Point Data

Peter Teuben - University of Maryland
Draft Version 0.4 (20-jun-2002)

1 Summary

A general outline is given how Point Data used in astrophysical simulations can be stored persistently and efficiently for archival and data-exchange purposes. No implementation details are given here, although translation to currently popular data formats should be straightforward. For illustrative purposes some examples of current data storage habits are interspersed in the discussion. Also, the translation of how this format implements data structures in memory is left open to the programmer.

see also `http://www.astro.umd.edu/nemo/tvo` for future updates of this document

2 Introduction

The design of a general enough dataformat that can be used for all theory data and for the foreseeable future, is a daunting task. However, a cursory analysis of theory data seem to show that most, if not all, data can be divided in two classes[1]: *tabular* and *grid* data. Particle (or Point) Data fit in the class of tabular data, where each particle fills a row in the table, and the columns are their different attributes (mass, position, age, chemical composition, etc.)
In some way grid data can even be viewed as a special case of tabular data, where there is no distinction between rows and columns, although the concept of planes needs to be introduced to handle a 3rd dimension.
Also noteworthy to add to the discussion here is that by *tabular* data I mean to say `normalized tables`, i.e. no redundancies exist in the table.
Data of this type can be easily implemented in a Relational Database (RDBMS). A few years ago Object Oriented Databases (OODMS) were thought as a new and innovative way to store data, and better matched to Object Oriented programming, and our Point Data example naturally would fit in this scheme, but OODMS have arguably not made an impact in the industry.

[1]this appears to be true for theoretical as well as observational data

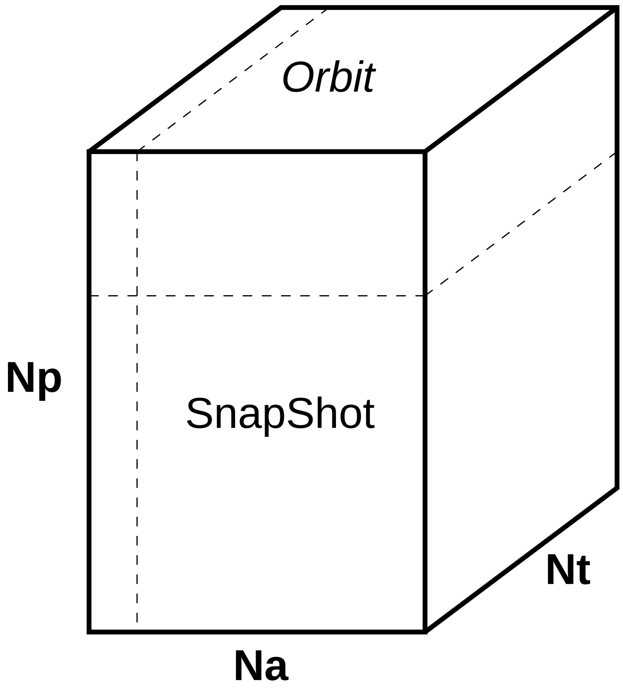

Figure 1: A Particle-Attribute-Time (PAT) Data Cube (`Np` number of particles, `Na` number of attributes, `Nt` number of time steps)

3 Particle Data

On the most simple and traditional level (ignoring "header" variables for the moment) the basic data of a particle simulation can be thought of a series of SnapShots, as visualized in Figure 1. Orbits are then located as horizontal slices through this cube. The notion of variable timesteps will be discussed later, but for now we assume a series of SnapShot's at some set of times.

4 Examples

4.1 Aarseth's NBODYx

Traditional languages like FORTRAN often "forces" programmers to store the data in a column-major way (i.e. not like a direct FORTRAN matrix dump). Here is an example from his latest NBODY1 version:

```
      OPEN (UNIT=3,STATUS='NEW',FORM='UNFORMATTED',FILE='OUT3')

      WRITE (3)  N, MODEL, NRUN, NK
      WRITE (3)  (A(K),K=1,NK), (BODY(J),J=1,N),
     &           ((XS(K,J),K=1,3),J=1,N), ((XDOT(K,J),K=1,3),J=1,N),
     &           (NAME(J),J=1,N)

      CLOSE (UNIT=3)
```

but note that although data has been written in the traditional non-portable UNFORMATTED way, a nice little header simplifies reading the variable parts of the data part. However, this data is by no means self-describing.

