or two about how they are to deal with, how knowledgeable they are about their stock, amount of stock. Is it mostly junk or prime, have you had trouble with them...?

11) Include any additional information you think is germane to *Micro C* readers.

If there's a shop you've just heard about but haven't tried, send the information you have and we'll try to get in contact with them.

Now, there's a reward for you in this. You receive a free Micro C issue disk (or *Micro C* back issue) for each firm you report on. (The disks and issues might not be surplus, but their price is right.)

Pascal Book

I don't use many books when I'm programming—for a long while just Borland's Turbo Pascal reference manual. However, not too long ago I found myself struggling with a sticky little graphics problem. It was so bad that I dug through all my Pascal books. Nothing, until I found *Complete Turbo Pascal*, Third edition.

How Jeff Duntemann has time to put this much effort into a book while being editor of... (well, it was *Turbo Technix*) escapes me. But he's done it. He's thrown in lots of sample routines, including, fortunately, copious comments on Borland's Graphics Interface. (He also does mouse drivers, inline and outline assembly language, low level system hooks, and lots more.)

It's a readable book and it has a good index. Plus, Duntemann isn't afraid to talk straight. I get a strong feeling that Borland doesn't print anything in its manuals that might sound like a bug. So I turn to Duntemann at least as often as I turn to Borland.

Complete Turbo Pascal
Third Edition (Covers 5.0)
822 pages $24.95
By Jeff Duntemann
Scott Foresman and Company
ISBN 0-673-38355-5

Speaking Of Books

One of the most famous and discerning places to get technical books, walk-in or mail order, is Computer Literacy. They're famous because they're famous. They're discerning because they've carried *Micro C* since around 2 BC (2 years Before the fancy Covers).

The New York Times, another significant publication (though they still don't have fancy covers) mentioned Computer Literacy in its July 25 issue:

"Engineers on their way back to their cubicles at AT&T or Xerox Corporation can stop at another valley landmark, the Computer Literacy Bookstore, where the magazine section includes copies of *Micro Cornucopia* and *HyperAge*."

Computer Literacy will sell you just about everything you'll need for those all-nighters, except the Jolt. So if it's technical, has pages, and has covers (not blankets), they probably have it. They say they ship within one day (national and international), they track down obscure books for no extra charge, and they'll be glad to put you on the mailing list for their free newsletter.

I usually see Dan Doernberg, founder of Computer Literacy, when I attend the West Coast Computer Faire, and I've always enjoyed talking to him. He's a great source if you're looking for the latest book on hi-tech arcane and obscure.

Computer Literacy Bookshop
520 Lawrence Expressway
Sunnyvale, CA 94086
(408) 730-9955

Ventura 2.0

Most of you have watched us fumble our way into desktop publishing. Despite the tribulation (which always occurs at deadline), I've found that desktop is as much an improvement over the old paste up procedures as word processing is over a typewriter. I couldn't go back.

We've used Ventura Publisher from the beginning. Version 1.0 was buggy, sure, but with three different ways to do any one thing, there was usually a work-around. As revisions rolled in, we spent more and more time doing the work and less and less time doing the around.

Now, with version 2.0 things seemed to take a giant step backwards. The most consistent thing was inconsistency. An article (chapter) that went together and saved perfectly might be garbage when called up again. Or, we'd load a figure into a frame and a nearby headline would get weird. Page numbers changed into headlines, often refusing to change back.

Sometimes Carol could fix the problems. Sometimes she couldn't. Either way, the next time the file was edited, there might be a whole new set of problems.

I called Xerox.

The only thing the support person asked about was memory. How much RAM? (640K.) Were we running TSRs? (No.) Did we have at least 520K of free memory before running Ventura? (Definitely.)

I hadn't been too concerned about RAM because version 1.X was running precisely, though slowly, on the same machine (as an article got long or complex, the older version would swap data between memory and disk).

Something about this new version was definitely not right.

Version 2.0 supports EMS memory. According to the manual, you don't have to have EMS memory (unless you want fancy hyphenation), but it supports it. So when Carol got exasperated with this issue's 3-D article (page two became totally strange the instant she loaded either of the two illustrations), I packed up her machine and headed for MicroSphere.

My hunch was that extra memory was more important than Ventura was letting on. I suspected that overflowing stacks or misaimed pointers were causing the problem.

"I want EMS, lots of EMS," I said as I dumped the dismembered machine onto their service bench. Fortunately RAM prices have come down a bunch. Unfortunately, Howard, Ed, and Allan were up to their ears in other work.

Ah well, I can stuff RAM and set dip switches with the best. So I did. Five hours and six false starts later, I had added 1.5 meg of EMS. (The EMS board was very smart, by the way, letting me configure its RAM as any combination of extended, expanded (EMS), and main memory. In fact, for over an hour it was a lot smarter than I was.)

The instant I got back to the office I cobbled up Carol's monitor, keyboard, printer, fired up Ventura, selected "Desk" (to get the version display), then pointed the mouse arrow to the word "Ventura" and clicked. Up came the secret diagnostic window. (Kind of like a secret decoder ring, but useful.) Sure enough, Ventura recognized 1.5 meg of EMS.

Then I loaded the troublesome 3-D article and looked at page two. Wow! The page number displayed correctly, unruly text from other pages had disappeared—the whole thing worked. I tried another article, and another, and another... everything worked. Ain't desktop fun?

Longhorn SOG Reborn

After seeing Don and Kim Jindra at both the Canadian and Colorado SOGs, I wasn't too surprised when I got a call from Kim. What did surprise me was the subject. The Longhorn SOG.

"We've got to do something more than the regular meeting at the Info Mart. I got hold of Stuart, he's hard to reach, and volunteered to take over the SOG arrangements. He said fine, so I've got it."

It turns out she's lined up meeting space for SOG talks, a barbeque, a place for two Jolt SIGs, T-shirts, and more. She's arranging SOG sessions on Friday in Denton, Texas (25 miles north of the Dallas/Ft. Worth Airport), then taking the group to Dallas for Saturday's Info Mart meeting.

If there's any chance you can attend, or speak, or whatever, get in touch with Kim right away. (Check the Longhorn SOG information box in this issue.) I'll see you there.

Longhorn SOG—Oct. 13-14
Kim Jindra
P.O. Drawer F
Denton, TX 76202
(817) 387-3339

David Thompson
Editor & Publisher

Letters *continued from page 6*

The Nature Of Radar

Regarding LIMBO's distance sensor ("Letters," Issue #49, p. 75): Whether the intensity of reflected light, or a radar signal, is inversely proportional to the square or to the fourth power of distance depends on the nature of the reflecting target. If the reflection is from a plane surface that reflects without scattering, the inverse square applies. If scattering occurs at the target, inverse square applies for both the forward and the return paths, and the product is the inverse fourth power, the radar equation.

LIMBO's received power from his (her) own source as seen in a mirror would be inverse square, but from a white object would be inverse fourth power. The polished ball bearing beloved of telescope makers as a test object will produce an inverse fourth power response for LIMBO's sensor in spite of the specular nature of the reflection, because it produces the scattering that is the underlying reason for the inverse square law.

Not only the library's dusty tomes on radar show the inverse fourth power. The Second (1988) Edition of *Antennas*, by Professor John D. Kraus of Ohio State University, gives the radar equation and explains the fourth power of distance in the denominator. The equation is alive and well, as good today as the first time someone found that doubling the transmitter power didn't buy you much early warning time. It also includes other factors, such as the effective cross section of the target.

The inverse fourth power behaviour was responsible for the effectiveness of a very successful anti-submarine measure during World War II. Submarines used to surface at night to run their diesel engines and charge their propulsion batteries. Maritime reconnaissance aircraft would search for them with radar, and the subs used receivers on the radar frequency to warn of their approach.

This was very effective, as the signal arriving at the sub was detectable by its receiver long before the reflected echo from it was strong enough to appear on the radar receiver in the aircraft. In fact, the aircraft were detectable at such long ranges that the subs would only submerge when the signals indicated the aircraft was getting close. This avoided the loss of valuable battery charging time when the aircraft could be a hundred miles away.

To counter this countermeasure, the airborne radars were fitted with attenuators in the transmitter output, in the form of a vane that could be cranked into a slot in the waveguide. When a sub was first detected, with the radar transmitter at full output, the aircraft would turn for a run at the submarine. The radar operator would crank in the attenuator as they ran, reducing the power steadily to keep the received signal constant as the range decreased.

At the submarine, the receiver operator would hear the radar signal decreasing from the time the aircraft went into kill mode, and assume that the aircraft was going away. With the noise of the sub's diesel to mask the sound of the aircraft's engines, it was usually possible to arrive over the sub before it submerged.

The effectiveness of this system is apparent if you use the inverse square to calculate the signal at the sub, and the inverse fourth power to calculate the signal at the aircraft, assuming a transmit power adjusted at all times for a constant signal at the airborne radar receiver.

So be careful! Failure to appreciate the inverse fourth power law in the radar equation has been known to be fatal!

John Innes
120 Macpherson Street
Cremorne, NSW 2090
Australia

Editor's note: All this leads me to a bit of specular reflection. Placing an alien infrared detector (for detecting alien infrareds, of course) and a fairly substantial laser on your robot might be the most effective way to sink other maze contestants. (I knew you'd get a charge out of that.)

♦ ♦ ♦

UNITS & MODULES

By Michael S. Hunt
2313 N. 20th
Boise, ID 83702
(208) 336-7413

A Short Look At Sparse Matrices

Linking Your Way Into Tables

Michael is short this time. It turns out he's still recovering from the Jolt SIGs at the Gunnison SOG. So he's covering sparse matrices in a sparse sort of way.

If you've looked ahead at the code for this issue, you've probably noticed that I haven't included the promised B-tree balancer. Instead, the code this time supports a linked list implementation of sparse matrices. (Sparse matrices are like regular matrices only smaller, we hope.)

Well, what can I say? It's summer, and there was SOG, the new Turbo Pascal 5.5 arrived, the new Stony Brook Modula-2 compilers arrived, and I went scuba diving. Still, I wrote the AVL routine (named after its developers), but right now it has a vaguely annoying tendency to drop pointers. So I'll cover the AVL routines in the next issue.

SOG

SOG, as always, was great! My wife Pamm and I made the trek from Boise to Gunnison, Colorado, where I gave a talk about the differences between Pascal and Modula-2. After about an hour, the talk turned into a discussion about the column.

Two camps emerged, one wanting more simple toolkits that explained common algorithms. The other wanted powerful applications-oriented toolboxes. I'll try to satisfy both camps by showcasing an algorithm from a robust toolbox. We can still put full source code on the issue disk and the Micro C BBS.

Turbo Pascal 5.5

On the surface, 5.5 looks like 5.0. But scratch the surface and hmmm, what's this, it looks like OOP (Object Oriented Programming). I also received the Turbo Assembler and Turbo Debugger. Soon I'll start digging into all of it. I have some code that could use assembly code speed so I'll discuss interfacing Pascal and Modula-2 to assembly code routines. I'll also compare writing with OOP and without OOP.

Stony Brook Modula-2

My upgrade to version 2 of Stony Brook's Modula-2 compilers just arrived. With support for DOS, OS/2, Windows, QuickMod compiler for development, and optimizing compiler for production, it's a very impressive package. I'll talk in greater detail about the Turbo and Stony Brook products next time.

Scuba Diving

It was great! A whole day of diving in a high mountain lake. The water was cold, the air was clean, and the fish weren't biting.

That Little Bug

At the last minute I realized I could save some CPU time and simplify my AVL code. Foolishly I began a quick rewrite. The VAX editor at work saves all versions of a file, but the Turbo editor saves only one .BAK file. Ah, to experience Laine's Lament for a universal editor. Soon the original version had been wiped out and I had no back up.

