Tekcont and *pscont* —- programs to display and print contour maps

Rob MacLeod, CVRTI, University of Utah, macleod@cvrti.utah.edu

November 29, 2000

Contents

1 Background

Displaying contour maps can be one of the most cumbersome and time-consuming parts of processing map data, both because of the relative complexity of the problem and due to the lack of commercial software with adequate power and flexibility. Add to this the constantly changing requirements of scientists for new and different options and output modes, and any solutions become, by definition, customizations requiring extensive support and continuous modifications. This document describes yet another attempt at solving this problem, *pscont*.

The program was first written, and called *Tekcont* in 1989 at Dalhousie University, Dept. of Physiology & Biophysics, to run on a Vax under VMS. The goal was to devise a program that could display maps on the screen of a Tektronix terminal (or personal computer running some sort of terminal emulation *e.g.,*Kermit or NCSA Telnet) connected to one of several Vaxes. The native Tektronix plot software library available at the time was `Plot10` and hence, this paradigm formed the initial model for the layout and structure of the graphics. The first expansion of this environment came with the addition of an HP Colorpro plotter, connected directly to the PC/XT, and supported by the EM 4014 (and later EM 4105) terminal emulator from Diversified Computer System (DCS). This allowed not only direct hardcopy printing from images captured on the display, but features like zooming and management of coloured pens (and colour display, had this been available to me at the time).

With the addition of Postscript output capabilities in the form of a QMS 810 laser printer, it was not long before I wanted to generate laser-printer images from map data. This led to the development of a library of graphics subroutines which matched a subset of `Plot10` and allowed me to convert existing two-dimensional graphics code from `Plot10` to Postscript with relative ease. This library, `psteksubs`, contained routines to open and manage a file in which the Postscript code was written, and had user-callable routines with similar names to those in `Plot10`, simply prefixed with "PS", *eg.,* `Draw` became `PSDraw`, `Move` became `PSMove`, and so on. The scaling and layout paradigm continued to be that of `Plot10`, a screen of 100 by approximately 130 units in size, onto which multiple windows could be defined and individually manipulated.

Tekcont than received some more flexible means of allowing the user to design a layout for maps, test the result on the video display, and then run the same layout through the Postscript subroutines and generate quality hardcopy. In order to preserve some of the layout in a form that would allow later reproduction of the graphics, I devised the `tekcont_` file format. These files contain the filenames and record numbers of the files from which map data is extracted to generate the plot, together with some text fields which can be used to place labels on the plot. The user selects the other details of plot layout at execution time. An added benefit of having the file driven by these simple ASCII files is that the user can easily edit them, or even have them generated by programs or scripts.

The future for *Tekcont* is rather uncertain, especially given the move to Unix computers and the GL graphics library. `Plot10` is as good as dead and there seems little point in expanding the `psteksubs` library in the direction of `Plot10`. On the other hand, GL development is occurring almost exclusively in the C programming language while *Tekcont* is written in Fortran.

Recent Developments: While the screen viewing supported by the original *Tekcont* program has decreased in importance, the need for Postscript output has grown if anything. With the extensive use of the Macintosh-based facilities at the CVRTI, a new need as arisen – a way to get plot data into a form that can be manipulated by the broad range of powerful programs for editing images. The path we are now pursuing is based on the PICT standard, an alternative with numerous disadvantages over Postscript, but the single overwhelming advantage that it can be read and editing by other programs, something that is not yet generally available with Postscript.

As a result, we are developing new libraries of routines to produce graphics from without our own programs in PICT format. This work has not progressed to the point of inclusion into *pscont*, but this is the next project on the waiting list and should be in some useful form by autumn or winter of 1994.

The name of the program has also been changed in the meantime from *Tekcont* to *pscont*, to reflect the fact that Tektronix support has been dropped. The functionality built into SGI GL-based applications is so much better that there seems little point in supporting Tektronix terminal output. The Plot10 form of the routines has, however, remained and there are no plans to change that in the near future.

Latest addition (November, 1994) is to keep track of the bounding box in the psteksubs library and thus produce a more complete postscript file. This shold permit easier inclusion of the file into other postscript files and into LaTeX or other printing programs.

2 New Features

Saving Contours: We have recently added the capability to save the line segments that the contour subroutines generate into a separate file. This has actually always occurred, but in the past, this was a temporary file that was deleted when the program ran to completion. With the new feature, the user can supply a filename to keep a file for each of the plots on a single pages. To read the contents of this file, we have written some Matlab code and interfaced this to some Matlab plotting routines in order to then apply some smoothing functions to the contours. See 6.1 for the details of the fine structure.

RMS scalar: The user can set a scalar plot in the upper right corner of each page. This scalar can be a selected lead or an RMS curve made up of all selected leads used in the map.

3 Using *pscont*

pscont is designed to be interactive in its operation, allowing the user to enter parameters, select data files and layout, and then generate a plot on a single page of Postscript output, in the form of a file.

To see how the program works, it is probably simplest to walk through a sample session and describe the question and answer dialog as it unfolds. The program currently (November 29, 2000) runs only on Unix computers, with ports to SGI/IRIX, IBM/AIX, and SUN/OS in existence.

But first, an environment variable that is important. The value of ROBDIR must be set to my home directory in order to find some files that *pscont* needs to run properly. If ROBDIR is not set the program will yell when it fires up, offering a chance to enter a value interactively.

