

Data handling in visualization

How not to get stuck in IO-related problems ?

Stefan Mayer,
Informatics and Mathematical Modelling,
Technical University of Denmark

December 12, 2001

Motivation

Input of data is often the critical task in visualization!

- Who might not be interested in this talk:
 - Those, who only use one software environment and do not need to know about the technical stuff
 - Those, who know all this already

Problem/Challenge in this talk

There is no systematic know-how anywhere!
(Not entirely true, of course)

Data handling is

- first of all a craft, not science
- experience-based
- usually need some programming skills

My background

My main problems in visualization

- Large amount of data stored and processed
- Geometry handling
- Temporal data, sequentially written and read

A simple ASCII data file

```
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
1.000000e+00 1.999867e-02 3.998933e-02 7.991470e-02
2.000000e+00 3.998933e-02 7.991470e-02 1.593182e-01
3.000000e+00 5.996400e-02 1.197122e-01 2.377026e-01
4.000000e+00 7.991470e-02 1.593182e-01 3.145666e-01
5.000000e+00 9.983341e-02 1.986693e-01 3.894183e-01
6.000000e+00 1.197122e-01 2.377026e-01 4.617792e-01
7.000000e+00 1.395431e-01 2.763557e-01 5.311862e-01
8.000000e+00 1.593182e-01 3.145666e-01 5.971954e-01
9.000000e+00 1.790296e-01 3.522742e-01 6.593847e-01
1.000000e+01 1.986693e-01 3.894183e-01 7.173561e-01
1.100000e+01 2.182296e-01 4.259395e-01 7.707389e-01
1.200000e+01 2.377026e-01 4.617792e-01 8.191916e-01
1.300000e+01 2.570806e-01 4.968801e-01 8.624042e-01
1.400000e+01 2.763557e-01 5.311862e-01 9.001005e-01
1.500000e+01 2.955202e-01 5.646425e-01 9.320391e-01
1.600000e+01 3.145666e-01 5.971954e-01 9.580159e-01
.....
```

Simple code example

Write:

```
file=fopen("test.dat","w");
for (i=0;i<m;i++)
{
    fprintf(file,"%12.6e %12.6e %12.6e %12.6e \n",
            x[i][0],x[i][1],x[i][2],x[i][3]);
}
fclose(file);
```

Read:

```
for (i=0;i<m;i++)
{
    flag=fscanf(file,"%e %e %e %e ",
                &(x[i][0]),&(x[i][1]),&(x[i][2]),&(x[i][3]));
}
fclose(file);
```

Problems:

- Suboptimal precision (often)
- Limited line length (system dependent)
- File size and processing speed
- Many editors do not read large files/long lines anyway
- possible UNIX - DOS transfer problems

Simple binary read/write code example

Read:

```
file=fopen("test.dat","w");  
fwrite(x,sizeof(REAL),n*m,file);  
fclose(file);
```

Read:

```
file=fopen("test.dat","r");  
fread(x,sizeof(REAL),n*m,file);  
fclose(file);
```

Size and Speed

Example on 800Mhz Athlon PC: ATA-100 DISK

“read/write 500.000 × 4 double precision floats”

	write time (sec)	read time (sec)	file size (MB)
ASCII %12.6e	5.7	4.4	27.2
ACCII %12.6e, gzipped	17.5	5.3	8.8
ACCII %20.16e	8.0	5.4	47.2
ASCII %20.16e, gzipped	22.6	7.0	17.1
binary	0.1	0.1	16.0
binary gzipped	5.8	0.8	13.0
binary, conv. to floats	0.2	0.1	8.0
binary, conv. fl., gzipped	2.9	0.5	6.6

Single value access

“take out 100 single values at random positions”

	random access time (sec)	file size (MB)
ACCII %20.16e	291.0	47.2
binary	< 0.1	16.0

Properties of raw binary data files

Advantages

- small size
- fast write, read and random access

Problems

- Not readable in editor (or ?)
- You need to know type, size, machine, etc. to retrieve

A simple useful tool

OD(1)

FSF

NAME

od - dump files in octal and other formats

SYNOPSIS

od [OPTION]... [FILE]...

od --traditional [FILE] [[+]OFFSET [[+]LABEL]]

DESCRIPTION

Write an unambiguous representation, octal bytes by default, of FILE to standard output. With no FILE, or when FILE is -, read standard input.