Sparse Matrices

In numerical analysis, modeling, simulation, and many other fields of mathematics and data analysis, we often deal with large (1000 x 1000 or larger) matrices of data. The data are often real numbers at four or eight bytes apiece. This can consume more memory than many computers have.

Many of the matrices used in these applications are called sparse matrices, meaning that most of their elements have the same value, usually zero. Some common forms of sparse matrices are upper and lower diagonal matrices and tridiagonal matrices found in systems of differential equations.

The toolkit for sparse matrices (Figure 1) stores only the nondefault elements of the matrix. The elements are stored in an array of linked lists. The function Sparse returns the i^{th} row and j^{th} column element from the m^{th} matrix.

PutSparse stores an element in the linked list if it doesn't equal the default value for that element. The unit supports MAXSPARSEMATRIX (currently 3) matrices. Each matrix can be up to MAXROW (currently 1000) rows by 65,365 columns. SetDefault lets you change the default value for the m^{th} matrix.

The sample program (Figure 2) shows several common matrix operations using the SparseMatrix unit.

Next Time

Will there be AVL trees next time? Yes Virginia, there really is an AVL routine. After that, back to reality.

♦ ♦ ♦

Figure 1—UNIT SparseMatrix;

```
INTERFACE
CONST MAXSPARSEMATRIX = 3; DEFAULTVAL = 0.0;
TYPE dataType = REAL;
VAR defaultValue : ARRAY [1..MAXSPARSEMATRIX] OF dataType;
FUNCTION  Sparse(i, j, matrix: WORD) : dataType;
PROCEDURE PutSparse(i, j, matrix : WORD;value : dataType);
PROCEDURE SetDefault(default : dataType;matrix : WORD);
IMPLEMENTATION
CONST MAXROW = 1000;
TYPE  nodePtr = ^nodeType;
      nodeType = record
        data : dataType;
        col : WORD;
        nextCol : nodePtr
      END;
VAR sparseMat:ARRAY [1..MAXROW,1..MAXSPARSEMATRIX]
               OF nodePtr;
    j, k : WORD;

PROCEDURE SetDefault(default : dataType;matrix : WORD);
BEGIN
  defaultValue[matrix] := default;
END (* SetDefault *);

FUNCTION  Sparse(i, j, matrix: WORD) : dataType;
VAR p : nodePtr; done : BOOLEAN;
BEGIN
  done := FALSE; p := sparseMat[i, matrix];
  WHILE (p <> NIL) AND NOT done DO
    IF p^.col < j THEN p := p^.nextCol
    ELSE done := TRUE;
  IF p = NIL THEN Sparse := defaultValue[matrix]
  ELSE IF p^.col > j THEN Sparse := defaultValue[matrix]
  ELSE Sparse := p^.data
END (* Sparse *);

PROCEDURE PutSparse(i, j, matrix : WORD;value : dataType);
VAR p, q, prev : nodePtr; done : BOOLEAN;
BEGIN
  prev := NIL; done := false;
  p := sparseMat[i, matrix];
  WHILE (p <> NIL) AND NOT done DO
    IF p^.col < j THEN BEGIN
        prev := p; p := p^.nextCol
      END
    ELSE done := TRUE;
  IF (p^.col = j) AND (value = defaultValue[matrix]) THEN
    BEGIN
      IF prev = NIL THEN
        sparseMat[i,matrix]:=sparseMat[i, matrix]^.nextCol
      ELSE prev^.nextCol := p^.nextCol;
      Dispose(p)
    END
  ELSE IF p^.col = j THEN p^.data := value
  ELSE IF value <> defaultValue[matrix] THEN BEGIN
      New(q); q^.data := value; q^.col := j;
      IF prev = NIL THEN BEGIN
          q^.nextCol := sparseMat[i, matrix];
          sparseMat[i, matrix] := q
        END
      ELSE BEGIN
          q^.nextCol := prev^.nextCol;
          prev^.nextCol := q
        END
    END
END (* PutSparse *);

BEGIN
  FOR j := 1 TO MAXSPARSEMATRIX DO BEGIN
      defaultValue[j] := DEFAULTVAL;
      FOR k := 1 TO MAXROW DO
        sparseMat[k, j] := NIL
    END
END.
```

♦ ♦ ♦

Figure 2—PROGRAM SparseExample;

```
USES SparseMatrix;
CONST MatA = 1; MatB = 2; MatC = 3;
VAR i, j, k : WORD;

PROCEDURE AddMatrix; (* MatA + MatB - MatC *)
BEGIN
  FOR i := 1 TO 10 DO
     FOR j := 1 TO 10 DO BEGIN
        PutSparse(i, j, MatA, i*1.0);
        PutSparse(i, j, MatB, j*2.0);
        PutSparse(i, j, MatC, Sparse(i, j, MatA)+
                              Sparse(i, j, MatB))
      END;
  WriteLn;
  WriteLn('MatA + MatB = MatC');
  FOR i := 1 TO 10 DO
    BEGIN
      FOR j := 1 TO 10 DO
        Write(Sparse(i, j, MatC):7:1);
      WriteLn
    END
END (* AddMatrix *);

PROCEDURE TransposeMatrix; (* MatB into MatA *)
BEGIN
  FOR i := 1 TO 10 DO
    FOR j := 1 TO 10 DO
      PutSparse(j, i, MatA, Sparse(i, j, MatB));
  WriteLn; WriteLn('MatB');
  FOR i := 1 TO 10 DO BEGIN
      FOR j := 1 TO 10 DO
        Write(Sparse(i, j, MatB):7:1);
      WriteLn
    END;
  WriteLn; WriteLn('MatA');
  FOR i := 1 TO 10 DO BEGIN
      FOR j := 1 TO 10 DO
        Write(Sparse(i, j, MatA):7:1);
      WriteLn
    END
END (* TransposeMatrix *);

PROCEDURE MultMatrix; (* multiply MatB * MatA = MatC *)
BEGIN
  FOR i := 1 TO 10 DO
    FOR j := 1 TO 10 DO BEGIN
        PutSparse(i, j, MatC, 0.0);
        FOR k := 1 TO 10 DO
          PutSparse(i, j, MatC, Sparse(i, j, MatC) +
            (Sparse(i, k, MatB) * Sparse(k, j, MatA)))
      END;
  WriteLn; WriteLn('MatB * MatA = MatC');
  FOR i := 1 TO 10 DO BEGIN
      FOR j := 1 TO 10 DO
        Write(Sparse(i, j, MatC):7:1);
      WriteLn
    END
END (* MultMatrix *);

BEGIN
  AddMatrix;
  TransposeMatrix;
  MultMatrix
END.
```

♦ ♦ ♦

SHARE WARE

Getting A Little Hyper

Anthony Barcellos
P.O. Box 2249
Davis, CA 95617-2249
Voice: (916) 756-4866
Data: (916) 758-1002

Tony is hyper about hypertext (he uses it to improve his manual dexterity). This time he also talks about his mini-BBS and about the Association of Shareware Professionals. These are three good connections for anyone considering a shareware future.

In the beginning was text, and it didn't seem all that bad at the time. Then came formatting and everyone saw that it was good. When graphics were mixed in, the real excitement began. But there's yet another step in this progression from text editors to word processors to desktop publishers.

It's called hypertext. It offers text, graphics, and freedom—with a bit of confusion tossed in. Unlike the documents most of us are used to, hypertext does not proceed in a straightforward linear fashion. Forget about beginnings, middles, and ends. Hypertext can go off in any direction.

Say you're reading a hypertext document. (Clearly this is not a hard copy system.) A particular phrase or caption catches your attention. A special symbol flags it, indicating that more information is available. You click on the symbol and a window opens. It may be a technical citation or reference. Perhaps it's an illustration or graph. It might even be an extensive digression, containing further branches of its own.

A hypertext document can be perused on several levels, tailored to each reader's individual needs and preferences. A novice, consulting a hypertext user's manual, could ask for additional help on difficult procedures.

An expert could lightly skim or, alternatively, deeply dig into technical details that would frighten off a beginner. Using symbol keys and internal "links," a hypertext author can create a file that approaches all things to all people.

That New Black Magic

NTERGAID has released a shareware hypertext authoring system called Black Magic. It includes a word processor to create your text, a screen grabber for importing graphics from other programs, and a public domain reader to distribute with the hypertext.

You can create four types of linkages within Black Magic. A simple Note link pops up a window that displays a short text string. Use Note links to clarify comments or add little digressions.

Replacement links are quite different. Unlike Note links, which the user may examine (or not) on a case-by-case basis, Replacement links are typically organized in families scattered throughout a document. When a Replacement link of a particular kind activates, all other Replacement links of the same name also turn on. Replacement links substitute something for each occurrence of the link word.

For example, "Boot your computer." could be uniformly replaced with "If your computer has a hard disk and is turned off, turn it on; if it's already turned on, press the keys Ctrl, Alt, and Del simultaneously. If your computer does not have a hard disk, place your DOS diskette in drive A and turn your computer on; if it is already turned on, place your DOS diskette in drive A and press the keys Ctrl, Alt, and Del simultaneously."

How's that for customizing the level of your documentation?

Reference links can take you from one hypertext document to another, thereby blurring the boundaries of individual hypertext files. So you don't get totally lost, the Reference link gives you a flowchart map of the document.

Black Magic's DOS link lets you shell out to a new program while Black Magic shrinks into a tiny memory-resident kernel. When you exit the DOS program, Black Magic resumes control. Thus you can build a highly customized DOS shell and help system.

In their press kit, NTERGAID says that DuPont is using hypertext as the documentation component of an active order system. The Environmental Protection Agency uses Black Magic to organize its voluminous regulations, and NASA uses it to distribute aerospace test data.

Despite its power, Black Magic's system requirements are modest. Although it prefers a hard disk, Black Magic can manage with a dual-floppy PC. The minimum RAM is 384K, although 640K is recommended.

Black Magic requires a graphics card; it supports CGA, Hercules, EGA, and VGA. A Microsoft mouse (or compatible) is optional. It supports Epson dot-matrix and Hewlett-Packard LaserJet printers for printed output. (Of course, Black Magic loses most of its hypertext features when you print a document. You can, of course, choose which part of a hypertext file to print, so Notes and such can be included.)

That old Black Magic doesn't have us under its speller, because it doesn't check your writing. They left that for release 2.0. Overall, however, the package is impressive.

NTERGAID suggests several applications for Black Magic, including technical documentation, educational courseware, computer-based training, and presentations. The clever user can probably come up with several original applications for this combination word processor/DOS shell/clip art gallery/reference system.

If you're eager to try it, you can download Black Magic from CompuServe or from NTERGAID's own HyperBoard at (203) 366-5698. There are also several payment levels, as noted below.

Black Magic, version 1.4
NTERGAID
2490 Black Rock Turnpike, Ste. 337
Fairfield, CT 06430
(203) 368-0632 (Voice)
(203) 366-5698 (BBS)

Demo disk, $7.00
Trial disk set (unregistered) $18.95
Registered copy
(with technical support), $49.95
Registered copy (with technical
support, printed manual, keybrd
template, quick reference card)
$89.95

A Heavenly Host

You're at the office. Your file is on your PC at home. You need it *now*. What to do?

If you have Minihost running on your system at home, you just call it up via modem and download the file to your office computer.

Minihost is by no means unique, since communications programs like ProComm offer a host mode for unattended operation. However, Minihost is more like a small bulletin board system with amazingly low maintenance requirements. Mankin's program comes practically ready to run as soon as you extract the components from the archive file and copy them to a Minihost directory on your hard disk. (Minihost supports floppies, too, but a hard disk is recommended.)