4.2 NEMO

In languages such as C/C++ (and also Fortran-90) the concept of structures was introduced and it became easy to keep all data associated with a particle near the same memory location. Although the data could also be directly written out as such:

```
    typedef struct {                          //  an example 'body' structure
      real   mass;
      vector phase[2];
      real   phi;
    } body;


    fp = fopen(fname,"w")                     // open the file
    fwrite(btab, sizeof(body), nbody, fp);    // write 'nbody' body's to the file
    fclose(fp);                               // close the file
```

the authors of NEMO decided against this, as can been seen in this human readable dump of the binary data in NEMO:

```
char Headline[35] "init_xrandom: seed used 1024449832"
char History[29] "mkplummer p10 10 VERSION=2.6"
set SnapShot
  set Parameters
    int Nobj 10
    double Time 0.00000
  tes
  set Particles
```

```
    int CoordSystem 66306
    double Mass[10] 0.100000 0.100000 0.100000 0.100000 0.100000
      0.100000 0.100000 0.100000 0.100000 0.100000
    double PhaseSpace[10][2][3] -0.197339 0.426812 -1.28257 0.199586
      -0.365966 0.350464 0.0339697 -0.409276 0.451770 0.655413
      0.460108 -0.233395 -0.321197 1.25690 -0.793112 0.199850
      -0.0891718 0.312586 0.0682481 -0.369760 0.870463 -0.0503744
      . . .
  tes
tes
```

The side-effect of storing particle information column-based on disk, but keeping them row-based in memory is a more complicated I/O layer.

4.3 FITS BINTABLE

The FITS BINTABLE format suits itself fairly easily to this format, with the caveat that FITS header data are stored in ASCII, in a somewhat contrived format limiting header variables to upper case, 8 characters, and leaving minimal space for comments (80 characters for the variable name, value and comment field). There is an old proposal by Teuben (1995) to store particle data in FITS BINTABLE format.

4.4 HDF

NCSA promotes the usage of their Hierarchical Data Format (Folk, 199x), but is currently going through a major backwards incompatible changeover from HDF4 to HDF5. HDF is a very rich dataformat, defined via an API. The Scientific Data Set (SDS) family is generally used for storing *grid* data. Our type of point data is *tabular*, and a HDF. A proposal by Greg Bryan and Frank Summers exists to store particle data in HDF format.

4.5 VOTable

The most recent kid on the block is XML (X Markup Language), which the IT industry is embracing. In the - for us - relevant Virtual Observatory this has already resulted in two proposals: the NVO is promoting the VOTable, whereas a few years ago NASA started promoting XDF (X Data Standard). Both are toy models and still have a limited implementations in code (VOTable in perl, XDF in java). We need to closely watch this community.

5 Data

Here is a attempt to meta-encode the data

```
Album     = set of SnapShot's, each at different times
SnapShot  = set of Family's, each of different composition with different sets of attributes
Family    = set of Particles, each with the same set of attributes
Particle  = a Point with numerous Attributes (or it could even be a Family or SnapShot?)
Attribute = a named Quantity (a number or vector) with a unit
```