Now, on to the action...

```
eli:macleod/maprodxn/p3200/dat77> pscont

 Welcome to the program to display potential data
 in Postscript

 Assume the Rob MacLeod home directory here is /vis/u/macleod/
```

This is just a reporting message of the home directory used for a lot of the filename building. If ROBDIR is not defined, it must be entered here.

```
 Want to save the contours from each maps
 in a separate file for later use    (y/return) ?
```

Here is where the user can opt to save the contour segments in a set of files, one file for each map. If the answer here is "y", then the program asks later for the filename.

Then we also offer the chance to save more details that appear on the screen in a log file. This is used mainly for debugging.

```
Want to save info in a log file  (y/return) ?
```

Now we go on to getting the data

```
What type of input data is to be used ?
CVRTI .data files ........ "d"
Pot files ................ "p"
Your choice [def=p]  ?
```

There are two types of data files which I use to store potentials. These are the .pot and .data files. See section 5.2 for details on data file format. Suffice it to say that .data files are the most efficient, being binary, while the .pot files are the easiest to use and to transport, being simple (and I do mean simple) ASCII files. The next question is to make sure that we point to the right directory for the individual potentials files. The .data file will often contain only pointers to other files, which contain the actual potential values.

```
Pak/Raw file path read from MAP3D_DATAPATH as /vax/data/mapping/
Should we use this value  (return/n) ?
```

Now to the geometry:

```
Dal torso data .................  1
Dal epicardium .................  2
CVRTI 192-lead torso............  3
CVRTI 64-leads sock ............  5
CVRTI 128-leads sock ...........  6
Andy3 tank .....................  7
Andy3 tank leadset 1 (192).......  8
Stus 88 lead olive .............  9
16x16 plaque array ............. 10
Quans slice 1................... 11
Quans slice 2................... 12
Your choice [Def=  1]  ?
```

This is obviously a growth industry since there are a lot more geometries defined already than those given here. New ones will be added as I get time and need to plot different data.

```
The geometry files have a base filename of
    /u/macleod//contour/pscont/geom//daltorso2d
```

Just some more reporting...

```
To enter the file numbers by hand ........... "h"
Read the file numbers in from a file ......return  ?

Enter filename with extension - or without it .tekcont or .pscont used
Enter the name of the tekcont file to work with
[Default: oc1-1.tekcont ]  ? epi-inverse-1.tekcont
```

This dialog is important. The .tekcont file format allows the files for a single page worth of data to be listed, along with some textual information. See the sample .tekcont file in section 5.1 for examples and descriptions. Entry of the filename can be with or without the extension as the program is smart enough to look for both. In fact, *pscont* forces use of the extensions .tekcont or .pscont (don't gripe, it's a programmer right to make some demands!). Note also the mention of .pscont files here – they will probably be the new format I am already cooking up to manage things better. To maintain compatibility, use of the .tekcont extension for current files is recommended

```
In GetFileListFrom File an end of file was
hit after 18 entries were read in from epi-inverse-1.tekcont

 How do you want to select channels files?',
 One at a time for each data file ........ o
 Or a single file for the whole set ...... s
 Or no channels mapping at all ........... n
 [def = s ] ?

 Enter the channels file to work with
 [Default: /home/macleod/torso/dal/daltorso2d.channels ] ?
```

Channels files are used to link the data values and their locations in the geometry. See section 5.3 or the documentation for *map3d* for more details of channels files and the indirection they facilitate. In practice, the defaults set by the program are usually more than adequate. There is seldom a need for anything other than a single channels file for all the plots on a page and the default filename is stored in the program and offered for confirmation.

```
 The list of files read in is as follows:
#    Filename   Series   Frame   Middle  Left  Right
   1  p2_3200_qrst_77_epi_140.pot    1     1  Epi-Inverse  140  Instant
   2  p2_3200_qrst_77_epi_142.pot    1     1  Epi-Inverse  142  Instant
   3  p2_3200_qrst_77_epi_144.pot    1     1  Epi-Inverse  144  Instant
   4  p2_3200_qrst_77_epi_146.pot    1     1  Epi-Inverse  146  Instant
   5  p2_3200_qrst_77_epi_148.pot    1     1  Epi-Inverse  148  Instant
   6  p2_3200_qrst_77_epi_150.pot    1     1  Epi-Inverse  150  Instant
   7  p2_3200_qrst_77_epi_152.pot    1     1  Epi-Inverse  152  Instant
   8  p2_3200_qrst_77_epi_154.pot    1     1  Epi-Inverse  154  Instant
   9  p2_3200_qrst_77_epi_156.pot    1     1  Epi-Inverse  156  Instant
  10  p2_3200_qrst_77_epi_158.pot    1     1  Epi-Inverse  158  Instant
  11  p2_3200_qrst_77_epi_160.pot    1     1  Epi-Inverse  160  Instant
  12  p2_3200_qrst_77_epi_162.pot    1     1  Epi-Inverse  162  Instant
  13  p2_3200_qrst_77_epi_164.pot    1     1  Epi-Inverse  164  Instant
  14  p2_3200_qrst_77_epi_166.pot    1     1  Epi-Inverse  166  Instant
  15  p2_3200_qrst_77_epi_168.pot    1     1  Epi-Inverse  168  Instant
  16  p2_3200_qrst_77_epi_170.pot    1     1  Epi-Inverse  170  Instant
  17  p2_3200_qrst_77_epi_172.pot    1     1  Epi-Inverse  172  Instant
  18  p2_3200_qrst_77_epi_174.pot    1     1  Epi-Inverse  174  Instant

 Now enter the full directory path
 for the input files (if needed)
 Enter the name of the directory file to work with
 [Default: /vis/u/macleod/maprodxn/p3200/dat77/epi-inverse/ ]
   ?
```

The filenames can be listed in the .tekcont file without directory names, in order to make the whole thing more portable. But we obviously need to know where to look for the data files.