You'll need to edit Minihost's parameters, but they've thrown in numerous example entries. Just enter your name in place of Don Mankin's (the author) as sysop, and edit the names in the user list to give your friends suitable access levels. Or you can lock everyone else out by declaring your Minihost a closed system.

You also get to set disk and directory protection levels. You can change the COM port from 1 to 2 (or whatever). Or, you can prevent others from downloading files with specified extensions.

The documentation explains each option in the order in which it occurs in the .CNF file, so I loaded the user's manual into one window of my word processor and MINIHOST.CNF into the other. As I scrolled through the manual, I moved through the parameters.

I have been using Minihost for over two years so folks can send me articles for *Sacra Blue*, the newsletter I edit. In the olden days, contributors would call and we'd attempt to get my Qmodem talking to whatever they had. Then we'd patiently go through the file transfer procedure.

Now I leave Minihost running and *Sacra Blue's* writers can upload files just about whenever they please. I don't have to do a thing. Minihost lets them leave messages to me (or to each other, since recent Minihost releases offer message conferences as well as messages to the sysop) and I can leave messages for them.

My system is configured so that anyone can log in to leave messages or upload files. I have entered a few people in my user base so that they can download files and access additional directories. Minihost is very flexible.

The simplest way to receive Minihost is to call Don Mankin's Minihost system at (209) 836-2402 and download the current version. If you do, you'll also discover that Don is in the process of creating a new system called Maxihost. He's converting to Turbo Pascal 5.5, thus shaking off the 64K limit.

Since Minihost has always served me well in its "mini" form, I'm not sure I'll jump into Maxihost. In any case, I can attest after years of reliable use that Mankin's little BBS is a gem of a system.

Minihost
Don Mankin
3211 Crow Canyon Place, #A296
San Ramon, CA 94583
(209) 836-2402 (BBS)

$25 hobbyist registration fee
$50 commercial registration fee

Shareware Sources

The Association of Shareware Professionals has already done a great deal to professionalize the shareware business. Many shareware authors have turned to ASP for information on entering this alternative software marketplace.

ASP also has other functions. ASP has an ombudsman to mediate disputes between users and ASP members. In addition to writing to the ombudsman at ASP's address, you can reach him via CompuServe. His ID number is 70007,3536.

ASP has also published vendor guidelines. Vendors who give shareware authors due credit and adhere to the ASP guidelines may describe themselves as "ASP approved."

As a shareware aficionado, I'm always curious where people obtain their shareware. Although users groups are a primary source for many people, the larger mail-order operations generally have more comprehensive inventories and many now carry the ASP seal of approval.

What's your favorite shareware source? I've mentioned PC-SIG and Nelson Ford's Public (Software) Library in the past. Let me know your experiences with shareware vendors and I'll report on which ones seem to be the best bets for quick and friendly service. Just write to me or leave a message on my Minihost system. Keep those bytes coming.

ASP Ombudsman
Assoc. of Shareware Professionals
P.O. Box 5786
Bellevue, WA 98006
CompuServe 70007,3536

♦ ♦ ♦

Order And Chaos:

Creating Fibonacci Attractors With Turbo Pascal Objects

By Gary Entsminger
P.O. Box 2091
Davis, CA 95617

My last conversation with Gary began with: "How in the heck am I going to introduce this kind of chaos?" (Actually, I didn't say "heck," but you understand.)

"You could tell them you assumed it was a Culture Corner until you got to the very end," he said.

"The end wouldn't have straightened me out," I muttered.

So here I am, still no introduction. (And, come to think of it, I still don't have a Culture Corner.)

Leonardo of Pisa created one of the first models of population growth in 1220, 7.69 centuries ago. Leonardo, better known to us as Fibonacci (renamed by mathematical historian, Guillaume Libri, in the 19th century), used a pair of rabbits as the basic unit for reproduction in his model. He reasoned the following—

(1) begin with one pair of rabbits;
(2) let them mature for one year;
(3) let them beget one pair like themselves;
(4) for each successive season, let each new (mature!) pair beget a new (immature!) pair;
(5) continue as far as integers (or rabbits) go.

Figure 1 shows how this scheme develops.

We call the pattern in the second, third, and fourth columns the Fibonacci sequence, and it has challenged and perplexed mathematical minds for almost eight centuries. The pattern's simple enough—*each successive number is equal to the sum of the two preceding numbers.*

Figure 1—Rabbit Population Model

Season	Mature-pairs	Immature-pairs	Total
1	0	1	1
2	1	1	2
3	2	1	3
4	3	2	5
5	5	3	8
6	8	5	13
7	13	8	21
8	21	13	34
9	34	21	55
10	55	34	89

♦ ♦ ♦

As a population model, the Fibonacci sequence has a basic flaw: it omits death. Consequently, rabbits grow by leaps and bounds (hops, actually), eventually filling the surface of the earth.

Fibonacci wasn't all that interested in overrunning the planet with rabbits; he wanted to understand patterns of growth. His sequence, it turns out, fits a very interesting ratio contemplated in Greece some 15 centuries before Fibonacci.

The Golden Ratio

The golden ratio (½(1 + sqrt(5)) or roughly 1.618034... is the ratio between two divisions of a line or plane, such that the smaller is to the larger as the larger is to the sum of the two.

The golden ratio recurs throughout nature—

- it's roughly the ratio of your total height to your navel's height. (If you don't believe me, measure yourself; even better, measure a good friend.)
- it accurately models the "bee-tree law," or pattern of reproduction in bee colonies (where a single female parent *asexually* reproduces males, and male and female parents together *sexually* reproduce females);
- it describes the number of ways light can be refracted and reflected through different layers of glass;
- it describes the arrangement of florets in sunflowers and other composites.

A typical sunflower, for example, has a head containing spirals of tightly packed florets. Each spiral winds a fibonacci number of florets in one direction and the next smaller (or larger) fibonacci number of florets in the other. (See Figure 2.)

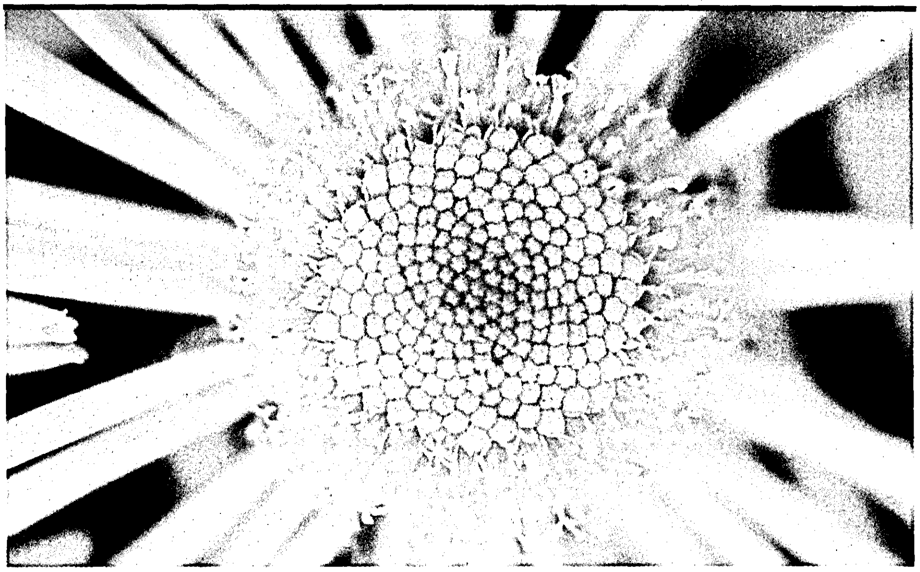


Figure 2—Chrysanthemum Morifolium (elusion). Photo by C. Anderson

Figure 3 —R1=1.61803398867 R2=0.6180339887

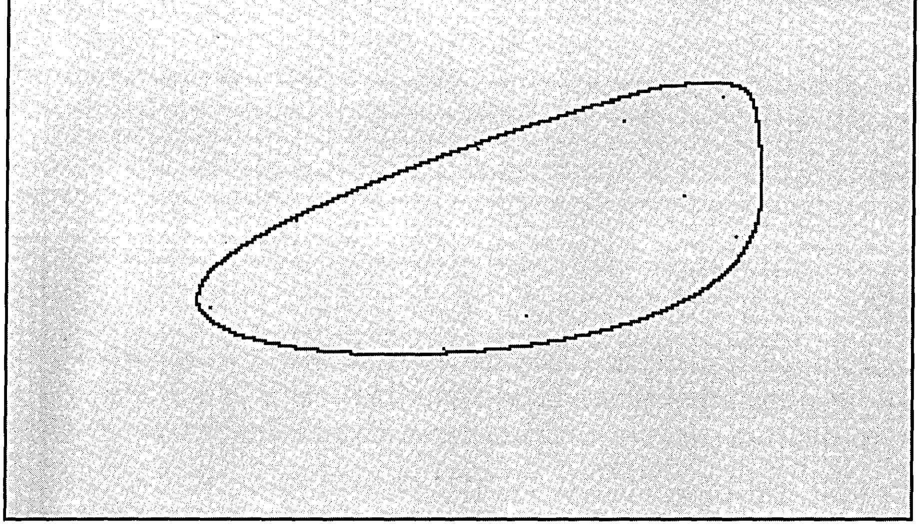
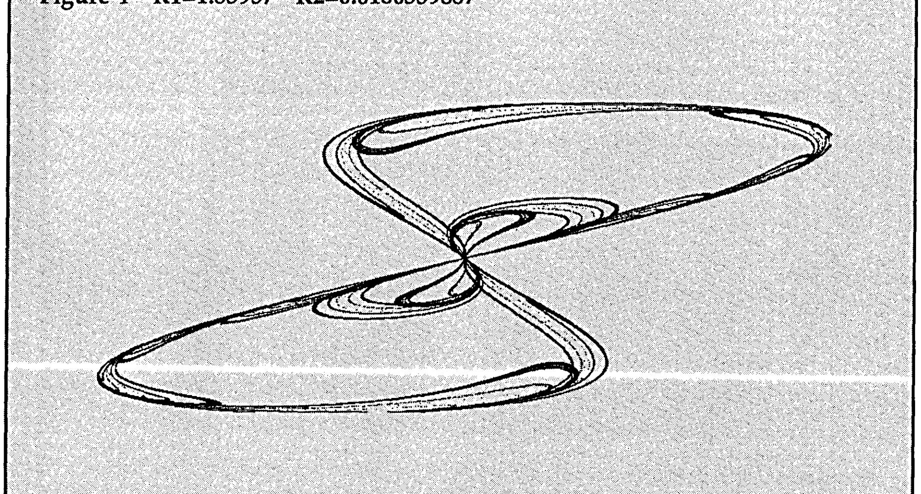


Figure 4—R1=1.83957 R2=0.6180339887



Sunflowers have been found spiraling in all the Fibonacci numbers up to 144.

The late summer hillside below (and for miles around) my writing table here in the Elk mountains glows golden with blooming composites. A hillside fractal of Fibonacci spirals.

Growth

Most models of population growth share a similar form—

$$X_{n+1} = F(X_n)$$

where X_n is the density in season n; X_{n+1} is the density in season n+1; and F is a mapping which shows how the density changes from season to season (or state to state).

F has two parts: one part represents birth rate, and one part represents death rate. Most of these mappings (at least most of the published ones) have one significant trait in common: they define a single-humped (i.e., parabolic) curve.

Qualitatively, they behave like the much-discussed logistic equation (which I briefly discussed in *Micro C*, #47, "Strange Attractors: Order In Chaos"). In particular, these mappings all exhibit both ordered and chaotic behavior depending on their initial conditions.

