An Introduction to the Piecewise Parabolic Method (PPM) Library
II. I/O and Data Analysis

David H. Porter,
Sarah E. Anderson,
Paul R. Woodward,
and B Edgar,

Laboratory of Computational Science or Engineering,
Department of Astronomy, School of Physics and Astronomy
116 Church St. SE.
University of Minnesota
Minneapolis, Minnesota 55455

5. A3d

a. Introduction

A3d is a powerful analysis tool which works on the compressed data dumps in the format supported by the PPMLIB I/O package. It is designed for calculations in both two- and three-dimensions on uniform grids. (If the computational grid is non-uniform, the compressed data dump could be preprocessed and interpolation used to create an equivalent compressed data dump for a uniform grid.) It embodies scalable algorithms for creating images from the data controlled by a text string defining the operations to be performed on the data or actions to be taken. This permits efficient batch processing of the hundreds of compressed dump files typical to hydrodynamic simulations while retaining the ability of individual processing of files.

With a compressed data file at hand, **a3d** may be used to calculate any combination of algebraic, differential, or integral forms on the data. The fields which are produced may be stored in the form of histograms, power-spectra, one-dimensional cuts through the data (suitable for graphical display), or 8-bit binary images. In two-dimensions the 8-bit binary images are suitable for viewing as is by **xraz**, a disk based raster viewer available from the devil Himself, or the user may convert these files to a format of his choosing. In three dimensions, this binary output comprises a ‘‘brick of bytes,’’ with one byte per voxel, stored as one continuous array, without any file header information. Each voxel corresponds to a computational zone. The data are stored so that the ‘‘X’’ axis index is the fastest varying and the ‘‘Z’’ axis index the least fastest. Thus the data consists of Z planes, with each plane consisting of Y rows of X bytes. These brick of bytes may be view by **bob**, a tool for browsing 3D data. This program is also available from the Devil for the nominal charge of one soul.

a. Commands

The command structure of **a3d** uses Reverse Polish Notation, a natural mode for the numerate and one familiar to users of classic Hewlitt-Packard pocket calculators. In this notation, the operands are entered into a stack (levels a, b, c, d, etc.). Mathematical operations are performed on the contents of these stacks, generally the uppermost stack, or pair of stacks.

This program is invoked by the command line

a3d *datafile [commandsfile*

where *datafile* is the name of the binary compressed data file to be analyzed and *commandsfile* is a file containing the commands which specify the quantities to be examined and the operations to be performed. If *commandsfile* is not provided, then the prgram searched for a file named ‘‘param’’ in the directory from which **a3d** was invoked. If such a file is not found, the program terminates. The this file contains a list of commands which are in the form of ASCII text strings of 20 characters, or fewer, delineated by spaces or carriage returns. Each command is an action which operates on one or more of the quantities in the stack. The names of any output files are generated by replacing the fourth character of *datafile* with a user specified string. If no such string is specified, the fourth character is incremented by one. If the resulting name corresponds to an existing file, that file will be over written. Information about the grid stored in the data file, such as mesh dimensions and coordinates, are automatically used by **a3d** as required.

The following tables list the most frequently used commands available in **a3d** and illustrates the richness of the **a3d** vocabulary if primitive its syntax may be. In the first table are commands which set global parameters or modes. The next table lists commands related to data output. All output take information from the top level of the stack. The final table table lists the mathematical operations available to the user.

Global Parameters	
Command	Action
@ *ccc*	set substring of output file name to *ccc*
show	display a summary of the contents of a data file
debug	display each command that a3d performs, for **a3d** wizards
:	delineate map from floating point physical values to 8-bit image values
set size_of_bin *a*	set width of histogram bins to *a*
set zero_of_bin *b*	values btw -5000 *a* and -5000 *a* will be histogramed

Output commands	
Command	Action
bob	write out data as an 8-bit per pixel image file truncated to 0-255
histogram	generate histogram of data and write it to a disk file
#	write columns of numbers to an ascii text file along a 1-D cut
dft1	calculate & output a 1-D discrete Fourier transform of vertically averaged data
vfield	write out streak lines of a vector field
s	toggle into and out of an interactive display mode within **a3d**