Now we begin a section that asks the user for parameters that determine the look and layout of the plot.

```
 To plot each map with the same contours ........ "s"
 Or to auto-contour each individually ........... "a"
 Your choice [def= a]  ?
```

This feature is now fixed and determines how the plots will be scaled. With autoscaling, each individual map is scaled on its own. This ensures that the maximum number of contour lines appear in each plot, but makes comparisons between plots more difficult. To nail the scaling down to a single set of values for all maps on the page, the user can choose "s" here. The program then scans all the data files and comes up with global extrema on which to base the contour level selection.

```
In the contour level setting routine (GetContVals):
For linear contour levels ...................."l"
or for logarithmic ..........................."g"
lab standard 7 log contours ............... return  ?
OK, lab-log 7 mapping selected
```

For linear scaling, further questions help set the step between contours and the way contour reflect around zero. Nothing really magic here, but some experimenting is usually required to get the desired appearance. The model used for setting contour values is also chosen at plot time. The "lab standard" is what I use quite often on BSPM data (it is Milan Horacek's idea) and computes 7 contour levels which logarithmically span a decade. The contours are based on a fixed sequence of 1.0, 1.5, 2.2, 3.3, 4.7, 6.8, and 10.0, which is scaled by factors of 10 until the mantissa of the largest extremum falls below 10. For example, if the largest extremum is -237.92, then the contours are fixed at ±222., ±150., 100, ±68, ±47, ±33, and ±22.

```
Want to have a scalar lead drawn (y/n)?
[def= n ] ?
```

This is a new feature that plots a scalar lead in the upper right corner of the page. It is very useful when plotting a selection of instant plots, or even a set of DWT maps to show where in the beat they came from. The catch (ah, there's always a catch) is that *pscont* needs the whole beat of data to extract the lead from. This can be tricky with the DWT data, but trivial when plotting instant maps from a .data file. I may have to come up with a workaround for more flexibility (and data-manipulation possibilities). Note that we can plot either a selected lead, or an RMS curve that includes all the leads in the file that are included for the plot (*i.e.,* the channels file determines which channels we include).

```
To write a single heading on the page .......... " 1"
To write a heading for each column ............. " 2"
To write both main and column heading........... " 3"
Or include no headings at all .................. " 0"
 Your choice [def =   0]   ?
```

Each page can have a main header, and each column a subheader. This s nice for producing plots to show others, and provides self-documentation for archiving the plots.

```
To plot without filename ..................... "-1"
To plot with lowercase filename .............. " 1"
To plot with uppercase filename .............. " 2"
To plot with filename as is .................. " 0"
 Your choice [def =   0]   ?
```

This feature is self-explanatory, isn't it?

```
Now for marking the plots
For no marking at all ........................ "0"
To draw all node points ...................... "1"
To draw in coronary arteries ................. "4"
To draw coronaries and node points ............. "5"
 Your choice [def=  0]   ?
```

The above feature is working again now. Use it with abandon, well almost. It will prompt differently based on the geometry (torso, epi, *etc,*). I have set the defaults so that standard precordial leads are plotted on the torso. For coronaries on the epicardium, *pscont* asks for guidance in finding the required .lmark file.

Note that the questions regarding max and min and coronary geometry type have been removed. They have become redundant so forget 'em. In the next section too, I have removed all reference to tektronix terminals since *pscont* is really just for postscript output.

```
Postscript options are:
            portrait..............."p"
            landscape.............."l"
return to take p  ?
```

What follows are several small decisions regarding layout of the plots. The defaults offered below are the "hard-wired" versions; if the user changes one of these, it becomes the new default value for the next plot layout, provided the user stays in the program.

```
There are 18 maps to display here
Enter the number of plots in the x and y-directions
or return to take the default
     X-direction: max = 4 ; def = 3 )  ?
     Y-direction: max = 6 ; def = 6 )  ?
Enter the desired line width in points
 [def =   0.75]  ?
```

The program currently allows a maximum of 24 maps in a four-row by six-column arrangement. In our experience, more maps than this on a page is seldom useful.

```
Re-do the last entries .............. "r"
Continue with plot ................ return  ?
Enter the name of the Postscript file to work with
[Default: epi-inverse-1.ps ]  ?
```

Last chance to correct things.

```
The file epi-inverse-1.ps exists
To enter new value ....................... "n"
To exit here and leave things untouched .... "e"
or to clobber the old version ........... . return  ?
File: epi-inverse-1.ps deleted
```

Now with everything laid out, *pscont* is ready to go ahead and produce postscript files. If the specified filename already exists, the user decides whether to overwrite it, enter the name anew, or even to exit the program. When running *pscont* via a shell script, this dialog can be troublesome since it typically messes up the list of responses that the script expects. The only solution I know of is to make very sure that the output files do not already exist before running the script.