I thought it might be fun to explore a population growth model based on the Fibonacci sequence. The equation I chose to model is simple enough, with a rate parameter for birth and a second rate parameter for death—

$$X_{n+1} = R1 * X_n * (1 - R2 * (X_{n-1} * X_{n-1}))$$

where R1 is the birth rate of this generation and R2 is the death rate of the last generation.

The Fibonacci twist is twofold:

(1) as a possible rate (for R1 or R2);

(2) as a three generation component: future generations (or states) depend on both this generation (birth rate) and the generation before (death rate).

Not surprisingly, the Fibonacci sequence (expressed as the golden ratio) produces dramatically different behavior (when used as a rate) than values very, very near the ratio. See the attractors in Figures 3 and 4 for immediate gratification. Later I'll talk results.

Prolog & Pascal

Usually I use Turbo Prolog for modelling systems, primarily for three reasons—

(1) Models are state systems (i.e., they're recursive). The value of the last state gets fed back into the system to produce the next state. Prolog is a recursor's dream;

(2) The BGI (Borland Graphics Interface);

(3) Speed.

Lately, though, I've been programming about half-time in Turbo Pascal, first beta-testing TP 5.5 and lately beta-testing Turbo Power's Object Pascal Toolbox. The new object extensions make TP a very intriguing compiler, so I couldn't resist an opportunity to test Turbo Pascal objects for chaos.

After a bit of thought and a few half-blind alleys, I decided to create two objects:

(1) a graphics object; to initialize and test the adapter; to set and modify scales; to setup basic variables such as screen coordinates.

(2) and a Fibonacci model object which inherits all the graphic and scaling methods from the graphics object.

The Fibonacci model object includes a recursive state-changing method and a menu.

Pascal Objects

I've described the objects and their methods in Figure 5. For convenience, I created and compiled the graphics object first as a unit. The Fibonacci object (created later) uses the graphics unit.

Objects are handy here for several reasons:

(1) It's easy to view an object as an entity with an internal state that it remembers (about itself!) even when you aren't using it;

(2) The graphics object is generic, so I can create many more complex objects from it later (by inheriting its methods);

(3) Since an object's methods know all about its data fields (i.e., its state), I don't have to pass parameters between the object's methods;

(4) Objects group data and methods (which might otherwise be confusing) in distinctive blocks.

My code is simple enough, so I'll only comment here on a few interesting (I think) nuances.

My first semi-blind alley generated one of Turbo Pascal's stack overflow errors. The problem—too much recursion. Even a 64K stack isn't enough for cranking out strange attractors. In Turbo Prolog, you can recurse 'til the cows go

Figure 5—OOF! (Object Oriented Fibonacci)

```
unit att_graph;
interface
uses crt, graph;

type
    scale = record
              Xmin,Xmax,Ymin,Ymax : double;
            end;
    gptr = ^graphics;
    graphics = object
       Maxx, Maxy, grDriver, grMode, grError: integer;
       Ix, Iy: longint;
       X, Y, Nx, Ny: double;
       current_scale : scale;
       constructor init;
       destructor done; virtual;
       function initialize: boolean;
       procedure calculate_scale;
       procedure change_scale;
    end;

implementation

constructor graphics.init;
begin
end;

destructor graphics.done;
begin
end;

function graphics.initialize: boolean;
begin
     grDriver := Detect;
     InitGraph(grDriver, grMode, '');
     grError := GraphResult;
     IF grError <> GrOK
     THEN
         begin
           Writeln('Graphics error : ',GraphErrorMsg(grError));
           initialize := false;
         end
     ELSE
         begin
           Maxx := getMaxx;
           Maxy := getMaxy;
           initialize:= true
         end;
end;

procedure graphics.calculate_scale;
var
   Lx, Ly, Xunits, Yunits: double;
begin
   Xunits := current_scale.Xmax - current_scale.Xmin;
   Yunits := current_scale.Ymax - current_scale.Ymin;
   Lx    := (Maxx/Xunits) * (X - current_scale.Xmin);
   Ly    := (Maxy/Yunits) * (Yunits - (Y - current_scale.Ymin));
   Ix := round(Lx);
   Iy := round(Ly);
   putpixel(Ix,Iy,7);
end;

procedure graphics.change_scale;
var
   Rs: string; R: real; Return: integer;
begin
   writeln('Xmin = ',current_scale.Xmin);
   writeln('Xmax = ',current_scale.Xmax);
   writeln('Ymin = ',current_scale.Ymin);
   writeln('Ymax = ',current_scale.Ymax);
   writeln('New Xmin: ');
   readln(Rs);
   Val(Rs,R,Return);
```

Continued on page 85

```
    IF Return <> 0 THEN
       writeln('Invalid input Xmin unchanged')
    ELSE
       current_scale.Xmin := R;
    writeln('New Xmax: ');
    readln(Rs);
    Val(Rs,R,Return);
    IF Return <> 0 THEN
       writeln('Invalid input Xmax unchanged')
    ELSE
       IF R > current_scale.Xmin THEN
          current_scale.Xmax := R
       ELSE
          writeln('Xmax must be > Xmin: no change');
    writeln('New Ymin: ');
    readln(Rs);
    Val(Rs,R,Return);
    IF Return <> 0 THEN
       writeln('Invalid input Ymin unchanged')
    ELSE
       current_scale.Ymin := R;
    writeln('New Ymax: ');
    readln(Rs);
    Val(Rs,R,Return);
    IF Return <> 0 THEN
       writeln('Invalid input Ymax unchanged')
    ELSE
       begin
          IF R > current_scale.Ymin THEN
             current_scale.Ymax := R
          ELSE
             writeln('YMax must be > Ymin');
       end;
end;
end. {unit att_graph}
```

```
program fibAttract;

uses crt, graph, att_graph;

type
     fptr = ^fibonacci;
     fibonacci = object(graphics)
       Rate1, Rate2: double;
       Ch: char;
       constructor init;
       destructor done; virtual;
       procedure init_x_y;
       procedure init_rates;
       procedure fib;
       procedure recurse_fib;
       procedure get_rate;
       procedure menu;
       procedure eval_menu;
     end;

var
      f: fptr;

constructor fibonacci.init;
begin
end;

destructor fibonacci.done;
begin
end;

procedure fibonacci.init_x_y;
begin
  X := 1;
  Y := 0;
end;

procedure fibonacci.init_rates;
begin
  Rate1 := 1.6180339887; {Golden ratio}
  Rate2 := 0.6180339887; {2nd root -- Golden ratio}
end;

procedure fibonacci.fib;
begin
  WHILE KeyPressed <> True DO
     begin
       Nx := Rate1 * X * (1 - (Rate2 * Y * Y));
       Ny := X;
       calculate_scale;
       X := Nx;
       Y := Ny;
     end;
end;

procedure fibonacci.get_rate;
var
   Rs: string; Return: integer; R: double;
begin
   writeln('Equation: Nx := R1*X*(1 - (R2*Y* Y))');
   writeln('Current R1: ',Rate1);
   writeln('Enter new R1:');
   readln(Rs);
   Val(Rs,R,Return);
   IF Return = 0 THEN Rate1 := R;
   writeln('Current R2: ',Rate2);
   writeln('Enter new R2:');
   readln(Rs);
   Val(Rs,R,Return);
   IF Return = 0 THEN Rate2 := R;
end;

procedure fibonacci.recurse_fib;
begin
  WHILE KeyPressed <> True DO
```

Continued on page 87

Continued from page 85

```
        fib;
    end;

    procedure fibonacci.menu;
    begin
       writeln('<G>o   <C>hange scale  <Q>uit');
       Ch := ReadKey;
    end;

    procedure fibonacci.eval_menu;
    begin
       case Ch of
         'c' : change_scale;
         'q' : exit;
         'g' : begin
                 get_rate;
                 init_x_y;
                 IF initialize = TRUE THEN
                   recurse_fib
                 ELSE
                   begin
                     OutText('False');
                     Readln
                   end;
                 Closegraph
               end;
               else exit;
           end;
       end;

begin
   New(f,init);
   WITH f^ DO
     begin
        current_scale.Xmin := -1.5;
        current_scale.Xmax :=  1.5;
        current_scale.Ymin := -1.5;
        current_scale.Ymax :=  1.5;
        init_rates;
        Ch := 'z';
        WHILE Ch <> 'q' DO
          begin
            menu;
            eval_menu;
          end;
        dispose(f,done);
     end;
end.
```

♦ ♦ ♦

home by inserting a cut (!) before the recursive call. Called tail recursion elimination, this is a beautiful technique.

In Turbo Pascal, there's no obvious way to insert a cut. But there is a subtle way to recurse 'til your heart's content.

My first (unsuccessful) attempt at a recursive method went something like this: (note: this procedure belongs to the object, fibonacci, hence it's one of fibonacci's methods)

```
procedure fibonacci.fib;
begin
  create_a_strange_attractor;
  WHILE KeyPressed <> True DO
    fib;
end;
```

This code generates a stack overflow error long before it's created much of an attractor.

My second (successful) attempt uses two procedures—

```
procedure fibonacci.recurse_fib;
begin
  WHILE KeyPressed <> True DO
    fib;
end;

procedure fibonacci.fib;
begin
  create_a_strange_attractor;
end;
```

There's no longer a stack overflow problem, since the stack is reclaimed between each call to fib. This sequence runs all night or until a KeyPressed. And there's no need for a large stack (a few K will do).

Note in Figure 5 the use of New and Dispose to allocate and deallocate heap space for objects. And the use of the caret (^) for referencing dynamically allocated objects.

Note also that init and done do nothing besides internal notekeeping. Specifically, they set up and reference the VMT (virtual memory table), which guarantees that the right amount of memory will be allocated on and deallocated from the heap.

Results

The results of my explorations were revealing. The attractors do vary depending on subtle changes in birth and death rates. If we use the golden ratio, the attractor is periodic and highly predictable (see Figure 3).

If we vary rates ever so slightly away from the golden ratio, the attractors vary ever and ever more strangely, looping and looping between and around its loops. (See Figure 4).

Incidentally, the data generated by the Fibonacci model equation plotted as a time series (i.e., each successive state versus time) shows a mass of chaotic dots. The two attractors in Figures 3 and 4 are phase portraits, generated by plotting one generation's data against the last generation's data, rather than against time. (See *Micro C* #47, "The Last Page" or *Dynamics, The Geometry Of Behavior*, by Abraham & Shaw for details.)

The Fibonacci sequence clearly brings an intriguing (if not yet fully illuminating) order to chaos. And Turbo Pascal objects offer a potentially elegant solution to the programming of complex models. I'll get back to you.

And that, friends, is Tidbits.

References

Abraham, Ralph & Christopher Shaw; *Dynamics, Vols. 1-4*; 1982, '83, '85, '88; Aerial Press. (An illustrative guide to attractors, basins, tangles, and the complex mathematics of dynamics. Highly recommended.)

Entsminger, Gary; "Strange Attractors, Order In Chaos," *Micro C #47*; May-June 1989. (A short introduction, with Turbo Prolog code, to strange attractors.)

Graham, Ronald; Knuth; Patashnik; *Concrete Mathematics*; Addison-Wesley; 1989. (Contains an excellent discussion of the Fibonacci sequence.)

Stevens, Peter S.; *Patterns In Nature.* (Excellent examples of how nature appears to an architect.)

Stewart, Ian; *Does God Play Dice? The Mathematics Of Chaos*; Basil Blackwell; 1989. (A terrific introduction to the mathematical side of chaos. Includes many recent experimental discoveries & discussion of recent applications.)