The next two tables list the operations which may be performed on the stack. Note that in the this table the string ‘‘...’’ refers to that part of the stack which is unaffected by the listed operation, and ";" is used to delimit the stack elements which are affected by the operation. The top of the stack is the first element listed. The strings given for the coordinates (X, Y, and Z), their derivatives (x, y, and z), and the field variables (QQQ) are only intended as examples. These symbols are arbitrary, chosen by the user, and determined by the contents of the data file, with the restrictions that the first character of the name of a field variable or constant is upper case (between A and Z), and that coordinate names are always one character long and upper case. Derivatives are always specified by the lower case character of the letter corresponding to the desired coordinate direction. The string *nn.nnn* refers to a decimal constant which may be pushed onto the stack. The only restriction on *nn.nnn* is that it be positive and the first character must be a digit between 0 and 9 inclusive. (Values less than one *must* be entered as 0.*nnn*. Negative values may be entered using the ‘‘change sign’’ operator, *c* listed below.) The fourth column, Δ, indicates the change in the number of stack elements induced by the listed command.

Input and Stack Operations				
String	Stack Before	Stack After	Δ	Comment
$nn.nnn$	...	$nn.nnn$; ...	+1	Push constant $nn.nnn$ onto stack
X	...	X; ...	+1	Push coordinate X onto stack
Y	...	Y; ...	+1	Push coordinate Y onto stack
Z	...	Z; ...	+1	Push coordinate Z onto stack
QQQ	...	QQQ; ...	+1	Push variable QQQ onto stack
+	a; b; ...	$b + a$; ...	-1	addition
-	a; b; ...	$b - a$; ...	-1	subtraction
*	a; b; ...	$b * a$; ...	-1	multiplication
/	a; b; ...	b/a; ...	-1	division
l	a; ...	$\log_e a$; ...	0	natural log
e	a; ...	e^a; ...	0	exponentiation
a	a; ...	$\lvert a\rvert$; ...	0	absolute value
c	a; ...	$-a$; ...	0	change sign
d	a; ...	a; a; ...	+1	duplication
n	a; ...	$128(1 + a/(1 + \lvert a\rvert))$; ...	0	nonlinear mapping to [0, 255]
%	a; b; ...	b; a; ...	0	interchange top two elements of stack
,	a; ...	...	-1	pop top of stack
h	$a[m][n]$; ...	$\sum_{k=0}^{n} a[m][k]$; ...	0	cumulative sum in horizontal
v	$a[m][n]$; ...	$\sum_{k=0}^{m} a[k][n]$; ...	0	cumulative sum in vertical
t	n; $a[i][j]$; ...	$a[i][j+n]$; ...	-1	translate in horizontal direction
x	a; ...	da/dx; ...	0	derivative in x direction
y	a; ...	da/dy; ...	0	derivative in y direction
z	a; ...	da/dz; ...	0	derivative in z direction

The last four operations (t, x, y and z) listed in the table above require information about boundary conditions. Two kinds of boundary conditions are currently supported for all three derivative operations. The boundary conditions for each direction may be independently changed by using the "set" command. For example, the string *set bndry1 4* in a commands file will set the first (or x in the examples above) direction to be periodic. Similarly *set˜bndry2˜4* will set the second direction to be periodic. The default boundary condition is linear extrapolation of the two zones next to each boundary, and corresponds to the value 0 (*set˜bndry1˜0* for example). Horizontally periodic boundary conditions are always assumed by the translation operator t.

There are also a variety of commands for manipulating three dimensional vectors on the stack. These commands allow the user to calculate dot products, divergence, curl, and norms of vector fields both easily and efficiently. These commands are listed in the following table, in which the notation $[\vec{A}]$ indicates the three stack elements A_z; A_y; A_x in that order.