As *pscont* does its thing, it outputs a progress report as it goes.

```
(readptsfile.f)
Read    97 points from /vis/u/macleod/torso/dal/dalepi2d.pts

(readfacfile.f)
```

```
 Finished reading   176 triangles from
/vis/u/macleod/torso/dal/dalepi2d.fac

 (readedgefile.for)
 Finished reading edge file /vis/u/macleod/torso/dal/dalepi2d.edge
 # Triangle vertices:    176 # Edges:    272
 Doing Postscript plot for map # 1....
 Doing Postscript plot for map # 2....
 Doing Postscript plot for map # 3....
 Doing Postscript plot for map # 4....
 Doing Postscript plot for map # 5....
 Doing Postscript plot for map # 6....
 Doing Postscript plot for map # 7....
 Doing Postscript plot for map # 8....
 Doing Postscript plot for map # 9....
 Doing Postscript plot for map #10....
 Doing Postscript plot for map #11....
 Doing Postscript plot for map #12....
 Doing Postscript plot for map #13....
 Doing Postscript plot for map #14....
 Doing Postscript plot for map #15....
 Doing Postscript plot for map #16....
 Doing Postscript plot for map #17....
 Doing Postscript plot for map #18....

 Output to the postscript file is complete
 Filename: epi-inverse-1.ps
```

Hooray, success!! The "s" option will send the Postscript file to the printer, if the version of *pscont* has been properly tuned to the local system. This option is still in its infancy, but works best if the environment variable PRINTER is set to a postscript printer of choice, and the system supports lpr-based (*versus* lp-based) printing. The other options offer paths to various jump-in points for getting new data or repeating the plot with a new set of control parameters.

```
 To send the file to the printer ................ "s"
 To display a new file .......................... "n"
 Use the same data but new contours ............. "c"
 Same contours but new parameters ............... "p"
 Or to end here .............................. return
 Your entry ?
```

4 Contouring

The contouring section of *pscont* is derived from some stand-alone programs written to generate "contour files" from potential data and geometry. The methods are based on work by Barr *et al.* [2, 3, 4] and require not only the geometry on which the data is to be projected, but also an "edge file", which must be generated for the particular geometry. The advantage of the edge-table based approach is that contour lines can be generated as continuous lines, a significant factor when mechanical plotters are used to produce hardcopy but not really an issue for video screen or Postscript printers. Even for Postscript plot, however, the procedure is useful since it produces contiguous contour lines that can easily be smoothed or manipulated before display. It is also simple to make an edge table from a .pts and .fac file set.

5 File types

5.1 Tekcont files

The filetype that is unique to *pscont* is the .tekcont file. Here is an example using a series of .pot files as the data source:

```
p2_3200_qrst_77_epi_140.pot 1 Epi-Inverse$140$Instant
p2_3200_qrst_77_epi_142.pot 1 Epi-Inverse$142$Instant
p2_3200_qrst_77_epi_144.pot 1 Epi-Inverse$144$Instant
p2_3200_qrst_77_epi_146.pot 1 Epi-Inverse$146$Instant
p2_3200_qrst_77_epi_148.pot 1 Epi-Inverse$148$Instant
p2_3200_qrst_77_epi_150.pot 1 Epi-Inverse$150$Instant
p2_3200_qrst_77_epi_152.pot 1 Epi-Inverse$152$Instant
p2_3200_qrst_77_epi_154.pot 1 Epi-Inverse$154$Instant
p2_3200_qrst_77_epi_156.pot 1 Epi-Inverse$156$Instant
p2_3200_qrst_77_epi_158.pot 1 Epi-Inverse$158$Instant
p2_3200_qrst_77_epi_160.pot 1 Epi-Inverse$160$Instant
p2_3200_qrst_77_epi_162.pot 1 Epi-Inverse$162$Instant
p2_3200_qrst_77_epi_164.pot 1 Epi-Inverse$164$Instant
p2_3200_qrst_77_epi_166.pot 1 Epi-Inverse$166$Instant
p2_3200_qrst_77_epi_168.pot 1 Epi-Inverse$168$Instant
p2_3200_qrst_77_epi_170.pot 1 Epi-Inverse$170$Instant
p2_3200_qrst_77_epi_172.pot 1 Epi-Inverse$172$Instant
p2_3200_qrst_77_epi_174.pot 1 Epi-Inverse$174$Instant
```

Here is another example using a pair of .data files as the source.

```
data/merged_370_658.data@1 24  From 370$At. Dr. 1$24 ms
data/merged_192_658.data@1 24  From 192$At. Dr. 1$24 ms
data/merged_117_658.data@1 24  From 117$At. Dr. 1$24 ms
data/merged_370_658.data@2 77  From 370$Pace 1$77 ms
data/merged_192_658.data@2 77  From 192$Pace 1$77 ms
data/merged_117_658.data@2 77  From 117$Pace 1$77 ms
data/merged_370_658.data@3 36  From 370$Pace 2$77 ms
data/merged_192_658.data@3 36  From 192$Pace 2$36 ms
data/merged_117_658.data@3 36  From 117$Pace 2$36 ms
data/merged_370_658.data@4 48  From 370$Pace 3$48 ms
data/merged_192_658.data@4 48  From 192$Pace 3$48 ms
data/merged_117_658.data@4 48  From 117$Pace 3$48 ms
data/merged_370_658.data@5 72  From 370$Pace 4$48 ms
data/merged_192_658.data@5 72  From 192$Pace 4$72 ms
data/merged_117_658.data@5 72  From 117$Pace 4$72 ms
data/merged_370_658.data@6 28  From 370$At. Dr. 2$28 ms
data/merged_192_658.data@6 28  From 192$At. Dr. 2$28 ms
data/merged_117_658.data@6 28  From 117$At. Dr. 2$28 ms
```

Each line of the file describes a single map, in the first case each map comes from a single .pot file. In the second case, the maps are all found in one of a pair of data files, each map representing a single frame in a time series within the file.