♦ ♦ ♦

Continued from page 14

```
                0,-1, ny-1,0,-1,incmode,fillcolor,
                edgecolor);
              break;
      case 4 : (horangle > 7.0*PI/4.0) ? (incmode = 1):
                (incmode = 0);
              dosurf(xmin,xmax,ymin,ymax,nx,ny,z,0,
                nx-1,1, ny-1,0,-1,incmode,fillcolor,
                edgecolor);
              break;
    }
  if (box) {
      invert ? drawboxbottom(boxedgecol) :
        drawboxtop(boxedgecol);
      drawboxfront(quadrant,boxedgecol);
    }
  free(Xa); free(Ya);
  return(0);
} /* int surface() */

/* converts from decimal degrees to radians. */
void degrees_to_rads(float * horangle,float *elangle)
{
/* check that angles are in the range 0 to 360: */
  do
    if (*horangle < 0.0) *horangle += 360.0;
  while (*horangle < 0.0);
  do
    if (*horangle > 360.0) *horangle -= 360.0;
  while (*horangle > 360.0);
  *elangle  = *elangle  * PI / 180.0;
  *horangle = *horangle * PI / 180.0;
}  /* void degrees_to_rads() */

/* finds standard math quadrant of viewer */
int get_quadrant(float horangle)
{
  if ((horangle >= 0.0) && (horangle < PI/2.0))
    return(1);
  else if ((horangle >= PI/2.0) && (horangle < PI))
    return(2);
  else if ((horangle >= PI) && (horangle < 3*PI/2.0))
    return(3);
  else return(4);
} /* int get_quadrant() */

/* finds global transformation angles and their cosines
   and sines for later use in calls to axonometric().*/
void transform_angles(float horangle,float elangle,
                      int quadrant)
{
  float phi,psi,sinel;

/* avoid dividing by 0. all this messing around with
   quadrants is because angles that generate cosines of
   0.0 are a function of the floating point library of
   the compiler and of round-off error. */
  if (cos(horangle) == 0.0) /* horiz angle 90 or 270 */
    switch (quadrant) {
        case 1 :
        case 3 : phi = PI/2.0;
                 psi = 0.0;
                 break;
        case 2 :
        case 4 : phi = -PI/2.0;
                 psi =  PI;
                 break;
      }
  else {
      phi = atan(sin(horangle)/cos(horangle));
      psi = PI/2 - phi;
    }
  sinel  = sin(elangle); Sinphi = sinel * sin(phi);
  Cosphi = cos(phi); Sinpsi = sinel * sin(psi);
  Cospsi = cos(psi); Cosel  = cos(elangle);
```

```
} /* void transform_angles() */

/* finds maxima and minima of projected real coords.
   stores axonometric projected box corner values in
   global arrays Yb,Zb,Yt,Zt. */
void get_max_min(float xmin,float xmax,float ymin,
                 float ymax,float zmin,float zmax,
                 float *yminp,float *ymaxp,
                 float *zminp,float *zmaxp)
{
  float ytemp,ztemp; float xb[5],yb[5];
  int i;

  *ymaxp = *zmaxp = -BIGNUM;
  *yminp = *zminp =  BIGNUM;


/* max and min defined by extremes of the function: */
  xb[0] = xmin;         yb[0] = ymin;
  xb[1] = xmin;         yb[1] = ymax;
  xb[2] = xmax;         yb[2] = ymax;
  xb[3] = xmax;         yb[3] = ymin;
  xb[4] = xmin;         yb[4] = ymin;
/* find the maxima and minima */
  for (i = 0; i < 5; i++) {
      /* do the bottom: */
      ytemp = yb[i]; ztemp = zmin;
      axonometric(xb[i],&ytemp,&ztemp);
      if (ytemp > *ymaxp) *ymaxp = ytemp;
      if (ytemp < *yminp) *yminp = ytemp;
      if (ztemp > *zmaxp) *zmaxp = ztemp;
      if (ztemp < *zminp) *zminp = ztemp;
      /* store the extremes for later box drawing: */
      Yb[i] = ytemp; Zb[i] = ztemp;
      /* now do the top: */
      ytemp = yb[i]; ztemp = zmax;
      axonometric(xb[i],&ytemp,&ztemp);
      if (ytemp > *ymaxp) *ymaxp = ytemp;
      if (ytemp < *yminp) *yminp = ytemp;
      if (ztemp > *zmaxp) *zmaxp = ztemp;
      if (ztemp < *zminp) *zminp = ztemp;
      /* store the extremes for later box drawing: */
      Yt[i] = ytemp; Zt[i] = ztemp;
    }
} /* void get_max_min() */

/* does axonometric transformation from 3D real coords
   to 2D real coords. result returned in y,z. */
void axonometric(float x,float *y,float *z)
{
  float ytemp;

  ytemp = *y;
  *y = ytemp * Cosphi - x * Cospsi;
  *z = -ytemp * Sinphi - x * Sinpsi + *z * Cosel;
}  /* void axonometric() */

/* draws 3D surface using painter's algorithm. arrays
   indexed according to original gridded surface. */
void dosurf(float xmin,float xmax,float ymin,
            float ymax,int nx,int ny,float **z,int i0,
            int i1,int inci,int j0,int j1,int incj,
            int incmode,int fillcolor,int edgecolor)
{
  register int i,j; int k,l;
  int p[2][2],q[2][2]; float x[2][2],y[2][2];
  float ytemp,ztemp,xdif,ydif,xdifp,ydifp;

  xdif = (xmax - xmin)/(nx-1);
  ydif = (ymax - ymin)/(ny-1);
  if (incmode)
    { /* draw and fill columns of quadrangles first */
      ydifp = incj * ydif;
      for (i=i0; i!=i1; i+=inci) {
          {                 /* draw the first quadrangle */
            for (k=0; k<2; k++)
              for (l=0; l<2; l++) {
                  x[k][l] = xmin + (i+inci*k) * xdif;
                  y[k][l] = ymin + (j0+incj*l) * ydif;
                  ytemp = y[k][l];
                  ztemp = z[i+inci * k][j0+incj*l];
                  axonometric(x[k][l],&ytemp,&ztemp);
                  transf(ytemp,ztemp,&p[k][l],
                        &q[k][l]);
                  itransf(&p[k][l],&q[k][l]);
                }
            drawfillquadrangle(p,q,fillcolor,
                               edgecolor);
            y[0][0] = y[0][1]; y[1][0] = y[1][1];
          }
        for(j=j0+incj; j!=j1; j+=incj)
          {                /* do the other quadrangles */
            for (k=0;k<2;k++) {
                p[k][0] = p[k][1];
                q[k][0] = q[k][1];
                y[k][1] = y[k][0] + ydifp;
                y[k][0] = y[k][1];
                ztemp = z[i+inci*k][j+incj];
                axonometric(x[k][1],&y[k][1],&ztemp);
                transf(y[k][1],ztemp,&p[k][1],
                      &q[k][1]);
                itransf(&p[k][1],&q[k][1]);
              }
            drawfillquadrangle(p,q,fillcolor,
                               edgecolor);
          }
        }
    }
  else    /* draw and fill rows of quadrangles first */
    {
      xdifp = inci * xdif;
      for (j=j0; j!=j1; j+=incj) {
          {                 /* draw the first quadrangle */
            for (k=0; k<2; k++)
              for (l=0; l<2; l++) {
                  x[k][l] = xmin + (i0+inci*k) * xdif;
                  y[k][l] = ymin + (j+incj*l) * ydif;
                  ytemp = y[k][l];
                  ztemp = z[i0+inci * k][j+incj*l];
                  axonometric(x[k][l],&ytemp,&ztemp);
                  transf(ytemp,ztemp,&p[k][l],
                        &q[k][l]);
                  itransf(&p[k][l],&q[k][l]);
                }
            drawfillquadrangle(p,q,fillcolor,
                               edgecolor);
            x[0][0] = x[1][0]; x[0][1] = x[1][1];
          }
        for (i=i0+inci; i!=i1; i+=inci)
          {                 /* do the other quadrangles */
            {
              for (l=0;l<2;l++) {
                  p[0][l] = p[1][l];
                  q[0][l] = q[1][l];
                  x[1][l] = x[0][l] + xdifp;
                  x[0][l] = x[1][l];
                  ytemp =  y[1][l];
                  ztemp = z[i+inci][j+l*incj];
                  axonometric(x[1][l],&ytemp,&ztemp);
                  transf(ytemp,ztemp,&p[1][l],
                        &q[1][l]);
                  itransf(&p[1][l],&q[1][l]);
                }
            }
            drawfillquadrangle(p,q,fillcolor,
                               edgecolor);
          }
        }
    }
} /* void dosurf() */


/* divides quadrangle into 2 triangles. calls
   filltriangle() to blank the triangle, then draws the
   quad perimeter. p holds integer device x-coords of
   the rectangle corners. q holds y-coords. */
void drawfillquadrangle(int p[2][2],int q[2][2],
                        int filcol,int edgecolor)
{
  filltriangle(p[0][0],q[0][0],p[1][0],q[1][0],p[1][1],
           q[1][1],filcol);
  filltriangle(p[0][0],q[0][0],p[0][1],q[0][1],p[1][1],
           q[1][1],filcol);
  Csetf(edgecolor);
  Linef(p[0][0],q[0][0],p[0][1],q[0][1]);
  Linef(p[0][1],q[0][1],p[1][1],q[1][1]);
```

```
  Linef(p[1][1],q[1][1],p[1][0],q[1][0]);
  Linef(p[1][0],q[1][0],p[0][0],q[0][0]);
}  /* void drawfillquadrangle() */

/* draws triangle described by the 3 points passed. */
void filltriangle(int x0,int y0,int x1,int y1,
                  int x2,int y2,int fillcolor)
{
  int  z,a,b,dx,dy,d,deltap,deltaq,jstart;
  int xl[2][3],yl[2][3]; register int x,y,i,j,k;

  /* sort the points in ascending vertical order: */
  if (y1 < y0) swapcoords(&x1,&y1,&x0,&y0);
  if (y2 < y0) swapcoords(&x2,&y2,&x0,&y0);
  if (y2 < y1) swapcoords(&x2,&y2,&x1,&y1);
  /*stick em in arrays for triangle edge computation:*/
  xl[0][0] = x0;              yl[0][0] = y0;
  xl[0][1] = x1;              yl[0][1] = y1;
  xl[0][2] = x2;              yl[0][2] = y2;
  /* to use loops instead of if statements, we pretend
     the long side of the triangle is made up of 2
     lines. one has zero length: (x2,y2) to (x2,y2). */
  xl[1][0] = x0;              yl[1][0] = y0;
  xl[1][1] = x2;              yl[1][1] = y2;
  xl[1][2] = x2;              yl[1][2] = y2;
  /* i loops over the 2 triangles: k loops over the two
     short sides and the one long side: */
  for (i = 0; i < 2; i++) {
      jstart = 0;
      for (k = 1; k < 3; k++)
        {
          dx = abs(xl[i][k] - xl[i][k-1]);
          dy = abs(yl[i][k] - yl[i][k-1]);
          x = xl[i][k-1]; y = yl[i][k-1];
          if (dy <= dx) {
              z = xl[i][k];
              a = (xl[i][k-1] <= xl[i][k]) ? 1 : -1;
              b = (yl[i][k-1] <= yl[i][k]) ? 1 : -1;
              deltap = dy << 1; d = deltap - dx;
              deltaq = d - dx;
              j = jstart;
              Xa[i][j] = x; Ya[i][j] = y;
              while (x != z) {
                  x += a;
                  (d < 0) ? (d += deltap) :
                     (y += b,d += deltaq,j += 1);
                  Xa[i][j] = x; Ya[i][j] = y;
              }
          }
          else {
              z = yl[i][k];
              a = (yl[i][k-1] <= yl[i][k]) ? 1 : -1;
              b = (xl[i][k-1] <= xl[i][k]) ? 1 : -1;
              deltap = dx << 1; d = deltap - dy;
              deltaq = d - dy; j = jstart;
              Xa[i][j] = x; Ya[i][j] = y;
              while (y != z) {
                  y += a; j += 1;
                  (d < 0) ? (d += deltap) :
                    (x += b,d += deltaq);
                  Xa[i][j] = x; Ya[i][j] = y;
              }
          }
          jstart = dy;
        }
    }
  /* draw the blank horizontal lines: */
  Csetf(fillcolor);
  for (j=0; j < (yl[0][2] - yl[0][0]); j++)
    Linef(Xa[0][j],Ya[0][j],Xa[1][j],Ya[0][j]);
}  /* void filltriangle() */

/* swaps the integer plot device coords of 2 points. */
void swapcoords(int *x1,int *y1,int *x2,int *y2)
{
  int temp;

  temp = *x1; *x1  = *x2; *x2  = temp;
  temp = *y1; *y1  = *y2; *y2  = temp;
}  /* void swapcoords() */

/* transforms 8 box corners to int plot device coords*/
void transformbox()
{
  int i;

  for (i = 0; i < 5; i++) {
      transf(Yb[i],Zb[i],&Pb[i],&Qb[i]);
      itransf(&Pb[i],&Qb[i]);
      transf(Yt[i],Zt[i],&Pt[i],&Qt[i]);
      itransf(&Pt[i],&Qt[i]);
    }
}  /* void transformbox() */

void drawboxbottom(int color)
{
  int i;

  Csetf(color);
  for (i = 0; i < 4; i++)
    Linef(Pb[i],Qb[i],Pb[i+1],Qb[i+1]);
}  /* void drawboxbottom() */

void drawboxback(int quadrant,int color)
{
  int i,j;
  static int back[4][2] = { { 0,1 },
                            { 0,3 },
                            { 2,3 },
                            { 1,2 } };

  Csetf(color);
  for (i=0; i<2; i++) {
      j = back[quadrant-1][i];
      Linef(Pb[j],Qb[j],Pt[j],Qt[j]);
    }
} /* void drawboxback() */

void drawboxtop(int color)
{
  int i;

  Csetf(color);
  for (i = 0; i < 4; i++)
    Linef(Pt[i],Qt[i],Pt[i+1],Qt[i+1]);
}  /* void drawboxtop() */

void drawboxfront(int quadrant,int color)
{
  int i,j;
  static int front[4][2] = { { 2,3 },
                             { 1,2 },
                             { 0,1 },
                             { 0,3 } };

  Csetf(color);
  for (i=0; i<2; i++) {
      j = front[quadrant-1][i];
      Linef(Pb[j],Qb[j],Pt[j],Qt[j]);
    }
}  /* void drawboxfront() */
```