3-D Vector Operations*				
String	Stack Before	Stack After	Δ	Description
div	$[\vec{A}]$; ...	$\vec{\nabla} \cdot \vec{A}$; $[\vec{A}]$; ...	+1	Divergence
curl	$[\vec{A}]$; ...	$[\vec{B}]$; $[\vec{A}]$; ...	+3	Curl: $\vec{B} = \vec{\nabla} \times \vec{A}$
norm2	$[\vec{A}]$; ...	A^2; $[\vec{A}]$; ...	+1	Vector magnitude squared
norm1a	$[\vec{A}]$; ...	$\lvert A\rvert$; ...	-2	Vector magnitude squared
dot	$[\vec{A}]$; $[\vec{B}]$; ...	$\vec{A} \cdot \vec{B}$; $[\vec{A}]$; $[\vec{B}]$; ...	+1	Dot product
normdot	$[\vec{A}]$; $[\vec{B}]$; ...	$\vec{A} \cdot \vec{B}/(\lvert A\rvert\,\lvert B\rvert)$; $[\vec{A}]$; $[\vec{B}]$; ...	+1	Cosine of angle between $\vec{A}$ and $\vec{B}$
grad	ϕ; ...	$[\vec{A}]$; ϕ; ...	+3	Gradient of scalar field: $\vec{A} = \vec{\nabla}\phi$
cross	$[\vec{A}]$; $[\vec{B}]$; ...	$[\vec{C}]$; ...	-3	Cross product: $\vec{C} = \vec{B} \times \vec{A}$

* These command are restricted to three dimensions only.

As mentioned before, a significant difference between the output behaviour for 2-D and 3-D data sets is that **a3d** produces a three-dimensional brick of data for its 8-bit raster output. This brick may be viewed using the 3D data browser **Bob**.

To provide a concrete example, run the two examples from the previous section, which will produce two compressed dump files “q42a0000” from the two dimensional example and “p14a0000” from the three dimensional example. **A3d** uses the user assigned strings (key words) provided in the compressed dump file to access the data. The command “show” is useful in this regard to summarize the contents of a compressed dump file and display the key words and the descriptions as supplied by the user when creating said file. The commands are passed to a3d as ASCII text in a parameter file. Now create a file named `param` which consists of the line `show`. The command

a3d q42a0000

will produce the following output:

```
name = q42a0000, nzzons = 59, nyzons = 1, nrzons = 61
Coordinate X   Range = [0.000000, 1.000000]   Mesh =  59   Derivative x
Coordinate A   Range = [0.000000, 1.000000]   Mesh =   1   Derivative a
Coordinate Y   Range = [0.000000, 1.000000]   Mesh =  61   Derivative y

     Symbol           Value    Name

      GAMMA    1.400000e+00    ratio of specific heats

3-D: Symbol                    Name         Minimum         Maximum

        RHO                 Density    3.000000e+00    8.000000e+00
          P                Pressure    3.000000e+00    8.000000e+00
         VX              X velocity   -9.996457e-01    9.996456e-01
         VY              Y velocity   -9.996685e-01    9.996685e-01
       Diag    2D chess board diagnostic    0.000000e+00    1.000000e+00

  Comment 0:
  This is the result from the program cdumps2dexample

  Comment 1:
  Its purpose is to demonstrate the PPMLIB compressed

  Comment 2:
  dumps routines. The arrays are REAL*4

  Comment 3:
File created 13Apr1999 at 13:47:24 Local Time ( -300 minutes from UT)
nroff = -2    nrs = 20   nrzons = 61
```

(The date and time that the file was created will, of course, differ.)

The first line gives the file name and the data dimensions. (Recall that because fBa3d works on either two or three dimensional data, two dimensional data treated as three-dimensional data, with one of the dimensions having a value of one. This gratuitous dimension is always the second dimension.) Next comes the ranges and axis labels. Here, the two axes are “X” and “Y”. (The “A” axis can be ignored.) Derivatives with respect to these axes would be computed using the operations “x” or “y,” resp. Next come the scaler value of the polytropic index which have been assign the keyword “GAMMA,”. This is followed by the arrays for the density and pressure (keywords “LNRHO” and “LNP”) and the two velocity components (keywords “VX” and “VY”). The diagnostic is references with the keyword “Diag”. The extrema values of the above arrays are also provided. Any comments included in the file are displayed last.