The fields in each line of a .tekcont files are defined as follows:

Tekcont file field formats	
Field #	Contents
1	Filename of the data file
2	Record number in the data file
3	Text to be placed in the centre cutout of the plot
4	Text for the left cutout of the plot
5	Text for the right cutout

Note that spaces separate the first 2 entries from the rest, while the dollar sign "$" separates the three text fields. If we have .data files as sources, the suffix "@" must be used to denote which time series is desired. The record number then becomes the frame number in the selected series.

The final locations of the "cutouts" depends on the type of geometry being used to plot the data. Fitting these cutouts into the layout of the geometry is why it takes some time to add a new geometry to *pscont*.

5.2 Potential data files

5.2.1 .pot files

The simplest way to store scalar data values (typically electric potentials) is in a series of .pot files. The values are generally ordered in the same way as the node points to which they are to be associated in the display, although channels files offer an opportunity for indirection of the link between data values and geometry nodes (see section 5.3).

The rules for `.pot` files are:

1. Each line of the file contains one data value, assumed to be a real or floating point number.
2. The order of the values must either agree with that of the associated set of geometry points or a channels file must be supplied to redirect the links between scalar value and nodes.
3. The file *must* end with a blank line.
4. A single file can contain only the data values associated with a single surface at a single instant in time.
5. The extension `.pot` *must* be used.

A set of files that contains a time sequence of data should have filenames of the form base_filename_001 to base_filename_NNN, with the numeric suffix (NNN) being a three-digit value, which can run in any regular increments. For example, pot_file_001.pot, pot_file_005.pot, pot_file_009.pot, pot_file_013.pot, ...

5.2.2 CVRTI data (.data) format files

The newly developed standard data files (.data extension) are capable of holding not only the scalar and vector data found in the older .pot and .grad files, but also links into existing "pak-tape" and "raw-tape" files under VMS, the name of the suggested geometry files, and various other bits of information that could be useful to a plot of data processing program. There is also a program called *pottodata* for converting a set of .pot files into one or more .data files.

There are some concepts of the data file structure that should be understood to appreciate how a program accesses the contents.

Time series or runs By a time series or run, we mean the basic structural entity of a .data file: Each .data file begins with a main header, and then continues with one of more time series of data, each with its own local header. Data within a time series belong together, typically because they have been recorded continuously at the same time. A time series of data is often multiplexed, *i.e.,* it has numerous leads of data, potentially over numerous surfaces. To select a particular run or time-series, its index number must be provided to the program. For *pscont* this information is wither stored in the .tekcont file or entered interactively.

Links to geometry The links between individual channels of data in the .data file and the nodes of the geometry to which the data is associated are established via *channel* array information. The channel array is either stored in the .data files as a set of associated scalars to the nodes of the geometry file (see documentation on Geometry files elsewhere), or stored in a dedicated .channels file. Section 5.3 contains more information on channels files.

Frames By a *frame* of data, we mean the set of values associated with a single instant in time. For each frame, there is a map. Frame numbers must also be specified, either in the .tekcont file or as part of the dialog between user and the *pscont*.

The .data files are binary and hence cannot be accessed without using a dedicated programs, which use elements of a special library written by Phil Ershler and Rob MacLeod. The suite of programs for reading, writing, and manipulating .data files is growing rapidly, and we can only refer the user to the likely authors of such programs for more information.

5.3 Leadlinks and Channels

5.3.1 Description of leadlinks and channels information

Leadlinks and channels files, and the arrays they contain, are identical in structure, but not in function. A program may require both, either, or neither of these, depending on the structure of the data files and geometries. The basic function of both leadlinks and channels information is to offer linkages between recording locations and the data that is to be associated with those locations. One , leadlinks, describes the connection between "leads", a measurement concept, with "nodes", and geometrical concept. The channels information, on the other hand, links the nodes to "channels" of data in a data file. Or, in more detail,

leadlinks The *leadlinks* information is primarily used to identify, and give numbers to, measurement leads within a set of nodes that make up the geometry. This can mean selecting a subset of the nodes, as would be used, for example, to identify the actual recording sites from a set node locations over which the data were interpolated. Leadlinks could also be used to renumber all the nodes of the geometry to reproduce the experimental setup.

In the leadlinks array, each entry refers, by its location in the array, to a particular lead; the array value at that location gives the number of the node in the geometry file to be associated with this lead. For example if lead 4 has a leadlinks entry of 22, then node 22 in the geometry will be displayed with a "4", not "22", when lead markings are selected in *map3d*. In *pscont*, there is no use for leadlinks arrays at this time.

leadlinknums The most recent change in the leadlinks structure has been the addition of the leadlinknums array, which has the same structure as the leadlinks array, but contains the actual lead number associated with each lead. This arrangement became necessary when there were cases of jumps in the lead numbers, for example when a lead is corrupted or damaged in recording, it is not available later, and should not be included in the list of leads of a display. Since the leadlinks array works purely by location in the array, we needed another level of indirection.

Hence we have the situation of a lead number K actually being called lead number L, pointing to node number M in the geometry. The *map3d* program now handles this additional indirection, and other programs are bound to follow suit as the need arises.

channels The *channels* information is used to relate nodes in the geometry to locations in the data file(s). For example, if we wish to find the data associated with node K in the geometry, then the value in location K of the channels array (`channels(k)`) will point to the correct channel in the data file.