♦ ♦ ♦

TECHTIPS

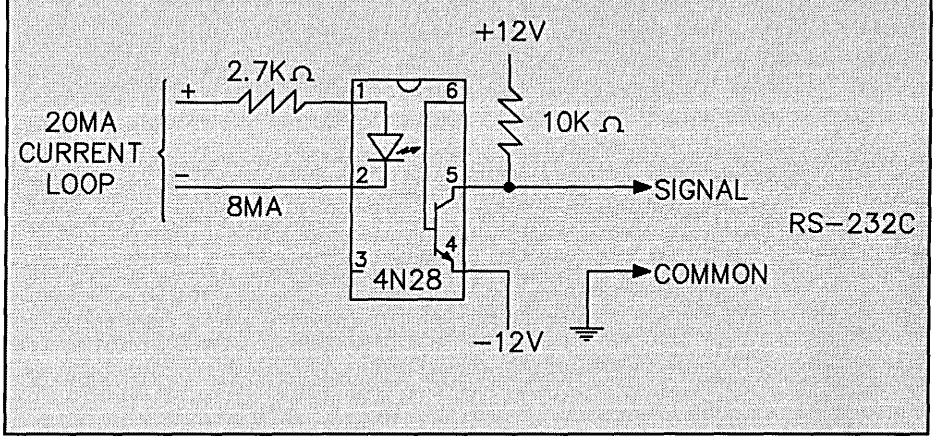
Printer Error Handler

Current Loop To RS-232C Converter

This small circuit will convert a 20 ma signal to RS-232C. I originally built it to convert Reliance Automate 31 program files from our Datapoint to an Allen Bradley T-50 Industrial Terminal. By building and using this converter, I saved countless hours of hand converting files.

The circuit consists of a single 4N28 IC. It decides which voltage (+12 VDC or -12 VDC) reaches the RS-232C port. I put a 2.7K resistor in series with the 4N28's input LED. The resistor limits the LED current to around 8 ma. I use a 10K resistor to pull up the 4N28's output when it's not conducting. The schematic (see Figure 1) shows how I wired the 4N28.

Figure 1—Current Loop to RS-232C



To transmit from the Datapoint, I fired up the 20 ma current loop and loaded the Automate 31 tape (this sets the baud rate). Then I connected the power supply and hooked up the signal to the computer's RS-232C port.

After selecting the same baud rate in my terminal package, I just captured the files as they came in. (Imagine all the typos and all the eye strain of comparing and checking that I avoided.)

Larry Kraemer
Rt. 2 Box 190
Jackson, MO 63755

Printer Error Handler

With all the compilers I've used, there's been one sore point when using the standard printing routines. If there's a problem, you get the ugly "WRITE DEVICE FAULT ERROR," or something similar all over your beautiful screen. I wanted something that could handle the error in a professional way.

I wrote Figure 2 for Borland's Turbo C 2.0. (The routines are not ANSI compatible.) The heart of the code is Turbo C's biosprint() routine.

Let me briefly describe the algorithm. I print strings only; printing floating points and integers is just a matter of converting the number to a string and then printing the string.

First, I send a string to the printing function. It prints out the string character by character, each time checking the status of the printer. If the function detects an error, it moves into the abort/retry function. The abort/retry function saves the contents of the current window. Then it creates its own window, prints the error message, and asks if the user would like to abort or retry.

If the user selects retry, it keeps checking the printer to see if the error has been corrected. If it has, it returns to the printing function and continues printing the string. If the user chooses to abort, it exits the printing function. The abort/retry function restores the previous screen and window coordinates before finishing.

You may notice a little double coding in the functions printer_ok() and bio_print_string(). The routines for checking in bio_print_string() are so small I didn't want to make the extra call to printer_ok(). However, you may want to call printer_ok() to determine the printer status before you start printing. This is the reason for some duplicate code. Also, the abort/retry function uses printer_ok() to keep determining printer status.

You can use these routines universally with all your Turbo C programs. Just compile STDPRINT.H to an .OBJ file and use it in your project files, or put the code in your program as a header file. If you compile STDPRINT.H to an .OBJ file, make sure you include the files BIOS.H, CONSOLE.H and GRAPHICS.H in the program.

I hope these routines will make your life a little easier when dealing with the printer. (Note: These routines do not work for serial printers.)

Jeffrey Scott Donovan
1515 Santa Barbara St. #A
Santa Barbara, CA 93101

Micro Ads

A Micro Ad is the inexpensive way to reach over 22,000 technical folks like yourself. To place a Micro Ad, just print out your message (make it short and sweet) and mail it to Micro C. We'll typeset your ad (no charge) and run it in the next available issue. You can also send camera ready copy. Rates: $99 for 1 time, $267 for three times, $474 for 6 times (a best buy at only $79 per insertion). Full payment must accompany ad. Each ad space is 2 1/4 inches by 1 3/4 inches.

Help System II

Briefly, a hypertext help system, context sensitive, up to 80% compression, requires less than 10K, includes editor for documents, and source for Pascal, C, and Basic.

(203)368 - 0632

NTERGAID, 955 Connecticut Ave. Bridgeport, CT 06607

DUPLICATOR TOOLKIT PRO

THE DUPLICATOR TOOLKIT PROfessional is a superfast diskette duplication utility for your PC. It does what diskcopy can't do! Copies a 360K diskette in 20 seconds, formats and verifies "on the fly," and even produces diskette labels at the same time! Copies 5 1/4 360K DDDS, 1.2M high-density, 5 1/4 360K using a 1.2M high density drive, 3 1/2 720k, 3 1/2 1.44M and 3 1/2 720K using a 1.44M drive. Stores master in RAM or on the hard drive. Format-only option allows for high-speed formatting. Eliminates the "floppy shuffle!"

$129.00 + 5.00 s/h Visa/MC/COD

COPY TECHNOLOGIES
14252 Culver Dr. Ste. 323
Irvine, CA 92714

Reader Service Number 163

F L A S H

THE DISK ACCELERATOR

PC MAGAZINE EDITORS' CHOICE

- EASY to Install
- Cache up to 32 MEGS of EXTENDED and/or EXPANDED
- Buffers up to 26 DEVICE driven drives
- Comes with 2 FREE utilities!!!!!

ORDER NOW $69.95
(800) 25-FLASH

SOFTWARE MASTERS 6352 North Guilford Ave.
Indianapolis, In 46220 / (317) 253-8088

$5.00 Shp/hnd in USA & CANADA, $15.00 overseas.

Reader Service Number 106

LATEST AWARD BIOS

PC/XT ☆ 286 ☆ 386

Support for:

- Enhanced Keyboards
- EGA & VGA Graphics
- 3.5 inch Floppies
- More...

Authorized AWARD Distributor
(800) 423-3400

KOMPUTERWERK, INC
851 Parkview Blvd
Pittsburgh, PA 15215

Reader Service Number 126

STOCKS OPTIONS FUTURES

Turn Your PC Into A MARKET QUOTATION MONITOR

100 page book covers satellite and radio data reception of financial news and quotes for your PC. $19 (includes demo diskette). Free informative catalog of:

* Data receivers and kits
* Quote processing and display software
* Descrambling software utilities

303-223-2120 $5 Demo Diskette

DATArx
111 E. Drake Rd. Suite 7041
Fort Collins, CO 80525

Reader Service Number 133

Technology for the Arts

Music Quest PC Music MIDI
Card with free software $119,
Sequencer or Toolkit $39 each.

Digital Vision B&W Video Digitizer $249
or Color Video Digitizer $399
Willow PC Publisher VGA/frame grabber $595
Everex Vision 8 frame grabber $795
Everex Vision 16 frame grabber $1295

CCTV Video Cameras
Special: Color VCC3700 Sanyo $395
Panasonic WV-1410 B&W $219
Panasonic WV-BL200 CCD B&W $579

Free Shipping! TX residents add sales tax.

joel Sampson Engineering

P.O.Box 550363 Phone 214-328-2730
Dallas, TX75355-0363 BBS 214-328-6909

Reader Service Number 176

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. BATCOM extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, BATCOM protects your source code, and you can distribute your compiled programs without royalties. For IBM PC. Only $59.95. Order today!

Wenham Software Company
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Reader Service Number 124

THOUSANDS OF NEW COMPUTER BOOKS

40% OFF LIST

Get the newest computer titles at the lowest prices with a low minimum order. Send 3 stamp SASE.

COMPUTER BOOKS • PO 70195 • SAN DIEGO, CA 92107

Reader Service Number 167

8031 μController Module $39.95

Shipping: $3.00 US/$5.00 Canada

Ideal for prototypes, one of a kind devices or short production runs. Perfect building block for PC or Macintosh data aquisition interface or stand alone control projects. Assembled and tested board includes 8031 microprocessor, crystal, 8K of EPROM, 128 bytes of RAM, 2 byte-wide I/O ports and provisions for a MAX232 to provide industry compatible serial I/O. All ICs are socketed and I/O is via a 2X17 header. Size: 2.75" by 4.0". OEM discounts.