-A, --address-radix=RADIX

decide how file offsets are printed

-j, --skip-bytes=BYTES

skip BYTES input bytes first on each file

-t, --format=TYPE

select output format or formats

--MORE--

A simple useful tool

od output:

```
od -t fD test.dat | more
```

```
0000000  0.0000000000000000e+00  0.0000000000000000e+00
*
0000040  1.0000000000000000e+00  1.999866669333308e-02
0000060  3.998933418663416e-02  7.991469396917270e-02
0000100  2.0000000000000000e+00  3.998933418663416e-02
0000120  7.991469396917270e-02  1.593182066142460e-01
0000140  3.0000000000000000e+00  5.996400647944460e-02
0000160  1.197122072889194e-01  2.377026264271346e-01
0000200  4.0000000000000000e+00  7.991469396917270e-02
0000220  1.593182066142460e-01  3.145665606161178e-01
0000240  5.0000000000000000e+00  9.983341664682815e-02
0000260  1.986693307950612e-01  3.894183423086505e-01
0000300  6.0000000000000000e+00  1.197122072889194e-01
0000320  2.377026264271346e-01  4.617791755414829e-01
0000340  7.0000000000000000e+00  1.395431146442365e-01
0000360  2.763556485641138e-01  5.311861979208834e-01
0000400  8.0000000000000000e+00  1.593182066142460e-01
0000420  3.145665606161178e-01  5.971954413623921e-01
0000440  9.0000000000000000e+00  1.790295734258242e-01
--More--
```

Simple types and machine formats

Binary types in C

char	1 byte
unsigned char	1 byte
short	usually 2 byte
unsigned short	usually 2 byte
int	either 2 or 4 byte
unsigned int	either 2 or 4 byte
long	usually 4 byte
unsigned long	usually 4 byte
float	usually 4 byte
double	usually 8 byte
long double	either 8 or 16 byte

Native machine formats in "modern computers"

- Most significant bit first, "Big endian"
unsigned short:

0	0000000000000000
1	0000000000000001
2	0000000000000010
3	0000000000000011
..
65535	1111111111111111

- Least significant bit first, "Little endian"
unsigned short:

0	0000000000000000
1	1000000000000000
2	0100000000000000
3	1100000000000000
..
65535	1111111111111111

Simple types and machine formats

Big-Endian Machines:

Processor type	Operating system
IBM RS/6000	AIX
HP PaRisc	HP-UX
Motorola	Macintosh
"	Linux
SGI Rxxxx	IRIX
Sun Sparc	Solaris
"	Linux

Little-Endian Machines:

Processor type	Operating system
Intel X86	DOS/Windows
"	LINUX
"	Solaris 86
DEC Alpha	Digital Unix
"	Alpha VMS

Matlab "fopen" statement

FOPEN Open file.

FID = FOPEN(FILENAME) opens the file FILENAME for read access. FILENAME can be a MATLABPATH relative partial pathname. FID is a scalar MATLAB integer, called a file identifier.

FID = FOPEN(FILENAME,PERMISSION) opens the file FILENAME in the mode specified by PERMISSION.

FID = FOPEN(FILENAME,PERMISSION,MACHINEFORMAT) opens the specified file with the specified PERMISSION and treats data read using FREAD or data written using FWRITE as having a format given by MACHINEFORMAT. MACHINEFORMAT is one of the following strings:

'native'	or 'n'	- local machine format - the default
'ieee-le'	or 'l'	- IEEE floating point with little-endian byte ordering
'ieee-be'	or 'b'	- IEEE floating point with big-endian byte ordering
'vaxg'	or 'g'	- VAX G floating point and VAX ordering
'cray'	or 'c'	- Cray floating point with big-endian byte ordering
'ieee-le.164'	or 'a'	- IEEE floating point with little-endian byte ordering

and 64 bit long data type
'ieee-be.l64' or 's' - IEEE floating point with
big-endian byte ordering
and 64 bit long data type.

.....

Matlab "fread" statement

FREAD Read binary data from file.

`A = FREAD(FID,SIZE,PRECISION)` reads binary data from the specified file and writes it into matrix A. FID is an integer file identifier obtained from FOPEN. The SIZE argument is optional;

The PRECISION argument is a string that specifies the format of the data to be read. Any of the following strings, either the MATLAB version, or their C or Fortran equivalent, may be used.

MATLAB	C or Fortran	Description
'uchar'	'unsigned char'	unsigned character, 8 bits.
'schar'	'signed char'	signed character, 8 bits.
'int8'	'integer*1'	integer, 8 bits.
'int16'	'integer*2'	integer, 16 bits.
'int32'	'integer*4'	integer, 32 bits.
'int64'	'integer*8'	integer, 64 bits.
'uint8'	'integer*1'	unsigned integer, 8 bits.
'uint16'	'integer*2'	unsigned integer, 16 bits.
'uint32'	'integer*4'	unsigned integer, 32 bits.
'uint64'	'integer*8'	unsigned integer, 64 bits.
'single'	'real*4'	floating point, 32 bits.
'float32'	'real*4'	floating point, 32 bits.
'double'	'real*8'	floating point, 64 bits.
'float64'	'real*8'	floating point, 64 bits.
.....		