Note that *channels arrays are used at the time the data is loaded into the internal data storage arrays!* The most frequent use of channels information is to unpack multiplexed data. This data come from input files with an inherently different structure in terms of geometry nodes and need to be sorted so that their spatial arrangements are known. The channels array provides the information needed to perform the sorting. An example would be data collected from multiple needles. The data are stored in a block, with no preconception of what spatial relationship that individual electrodes on those needles might have to one another. These relationships can only be fixed by the geometry information of where each electrode was located and what surfaces these locations are grouped into. To untangle this mess, a separate *channels* array is used for each surface, to point to the measured data values which belong to that surface, and to determine which data value is associated with each node location.

If data and geometry nodes match one-to-one, there is no need for a channels array. In *pscont*, a channels array is almost always a necessity because of the mapping from three to two dimensions. There are single points in the three-dimensional version of the geometry that have two equivalent points in the two-dimensional version of the geometry used for map display. Hence, *pscont* stores those channel filenames internally and associates then to a particular choice of plot geometry. In *map3d*, on the other hand, there are many cases in which channels information is not needed.

The figure 1 shows an example of lead and channels information layered one on top of the other. See the figure caption for details.

5.3.2 Source of leadlink and channel information

The source of both channels, leadlinks, and leadlinknums information can be either the geometry (`.geom`) file or the data (`.data`) file, or two explicit files, called "leadlinks" and "channels" files, which are described in the next sections below. In practice, for *pscont* this information almost always comes from explicit .channels files.

In .geom files Information for the *channels* array is stored as an associated scalar with the data information in the standard .geom files. At present, there is no leadlinks array stored in the .geom file but this could change at an time.

The .leadlinks file We include this for completeness sake since, as mentioned above, *pscont* does not use the leadlink information at this time.

A *leadlinks* file is an ASCII file, the first record of which contain a line `nnn leads`, where `nnn` is the number of leads to be described in the file (and also one less than the total number of lines in the file). Each following record contains two integer values:

1. the first number is the number of the lead, as it should appear in any labeling of the lead, the leadlinknum information.
2. the second entry in each row is the leadlink information for that lead.

For example, the file for a surface which reads:

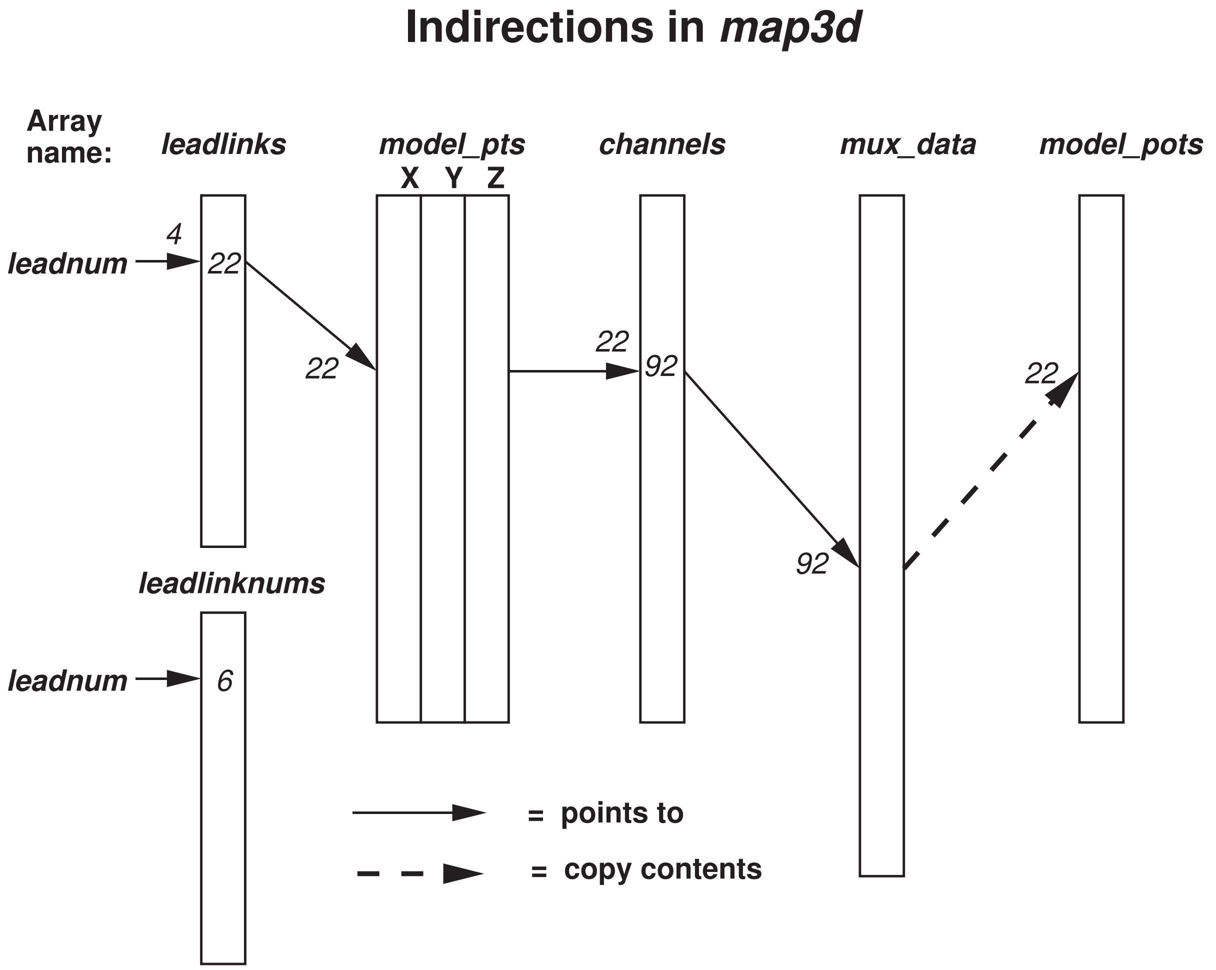

Figure 1: Example of the indirection possible in *map3d* through the use of *leadlinks* and *channel* information. Lead number 4 points, via the *leadlinks* array to node number 22. This, in turn, points via the *channels* array to location 92 in the multiplexed data buffer, which causes the value at location 92 to be loaded into location 22 in the *model_pots* array.
In a separate, leadlinknums array, shown below the leadlinks array, the entry in lead 4 says that that lead should actually be called lead 6, and so any labeling of the leads should reflect this additional indirection. In *pscont*, the indirection via *leadlinks* array is meaningless, but the *channels* array is crucial in data files that do not contain exactly the same number and order of data values as there are nodes in the geometry. This is almost never the case, however, since the data are stored based on the 3-dimensional locations, but pscont displays in 2D, so some data values gets used twice, once on each side of the display. Normally, the user may accepts the defaults offered by *pscont* since they are tuned for the particular display geometry.

```
32 Leads
1   1
2   42
4   31
7   65
.   .
.   .
.   .
32  11     <---- 32nd entry in the file, at line 33 of the file.
```

indicates that there are 32 leads to be linked, and that lead 1 is called lead 1 and is node 1 in the geometry file. Lead two is at node 42, but lead 3 is called "4" and is found at node 31. Likewise, lead 4 is called 7, and is located at node 65, and so on, up to lead 32, called 32, at node 11.

The `.channels` file A .channels file is an ASCII file, the first record of which contain a first line **`nnn nodes`**, where **`nnn`** is the number of nodes to be described in the file (and also one less than the total number of lines in the file). Each following record contains two integer values:

1. the first number is simply a running counter that indicates which node number the second value in the row is
2. The second element in each row is the *channel* number for that node.

For example, the file for a surface which reads:

```
352 Nodes
1    123
2    632
.    .
.    .
32   12
```

indicates that there are 352 leads to be linked, and that the data value for the first node is located at location 123 of the data file. For node 2, the data value is to be found at location 632, and so on.

5.4 Coronary artery (landmark) files

The landmark files contain what I refer to as "landmarks" on the surface(s) of the geometry. Initial use was primarily for coronary arteries, but the idea has been expanded to incorporate a number of different orientation landmarks; the only requirement is that these landmarks can be described as a series of connected points, with a radius defined for each point. The landmark is then displayed as a faceted "pipe" linearly connecting the points at the centre, with a radius, also linearly interpolated between points, determining the size of the pipe. The landmark file can contain numerous, individual segments of such pipe-work, each of which is drawn separately.

A single point can also be a landmark and in this case, all that is required is a point in 3D space and a radius. The *map3d* icon for this sort of a landmark is a sphere of fixed radius and different colours, depending on the type of landmark.

In the 2D case of *pscont* the drawing of the landmarks is much simpler, and used so far exclusively for the epicardial distribution of coronary arteries.

The format of the landmark file is as follows:

Line number	Contents	Comments
1	nsegs	number of landmark segments in the file
2	1 landmark_type nsegpts	segment number (1), segment type, and number of points in segment 1
3	X, Y, Z, radius	point location and radius of point 1
4	X, Y, Z, radius	point location and radius of point 2
.	.	.
.	.	.
nsegpts + 2	X, Y, Z, radius	point location and radius of last point in segment 1
.	2 landmark_type nsegpts	segment number (2), segment type, and number of points in segment 2
.	X, Y, Z, radius	point location and radius of point 1
.	X, Y, Z, radius	point location and radius of point 2
.	.	.
.	.	.
.	X, Y, Z, radius	point location and radius of last point in segment 2
.	.	.
.	.	.
.	.	.
.	.	.

In the two-dimensional case, the Z-values (that's "zed" values, mind you) are set equal to zero.

The standard extension of the file is `.lmark` and the filename is usually made up either based on a patient number for customized arteries, or in a generic coronary circulation geometry file.

6 Programming details

6.1 Binary contour files

Each contour segment is stored in a binary file from within the contouring libary. The format of this file is as follows:

```
WRITE(luout) contnum, ncpts, vcont
WRITE(luout) ((cpoint(i,j), i=1,2), j=1,ncpts)
```

To read the result back into Matlab, we have created the following function:

```
function  [contours, numcontours]=read_contourfile(filename)
% Read in a matrix in Rob's Fortran contour format
%
% The contours variable returned is an array of contourinfo structures,
% each contaning all the points for a single contour line
%
% .numpts   = number of points in the contour line
% .contval  = scalar value of the contour line
% .contpts  = (2 X numpts) array of order X/Y points for the contour
%
% numcontours = the number of contours found in this file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[fid, message] = fopen([filename],'r');
if ( fid < 0 )
  fprintf('Illegal value returned by fopen = %d\n', fid);
  fprintf('Return message is %s\n', message)
  return
end
numcontours = 0;
keepreading = 1;
%  contours = struct('numpts', 0, 'contval', 0.0, '
while( keepreading > 0 )

  % Read in the first block of info, contnum, contval, and ncpts.

  [header1, numreads]=fread(fid,3,'int32');
  if  numreads < 1
    return;
  end
  contnum = header1(2);
  ncpts = header1(3);

  % Read in the contour value

  header2 = fread(fid,3,'float32');
  contval =  header2(1);

  % Read in the actual contour points
  [contourset, numreads] = fread(fid,[2,ncpts],'float32');

  % Read in a dummy record terminator.
  dummy = fread( fid, 1, 'int32');

  if ( numreads ~= ncpts*2 )
    fprintf(' Error reading contour file\n');
    fprintf(' numreads = %d but ncpts*2 = %d\n', numreads, ncpts*2);
    return;
  else
    numcontours = numcontours + 1;
    fprintf(' Contour #%d contains %d points and contour value %f\n',...
numcontours, ncpts, contval );
  end
  contours(numcontours).numpts = ncpts;
  contours(numcontours).contval = contval;
  contours(numcontours).contpts = contourset;
end

fprintf(' Read in %d contours\n', numcontours);
```