A param file containing the following

```
@ RHO
RHO 5 * bob
@ P
P 5 * bob
@ VX
VX 1. + 127 * bob
@ VY
VY 1. + 127 * bob
@ Diag
Diag 1. + 127 * bob
```

would create five files, “q42RHO0000”, “q42P0000”, “q42VX0000’, “q42VY0000” and “q42DIAG0000” containing binary images of scales values of density, pressure, x-velocity, y-velocity, and the diagnostic. When viewed by **xraz** they appear as in figure 1.

Similarly, when used on the three dimensional compressed dump file ‘‘p14a0000’’ created by the second example from the last section, a param file containing

```
@ RHO
RHO 5 * bob
@ P
P 5 * bob
@ VX
VX 1. + 127 * bob
@ VY
VY 1. + 127 * bob
@ VZ
VZ 1. + 127 * bob
@ Diag
Diag 1. + 127 * bob
```

would create five files, ‘‘p14RHO0000’’, ‘‘p14P0000’’, ‘‘p14VX0000’, ‘‘p14VY0000’’ and ‘‘p14DIAG0000’’ when **a3d** which consist of ‘‘brick of bytes’’ vo density, pressure, the three components of velocity, and the diagnostic, resp. When viewed by **bob** these data appear as in figure 2. z-veloc

```
@   vort
VY   x   VX   y   -
:   n   :   bob
```

When applied to the compressed data file A00a0050 used in the last example, **a3d** will create the file *A00vort0050* containing an 8-bit raster image of the vorticity. The commands file could equally well have been written on a single line, (i.e., the carriage returns could be replaced by spaces), but the commands have been grouped on three lines to emphasize three separate steps. The "@" symbol on the first line is used to control the name of the output file to be created: the string following the "@" will replace the fourth character in the input file’s name. On the second line, *Vy* reads the y-velocity field information from the compressed data file and places it on the stack. Since *X* (upper-case) was specified as a coordinate, the string *x* (lower-case) takes the x-derivative of whichever field is at the top of the stack. *Vx* places the x-component of velocity onto the stack, and *y* then takes the y-derivative. The "-" command subtracts the top of the stack from the next stack level down and pops the stack. The operator *n* maps the floating point number in each mesh zone of the top stack level to a value between 0 and 255, which is suitable for an 8-bit raster image, via the mapping: $x \rightarrow 128(1 + x/(1 + |x|))$. The two colons ":" on the third line are used to delineate the map from vorticity in physical units to pixel values. The colons have no effect unless **a3d** is run with the header option at the end of the command line, as mentioned above. Finally, "*bob*" writes the top stack level (an array of floating point numbers) to the 8-bit raster file (one byte per array element). *image* first truncates each floating point number to an integer value, then clips the number to be in the range [2, 254] before writing out the final 8-bit value. The color map indexes 0, 1, and 255 are used by many windowing systems for special purposes.

Another example constructs a histogram of the entropy, $ln(P/\rho^{\gamma})$, of the fluid simulation described above. A suitable commands file might read

```
@   s
LNP LNRHO   GAMMA   *   - e
histogram
```

where *GAMMA* refers to the adiabatic index of the gas in the data file. In this example, the operator "*" multiplies the top two levels of the stack, ‘‘-’’ subtracts the top two stacks, and ‘‘e’’ exponentiates the values in the top stack. Finally, *histogram* makes **a3d** write out three columns of numbers into the ascii text file *A00s0053hist*. The first column is the centroid for each bin of the histogram, the second column is the number of zones with values in each bin, and the third column is the cumulative distribution. An ASCII text file composed of columns of numbers can be plotted as a histogram with any of a variety of interactive graphics tools. One such widely available tool is "mongo".