Cottage Resources
Suite 3-672C, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

Reader Service Number 158

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs
- Uses serial ports and 5 wire cable
- Runs at 115 K baud
- Runs in background, totally transparent
- share any device, any file
- Needs only 14K of ram

Skeptical? We make believers!

Information Modes
P.O. Drawer F
Denton, TX 76202
817-387-3339

Reader Service Number 149

WHITNEY EDITOR $39

Small and fast
Uses all available memory
Split-screen editing
Configurable keyboard
Regular expression search
One key compile
Features for writing documentation
Condensed/Outline display
Runs on IBM PC's, AT's, and PS/2's

USA shipping & handling $3; Outside USA $15
CA residents add sales tax

Whitney Software, Inc.
P.O.Box 4999, Walnut Creek, CA 94596 (415) 933-9019

Reader Service Number 164

16 Megabytes EMS and/or Extended Memory

- Works on 8 or 16 bit bus
- 16 bit transfer on AT bus
- Single board design
- Includes RAM disk and extensive diagnostics
- Quantity/OEM discounts

XT and AT Compatible

Designed, Manufactured, Sold and Serviced by PTech

907 North 6th St. Lake City, MN 55041 (612)345-4555

CROSS ASSEMBLERS

PseudoCode releases the PseudoSam professional series of Cross assemblers. All popular processors. Macros, Conditional Assembly, and Include Files. Virtually unlimited size. For IBM-PC's MS-DOS 2.0 or greater with manual $50.00. Simulators and disassemblers also available. (MI res. 4% tax). S&H USA $5, Canada $10, Foreign $15. Visa/MC.

KORE Inc.
6910 Patterson S.E.
Caledonia, MI 49316
616-887-1444.

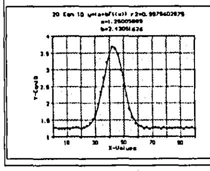
30 Day satisfaction guaranteed or purchase price refunded.

Reader Service Number 136

Figure 2—STDPRINT.H

```
/* This function checks printer status to see if printer is ok. The
   variable pointer prn_status returns status in case further
   examination is needed. If printer ok return 1, otherwise return 0.
   prn_num=0 for LPT1:   1 for LPT2:   etc. */

int printer_ok(int prn_num,int *prn_status)
   {
   int ok;

   /* CHECK STATUS OF PRINTER */
   *prn_status=biosprint(2,'x',prn_num);
   /* IF PRINTER NOT BUSY AND SELECTED, THEN IT'S OK */
   if ( (((*prn_status) & 128)==128) && (((*prn_status) & 16)==16) )
     ok=1;
     else ok=0;
   return(ok);
   }

/* This function is called when a printer error has already been
   detected. It then saves the screen and window, retrys printer,
   prints the approp error, and asks to abort or retry. If aborted
   then the function returns 1, if retry then we keep checking printer
   status until we are ok or user decides to abort. If ok then we
   return 0. Before returning we restore the old screen and window. */

int abort_retry_printer(int prn_num,int *prn_status)
   {
   char ch;
   int time_out,io_err,select_err,no_paper,power_err,busy;  /*ERRORS*/

   int early_exit,ok;           /* TO TEST USER ANSWER, PRINTER    */
   char *screen_0buf[80*25*2];  /* TO SAVE SCREEN TO               */
   struct text_info w;          /* OLD WINDOW COORDINATES          */

   /* SOME SLOW PRINTERS NEED LONGER TO SEND STATUS, IF YOURS IS FAST
      TAKE THIS OUT, BUT IS YOUR CUSTOMER'S? */
   delay(1000);
   printer_ok(prn_num,prn_status);

   /* SAVE THE CURRENT SCREEN    */
   gettext(1,1,80,24,screen_0buf);

   /* SAVE CURRENT WINDOW COORDS */
   gettextinfo(&w);

   /* MAKE NEW WINDOW FOR MESSAGE*/
   window(1,1,80,25);
   textbackground(4);          /* LO RED FOR BACKGROUND ERROR MSG */
   textcolor(0);               /* TEXT COLOR BLACK                */
   window(18,11,62,12);        /* PROGRAMMER CAN DRAW BOX AROUND  */
   clrscr();                   /* WINDOW TO MAKE IT LOOK BETTER.  */
                               /* WE WON'T FOR SIMPLICITY'S SAKE  */

   /* SET STARTUP-END VARIABLES  */
   early_exit=0; ok=0;

   /* PRINT ABORT RETRY MESSAGE  */
   textcolor(0); gotoxy(18,2);
   textcolor(14);cprintf("A");textcolor(1);cprintf("bort, ");
   textcolor(14);cprintf("R");textcolor(1);cprintf("etry ?");
   textcolor(0);

   /* DO THIS ROUTINE UNTIL WE  HAVE ABORTED OR PRINTER IS OK. */
   do
      {

      /* FIGURE OUT WHAT THE ERROR IS      */
      time_out=  ((*prn_status & 1))==1?1:0;
      io_err=    ((*prn_status & 8))==8?1:0;
      select_err=((*prn_status & 16))!=16?1:0;
      no_paper=  ((*prn_status & 32))==32?1:0;
      power_err= ((*prn_status & 64))==64?1:0;
      busy   =   ((*prn_status & 128))==128?1:0;
```

more Micro Ads...




```
    /*IF ANY ARE ON SO IS SELECT. TURN IT OFF.*/
    if ((no_paper)||(power_err)||(busy)||(time_out))
      {select_err=0;io_err=0;}
    if (io_err) select_err=0;

    /* NOW PRINT OUT THE ENTIRE ERROR.    */
    gotoxy(2,1);
    if (time_out)
      cprintf("Time-Out Err or Device Write Err");
    else if (io_err)
      cprintf("I/O Error/Is Cabling Connected?");
    else if (select_err)
      cprintf("Select Error/Is Select Light On?");
    else if (no_paper)
      cprintf("No Paper Error/Check Paper Tray.");
    else if (power_err)
      cprintf("Printer Ack Error/Printer On?");
    else if (busy)
      cprintf("Printer Busy Erro+r/Wait?");
    cprintf("%c",7);
    ch=getch();
    /* IF KEY = A,a THEN ABORT PROCESS    */
    if ((ch==65)||(ch==97)) early_exit=1;
    /* IF KEY = R,r THEN RETRY STATUS     */
    if ((ch==82)||(ch==114))
      ok=printer_ok(prn_num,prn_status);
    }
while( (!early_exit)&&(!ok));

/* PUT ORIGINAL SCREEN BACK    */
puttext(1,1,80,24,screen_0buf);

/* RESET THE ORIGINAL WINDOW AND COLORS */
window(1,1,80,25);
window(w.winleft,w.wintop,w.winright,w.winbottom);
textcolor(w.attribute);
textbackground(w.normattr);
gotoxy(w.curx,w.cury);

/* IF EARLY EXIT, WE ABORTED, ELSE PRINTER OK. */
return(early_exit);
   }

/* This function trys to print a string to the printer.
   If errors occur, goto abort-retry-printer to see if
   user wants to abort. Continue until string printed
   or user aborts. Form feed (\f), CR, etc. work. */

int bio_print_string(const char string[255],int prn_num)
   {
   int byte,loop,status;
   int ok;

   loop=0;

   /* TRY TO PRINT ENTIRE STRING */
   while (loop<=strlen(string))
      {

      /* CHECK PRINTER STATUS    */
      status=biosprint(2,string[loop],prn_num);

      /* IF TIMEOUT OR SEL ERR OR NO PAPER OR IO ERR.
         ASK TO ABORT. IF USER ABORTS, RETURN HERE. */
      if ( ((status & 1)==1) || ((status & 8)==8) ||
           ((status & 16)!=16) || ((status & 32)==32) )
        if (abort_retry_printer(prn_num,&status))
           return(0);
      /* IF STATUS OK AND PRN NOT BUSY, PRINT CHAR
         AND UPDATE LOOP */
      if ((status & 128)==128)
      {
        biosprint(0,string[loop],prn_num); ++loop;
      }
      }
   /* ALL DONE, RETURN TRUE  */
   return(1);
   }
```

♦ ♦ ♦

Complete Your Education ...

Fill Out Your Collection of Micro C today!

ISSUE #1 (8/81)
Power Supply
1/2 PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE #4 (2/82)
Keyboard Translation
More 4 MHz Mods
Modems, Lync, and S10s
Undoing CP/M ERASE
20 pages

ISSUE #5 (4/82)
Two Text Editors
Double Density Review
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K Of RAM
On Your Own Begins
24 pages

ISSUE #8 (10/82)
SOLD OUT

ISSUE #9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
32 pages

ISSUE #10 (2/83)
ISSUE #11 (4/83)
SOLD OUT

ISSUE #12 (6/83)
Bringing Up BBII
Double Sided Drives for BBI
Packet Radio
5 MHz for Kaypro
40 pages

ISSUE #13 (8/83)
CP/M Disk Directory
More 256K for BBI
Mini Front Panel
Cheap Fast Modem
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
BBI Video Size
Video Jitter Fix
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
Upgrading Kaypro II To 4
Upgrading Kaypro 4 To 8
48 pages

ISSUE #16 (2/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE #18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Kaypro Joystick
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE #19 (8/84)
Adding Winchester To BBII
6 MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4 MHz For The 820-I
64 pages

ISSUE #20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans S10
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE #21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE #22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For the STD Bus
Reviews Of 256K RAM Expansion
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Drive Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
68000 Versus 80x86
Soldering: The First Steps
88 pages

ISSUE #25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE #26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
Graphics In Turbo Pascal
104 pages

ISSUE #27 (12/85)
SOLD OUT

ISSUE #28 (2/86)
Rescuing Lost Text From Memory
Introduction To Modula-2
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Prototyping In C
C Interpreters Reviewed
Benchmarking The PCs
104 pages

ISSUE #30 (6/86)
PROLOG On The PC
Expert Systems
Logic Programming
Building Your Own Logic Analyzer
256K RAM For Your 83 Kaypro
PC-DOS For Non-Clones
104 pages

ISSUE #31 (8/86)
RAM Resident PC Speedup
Practical Programming In Modula-2
Unblinking The PC's Blinkin' Cursor
Game Theory In PROLOG and C
104 pages

ISSUE #32 (10/86)
Public Domain 32000: Hardware And Software
Writing A Printer Driver for MS-DOS
Recover A Directory By Reading & Writing Disk Sectors
96 pages

ISSUE #33 (12/86)
ISSUE #34 (2/87)
ISSUE #35 (4/87)
ISSUE #36 (6/87)
ISSUE #37 (9/87)
SOLD OUT

ISSUE #38 (11/87)
Parallel Processing
Laser Printers, Typesetters And Page Definition Languages
Build A Graphics Scanner For $6, Part 2
Writing A Resident Program Extractor In C
96 pages

ISSUE #39 (1/88)
PC Graphics
Drawing The Mandelbrot And Julia Sets
Desktop Graphics
Designing A PC Work-station Board
Around the TMS-34010
96 pages

ISSUE #40 (3/88)
The Great C Issue
11 C Compilers
Writing A Simple Parser In C
C++, An Object Oriented C
Source Level Debugger For Turbo C
96 pages

ISSUE #41 (5/88)
Artificial Intelligence
3-D Graphics
Neural Networks
Logic Of Programming Languages
Applying Information Theory
96 pages

ISSUE #42 (7/88)
Maintaining PCs
Keeping Your Hard Drives Running
Troubleshooting PCs
XT Theory of Operation
Simulating A Bus
Ray Tracing
96 pages

ISSUE #43 (9/88)
Building Databases
Build a C Database
Selecting a dBase III Compatible Compiler
Working with Paradox
Designing Custom PC Cards
Accessing dBase III Plus Records from Turbo Pascal
96 pages