6.2 Make files

The following make file creates the executable under Unix on the SGI's at the CVRTI. On Suns, use the gcc compiler, but any ANSI compatible compiler should handle the C-libraries. The Fortran is pretty standard and seem to run on most systems, with some rather bizarre changes needed for the Suns. Seems they have a limit on the number of continuation lines a program is allowed to have. Don't ask me...

```
# pscont.make The makefile for pscont
# Author: Rob MacLeod
# Last update: Fri Oct 31 19:40:07 1997
#    - needed proper use of ${SYSEXT} macro!
# Last update: Fri Oct  4 11:34:57 1996
#    - added TARGET
# Last update: Wed May 18 16:09:59 MDT 1994
#    - made more portable with update facilities too
# Last update: Mon Sep 28 12:35:33 MDT 1992
#    -


SHELL = /bin/sh
RM = /bin/rm -f
REMDIR = vissgi.cvrti.utah.edu:dfile/show
REMLIBDIR = vissgi.cvrti.utah.edu:dfile/lib
SYSTEM =
SYSEXT = -${SYSTEM}
SYSEXT =
TARGET = pscont
################## Defines ####################################

OBJECTS = ${TARGET}.o \
dopscont.o \
drawarrow.o drawlandmark.o drawscalar.o \
enterfilelist.o maxmindata.o \
getplotparam.o getfilelistfromfile.o getcorfilename.o \
getscalar.o getmap.o


LIBS = -B static -ldatafile -lfgraphicsio -ltorso -lpstek -lcontour\
-lutil    -B dynamic


LOCALHEADERS = pscont.fh

HEADERS = pscont.fh \
dfileio.fh \
        fgraphicsio.fh

##################### Dependencies ###########################
all: ${TARGET}${SYSEXT}

${TARGET}: ${OBJECTS}
$(F77) -o $@${SYSEXT} ${OBJECTS}  -L${HOME}/lib $(LIBS)

#f.o.:
# $(F77) -c $(FFLAGS) *.f
```

```
${OBJECTS}: ${HEADERS}


############## Machine Specific settings #####################
# SGI
F77 = f77
FFLAGS = -g -I${HOME}/include#-n32
RCP = /usr/bsd/rcp
LIBDIR = ${ROBHOME}/dfile/lib
CP = /sbin/cp

# Suns
#F77 = f77
#FFLAGS = -g -w
#RCP = /usr/ucb/rcp
#LIBDIR = ${ROBHOME}/dfile/lib
#CP = /sbin/cp

# IBM
#F77 = xlf
#FFLAGS = -g
#RCP = /usr/bsd/rcp
#LIBDIR = ${ROBHOME}/dfile/lib
#CP = /sbin/cp

########################### Utilities ############################

install: ${TARGET}${SYSEXT}
${CP} ${TARGET}${SYSEXT} ${ROBHOME}/bin

touch:
touch  ${OBJECTS:.o=.f}

clean mostlyclean:
-$(RM) *.o
-$(RM) *~
-$(RM) \#*
-$(RM) a.out
-$(RM) core

clobber distclean realclean: clean
-$(RM) ${TARGET}${SYSEXT}

update:
for filename in ${OBJECTS:.o=.f}; do \
echo "Copying $${filename} from ${REMDIR}" ; \
${RCP} macleod@${REMDIR}/$${filename} . ; \
done
for filename in ${HEADERS}; do \
echo "Copying $${filename} from ${REMLIBDIR} to ${LIBDIR}" ; \
```

```
${RCP} macleod@${REMLIBDIR}/$${filename} ${LIBDIR} ; \
done

# Print the sources

print: $(OBJECTS:.o=.f)
    lpr $<

maketar:
echo "Tarring files into  ${TARGET}-source.tar "
tar cvf ${TARGET}-source.tar ${OBJECTS:.o=.f} ${HEADERS}  Makefile
```

For details on all the different libraries, consult the Library binder in my office or the Makefiles in the individual subdirectories in the LIBS macro.

References

[1] R.S. MacLeod. *Percutaneous Transluminal Coronary Angioplasty as a Model of Cardiac Ischemia: Clinical and Modelling Studies.* PhD thesis, Dalhousie University, Halifax, N.S. Canada, 1990.

[2] R.C. Barr, T.M. Gallie, and M.S. Spach. Automated production of contour maps for electrophysiology I. Problem definition, solution strategy, and specification of geometric model. *Comp & Biom Res*, 13:142–153, 1980.

[3] R.C. Barr, T.M. Gallie, and M.S. Spach. Automated production of contour maps for electrophysiology II. Triangularization, verification, and organization of the geometric model. *Comp & Biom Res*, 13:154–170, 1980.

[4] R.C. Barr, T.M. Gallie, and M.S. Spach. Automated production of contour maps for electrophysiology III. Construction of contour maps. *Comp & Biom Res*, 13:171–191, 1980.