Matlab read of "test.dat"

```
function a=matlab_read(filen,dsizе,dtype,dformat)
    fid=fopen(filen'r',dformat);
    a=fread(fid,dsizе,dtype);
    fclose(fid);
```

```
>>
```

```
>> t=cputime;
    a=matlab_read('test.dat',[4 500000],'double','l');
    cputime-t
```

```
ans =
```

```
    0.1800
```

```
>> a(:,1:4)
```

```
ans =
```

```
Columns 1 through 7
```

0	1.0000	2.0000	3.0000	4.0000
0	0.0200	0.0400	0.0600	0.0799
0	0.0400	0.0799	0.1197	0.1593
0	0.0799	0.1593	0.2377	0.3146

```
>>
```

Matlab read of "test.dat"

```
function a=matlab_read(filename,dsize,dtype,dformat)
    fid=fopen(filename,'r',dformat);
    a=fread(fid,dsize,dtype);
    fclose(fid);
```

```
>>
```

```
>> t=cputime;
```

```
    a=matlab_read('test.dat',[4 500000],'double','b');
```

```
    cputime-t
```

```
ans =
```

```
    0.1800
```

```
>> a(:,1:4)
```

```
ans =
```

```
1.0e+286 *
```

```
Columns 1 through 7
```

0	0.0000	0.0000	0.0000	0.0000
0	0.0000	-4.2923	-0.0000	0.0000
0	-4.2923	0.0000	-0.0000	0.0000
0	0.0000	0.0000	-0.0000	0.0000

>>

Structure of "test.mat"

```
>> save test.mat a
```

```
>> ls -l test*
```

```
ans =
```

```
stm      stm      16000000 Nov 29 14:35 test.dat
```

```
stm      stm      16000184 Nov 29 14:42 test.mat
```

```
>>
```

Structure of "test.mat"

```
od -t c test.mat | more
```

```
0000000 M A T L A B 5 . 0 M A
0000020 i l e , P l a t f o r m
0000040 L N X 8 6 , C r e a t e
0000060 n : T h u N o v 2 9
0000100 : 4 2 : 1 7 2 0 0 1
0000120
*
0000160 \0
0000200 016 \0 \0 \0 0 $ \0 006 \0 \0 \0 \b
0000220 006 \0 \0 \0 \0 \0 \0 \0 005 \0 \0 \0 \b
0000240 004 \0 \0 \0 \a \0 001 \0 001 \0 $
```

```
od -j184 -t f8 test.mat | more
```

```
0000270 0.0000000000000000e+00 0.0000000000000000e+00
*
0000330 1.0000000000000000e+00 1.999866669333308e-02
0000350 3.998933418663416e-02 7.991469396917270e-02
0000370 2.0000000000000000e+00 3.998933418663416e-02
0000410 7.991469396917270e-02 1.593182066142460e-01
0000430 3.0000000000000000e+00 5.996400647944460e-02
0000450 1.197122072889194e-01 2.377026264271346e-01
0000470 4.0000000000000000e+00 7.991469396917270e-02
0000510 1.593182066142460e-01 3.145665606161178e-01
0000530 5.0000000000000000e+00 9.983341664682815e-02
```

System independent data formats

Self-explanatory files

- Descriptive header
- Data in various (binary) formats

Examples: Graphics files

Adobe Photoshop	–	Big Endian
BMP (Windows and OS/2)	–	Little Endian
DXF (AutoCad)	–	Variable
GIF	–	Little Endian
IMG (GEM Raster)	–	Big Endian
JPEG	–	Big Endian
FLI (Autodesk Animator)	–	Little Endian
MacPaint	–	Big Endian
PCX (PC Paintbrush)	–	Little Endian
PostScript	–	Not Applicable (text!)
Microsoft RIFF (.WAV & .AVI)	–	Both
Microsoft RTF (Rich Text Format)	–	Little Endian
Sun Raster	–	Big Endian
TGA (Targa)	–	Little Endian
TIFF	–	Both
XWD (X Window Dump)	–	Both

General purpose data files

- CDF: simple multi-D data
- FITS: astronomical (image) data
- GRIB: meteorological data format
- HDF: generalized multi-D data
- XDF: object oriented generalized multi-D data within XML
- netCDF: generalized multi-D data (geophysics)
- CXF: Chemical Exchange Format, chemical structural data
- MedFileS: Records of clinical data
- OpenMath: Mathematical formulae and equations
- DEM: cartographic elevation data
- SAIF: object oriented geographic data
- DLG-3: geographic data
- VICAR: Image data
- PDS: Planetary data
- CIF: Crystallography data

General purpose data files

Two parts:

- File structure and formats
- I/O library

Example: CDF

- CDFCreate(.....)
- CDFOpen(.....)
- CDFDOC(.....)
- CDFInquire(.....)
- CDFDelete(.....)
- CDFattrCreate(.....)
- CDFvarCreate(.....)
- CDFvarPut(.....)
- CDFvarGet(.....)
- CDFvarClose(.....)
- ...

Filters available

Matlab

- CDF: simple multi-D data
- HDF: generalized multi-D data
- netCDF: generalized multi-D data (geophysics)
- TIFF, BMP, a lot more image formats
-

IBM Data Explorer

- CDF: simple multi-D data
- HDF: generalized multi-D data
- netCDF: generalized multi-D data (geophysics)
- TIFF, BMP, a lot more image formats
-

Ggobi

- XML
- MySql
-