ISSUE #44 (11/88)
Object-Oriented Programming
A Taste of Smalltalk
Actor
Thinking Objectively
Building MicroCad
Peripheral Technology-PT68K-2
Hercules Graphics Printer Dump
96 pages

ISSUE #45 (1/89)
Computer Aided Design
CAD In A Consulting Business
Choosing PCB Layout Systems
Building Circuits With Your Computer
Secrets of Optimization
Finding Bargains in the Surplus Market
MASM 5.1
96 pages

ISSUE #46 (3/89)
Software Tools
The Art of Disassembly
Handling Interrupts With Any C
Hacking Sprint: Creating Display Drivers
Greatest C Compilers
Turning A PC into An Embedded Control System
Practical Fractals
96 pages

ISSUE #47 (5/89)
Robotics
The LIMBO Project
Starting A Robotics Company
How To Write and Use A SystemProfiler
Problem Solving and Creativity
Turn Your XT Into A Controller
Writing Code For Two Operating Systems
96 pages

ISSUE #48 (7/89)
Tools For The Physically Impaired
The Adventure Begins
Selecting A Talking Computer For A Blind Friend
Writing Software For The Blind
File Transfer Via The Parallel Port
The LIMBO Project—Part Two
PCX Compatibility
A 68000-Based Multitasking Kernel
The Very Early Days of Computing
96 pages

ISSUE #49 (11/89)
I/O, I/O...
Build A Computer The Easy Way
PostScriptals
Prog. Logic Controllers
Driving Stepper Motors
Writing TSR Programs
Low Cost I/O For The PC
Interfacing 16-Bit Devices
96 pages

To Order:

Phone: 1-800-888-8087
Mail: PO Box 223
Bend, Oregon 97709

United States,	
Issues #1-34	$3.00 each ppd.
Issues #35-current	$3.95 each ppd.
Canada & Mexico	
All issues	$5.00 each ppd.
Foreign (air mail)	
All Issues	$7.00 each ppd.

MICRO C ADVERTISER'S INDEX

Issue 50

** Contact Advertiser Directly.

When you write for information, please tell these folks you read about their products in **Micro Cornucopia.**

Issue #51

Embedded Systems

- **Voice Capture and Analysis, Part 2**
- **TTL Families Explained**
- **Filling Memory Holes on Your XT**
- **3-D Graphics, Part 2**
- **More LIMBO**
- **Bits From Our Past**

THE LAST PAGE

By Gary Entsminger
P.O. Box 2091
Davis, CA 95617

Dynamics & Objects

Interested in learning more about chaos? (I mean the sophisticated kind, not the kind we have around the office.) Interested in objects? Gary looks at a few good books on the subjects.

In a small bookstore along the main tourist strip of Durango, Colorado, near the old Durango-Silverton railroad, I happened on *The Geometry Of Behavior Part 2: Chaotic Behavior*, and couldn't resist a longer look. (Maria's Bookstore has a couch towards the back for perusing, or snoozing—I've seen both.) Thirty minutes later I decided I needed not just Part 2, but volume 1 in the series.

I called Aerial Press (the publisher) in oceanside Santa Cruz and discovered a small publishing operation devoted to dynamics. Dynamics is that intriguing science that "deals with the concepts of change, rate of change, rate of rate of change, and so on, as they occur in natural phenomena."

Aerial publishes 16mm films, VHS, U-Matic, and Beta videos, books, lectures, and software on dynamics, chaos, fractals, and visual mathematics.

Dynamics

The methods of dynamics are becoming increasingly important to scientists modelling physical, biological, and social systems. Dynamics has exploded into a great hope for unifying the sciences. If the beauty of fractals or the philosophical implications of chaos has tweaked your curiosity, and you want to get a quick background in dynamics, I enthusiastically recommend Aerial's Visual Mathematics Library.

The four-volume set, *Geometry Of Behavior*, by mathematician Ralph Abraham and illustrator Christopher Shaw, includes 850 informal illustrations which embed mathematical complexities in pleasing semi-cartoons and caricatures. The state space discussion, for example, begins with a cartoon of an alligator-faced waffle iron and a concise paragraph about modelling—

An organism—physical, biological, or social—is observed in different states. This observed system is the target of the model. Its states can't really be described by only a few observable parameters, but we pretend that they can. This is the first step in the process of "mathematical idealization" and leads to a geometric model for the set of all idealized states: the state space of the model. Different models may begin with different state spaces. The relationship between the actual state of the real organism and the points of the geometric model is a fiction maintained for the sake of discussion, theory, thought, and so on.

The books explain (or more precisely, show through illustrations) such terms as state spaces, attractors (strange and otherwise), manifolds, saddles, insets, tangles, and basins.

The more I peruse this series, the better I like it.

Aerial Press, Inc.
P.O. Box 1360
Santa Cruz, CA 95061
(408) 425-8619

Thinking About Objects

That goes double for Bruce Eckel's second book, *Using C++*. It's the best in-depth study of object-oriented programming I've seen. In fact, I'm not even using it to learn more about C++. I'm using it to better understand Turbo Pascal 5.5 (the object-oriented version), which I use to model dynamic systems (see "Tidbits," this issue).

Bruce has been "building systems out of objects" for some time and uses the book to get us thinking about objects—

You can think of an object as an entity with an internal state and external operations. The external operations in C++ are member functions. The functions that execute the messages in an object-oriented language are called methods, and messages are the actual function calls. The concept of state means an object remembers things about itself when you aren't using it. An ordinary C function (one without static variables) is stateless because it always starts at the same point whenever you call it. Since an object has a state, however, you can have a function that does something different each time you call it.

Then Bruce gives an example; in fact, in these 600 thoroughly-crammed pages, he gives many examples.

Bruce has succeeded on several levels, I believe. *Using* C++ works: as a programmer's introduction to object-oriented programming (by using C to teach object-oriented programming, he's ironically turned the tables on Pascal, since Pascal has always been noted as a teaching language); as a programmer's introduction to C++; and as a toolbox for developing applications. The "big examples" include windows, a micro-CAD, and matrix operations.

Thorny topics such as pointers and references, and overloading functions and operators get ample coverage (60 and 40 pages, respectively).

C and Pascal programmers who want to understand objects will find much to learn from Bruce. You real-world folk won't be disappointed either. He spends 26 pages developing a clock-based event controller, which you can use as a model for your own object-oriented control systems.

Eckel, Bruce, 1989: *Using C++*; Osbourne-McGraw Hill; Berkeley, CA.

♦ ♦ ♦

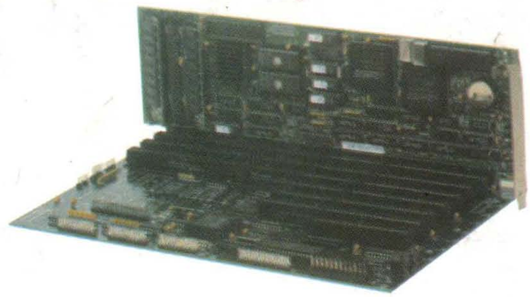
VERY HIGH PERFORMANCE

Processors, Memory, and Display Adapters

The X24 High performance processor

- 12 or 16 MHz 80286 with NO WAIT STATES!
- Small size ("XT" height and length) passive bus design
- 1 to 4 Mbyte 0 wait state dynamic memory
- Fully "AT" compatible Award BIOS
- Runs DOS versions 2.2 and later, Xenix and OS/2

The X24 combines the best of motherboard and backplane designs in a 100% AT compatible system. Incorporating a 16 MHz 80286, the X24 processor is designed to operate with the PC Tech Advanced System Motherboard, which contains the peripheral interfaces (hard disk, floppy disk, two serial ports and a parallel port). The X24 processor can also be used with other totally passive bus backplanes. Most critical components including the microprocessor and up to 4 megabytes of fast memory are contained on a single PC size plug-in card. This allows the processor and main system memory to be serviced or upgraded without disturbing other peripherals such as serial ports and disk drives.



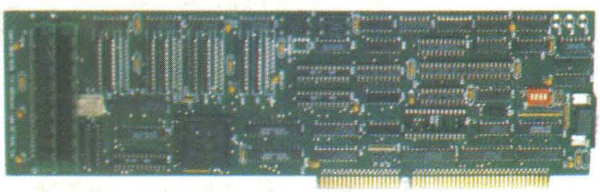
PC Tech X24 and ASMB

The PC Tech Advanced System Motherboard

- Built in "IDE" interface for AT interface type hard drives
- Fully AT compatible floppy disk support for 3.5", 5.25" drives, capacities of 360k, 1.2m and 1.44m
- Two serial ports and one parallel port
- 8 total expansion slots PC/XT/AT compatible (4 slots have 32 bit bus)

The PC Tech Advanced System Motherboard is designed to complement PC Tech's X24 and X32 high performance processor cards. It contains the mass storage interfaces necessary for a complete system, plus the basic I/O required in most systems. Extra care has been given to FCC compliance by design.

34010 Monochrome Graphics Adapter II



PC Tech Mono-II

- Up to 384k bytes display memory
- Up to 2 Megabytes program memory
- Software is RAM based, allowing complete operating software replacement and timing re-programming from the host bus
- 34010 program loader included. Assembler, debugger, and C compiler available.
- Full hardware and software CGA, MDA and Hercules emulation
- Single bit shared memory bit-map with optional resolution up to 2048 x 1536 (736 x 1008 standard)
- Very high resolution COLOR version available
- Custom 34010 software development available

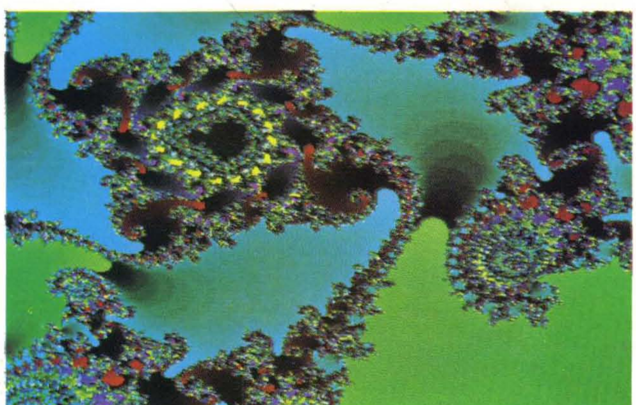
The TMS34010 is a true general purpose graphics processor. PC Tech makes the total processing power of the 34010 available to both programmers and end users. Our 34010 Monochrome Graphics Adapter is designed to allow programming from the PC/XT/AT host bus. You can completely replace our 34010 software with yours to directly harness the incredible image processing power of the TMS 34010 for your application. We make a complete set of development tools available, including an assembler, C compiler, program loader, 34010 debugger, and PC interface tracer/debugger. Our standard product includes support for extended CGA, MDA and Hercules emulation as well as a host addressable graphics bit-map. We also support and recommend the DGIS graphics interface standard (from Graphic Software Systems) for applications development as an alternative to native 34010 software development. Ready to run drivers are available for most major applications software packages as well.

Custom Designs Available

PC Tech will license most products for non-exclusive manufacture. We will also customize any of our designs to better meet your needs on our in-house CAD systems. All of our standard products are available in private label versions.

About PC Tech

PC Tech has been designing, manufacturing and marketing high performance PC related products for over three years. Our standard product line includes processor, memory, and video products. All products are designed, manufactured and supported in our Lake City, Minnesota facilities.



High resolution fractal produced on the PC Tech COLOR 34010

Designed, Sold and Serviced By:

PC Tech

907 N. 6th St., Lake City, MN 55041
(612) 345-4555 • (612) 345-5514 (FAX)

Reader Service Number 3