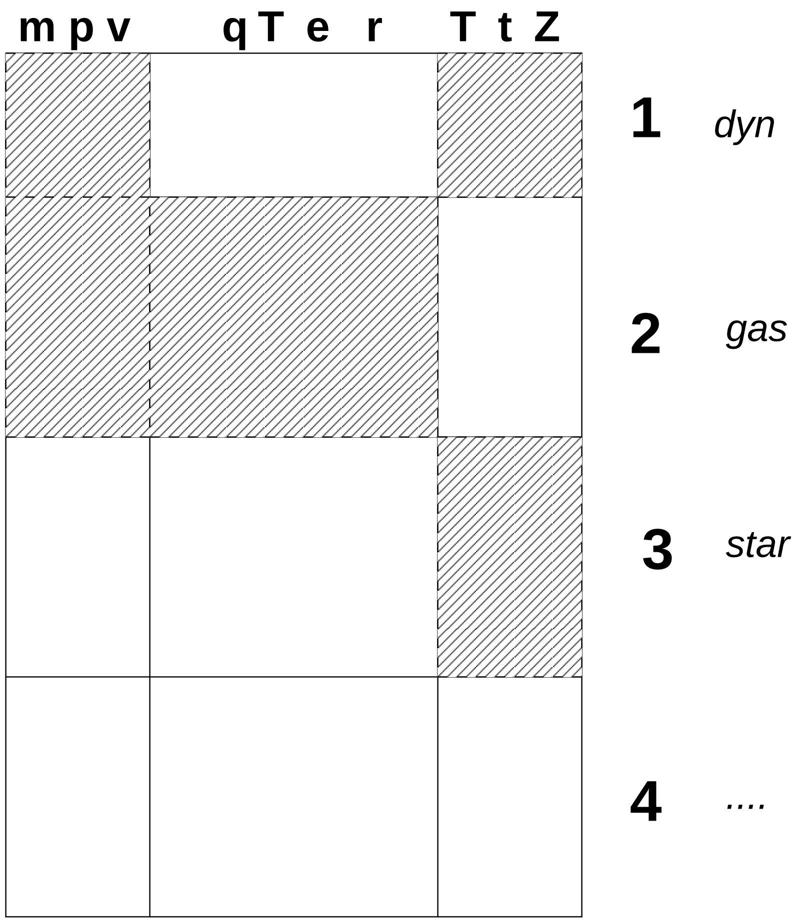

Figure 2: Different families of particles contain different sets of attributes

```
int  Npoint         // total number of particles
int  Nfamily        // number of families
real Time           // time of snapshot (unit string optional)
{
  int Np            // number of particles in this family
  string Name       // a nice descriptive name for this family
  string CoordSys   // Coordinate system (cartesian-xyz, polar-rpt, ...)
  vector Pos[]      // positions
  vector Vel[]      // velocities
  real  Mass[]      // masses
}
{
  int Np
  string Name       // a nice descriptive name for this family
  string CoordSys   // Coordinate system (cartesian-xyz, polar-rpt, ...)
  vector Pos[]      // positions
```

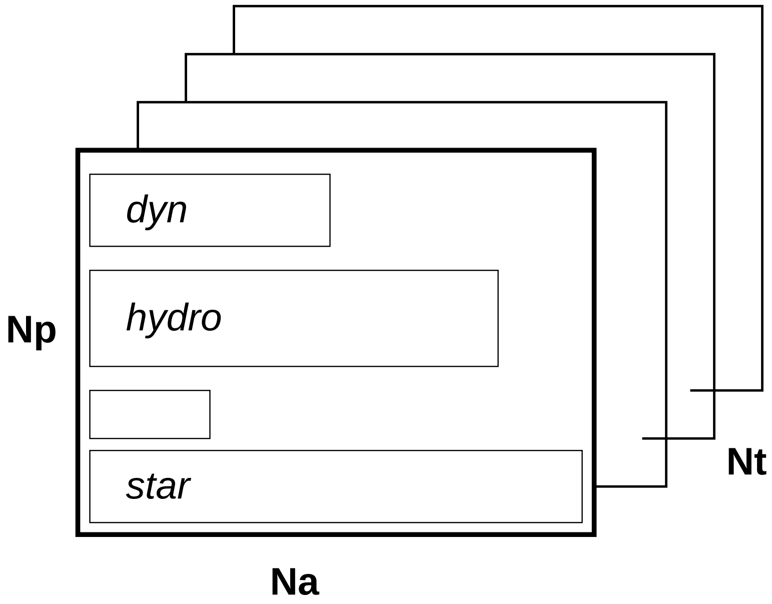

Figure 3: Particle ∈ Family ∈ SnapShot ∈ Album

```
    vector Vel[]       // velocities
    vector Mass[]      // masses
  }
  ...
```

5.1 Attribute (Column) Names

```
          SnapShot and Family header variables

  int    Npoint              //  9223372036854775808
  int    Ndim                //  3     (number of dimensions, mostly 2 or 3 i guess)
  int    Nattributes         //  3     (number of attributes, e.g. mass,pos,vel)
  string FamilyName          // 'disk', 'halo', 'gas', 'bulge'
  string CoordSys            // 'cartesian-xyz', 'polar-rtp', 'cartesian-zxy'
  string CodeName            // 'arik'
  string CodeAuthor          // 'Roald Teuben'
  string CodeVersion         // '3.0'
  string CodeDate            // '23-jun-2014'
  string Hardware            // 'grape12'

           Particle Attributes (-vector means Ndim applies)

  real-vector Pos[]          // simple dynamics
  real-vector Vel[]
  real-vector Pos[]
  real        Mass[]
  real-vector Acc[]
```

```
real        Potential[]
real        Density[]

real  SPHEntropy[]          // gas properties (SPH)
real  SPHTemperature[]      //
real  SPHSmoothingLength[]  // also needs to know the SPH kernel type
real  SPHDensity[]          // contrast also a Hut/Casertano like density
real  SPHAcc[]              // pressure gradient
int   SPHNneib[]            // number of nearest neighbors, if applicable

real  Age[]                 // stellar evolution
real  Temperature[]
real  Metallicity[]
real  Radius[]

real  SemiMajorAxis[]       // orbital elements
real  Eccentricity[]
real  Inclination[]
real  LongAscendNode[]
real  LongPeriapse[]
real  TrueLong[]
```

6 Unresolved issues

- Variable time-step integrators that dump particles "as they see fit" (cf. the `tdyn` format in Starlab). The dataformat could allow for a column labeled time, instead of it being a table header. It has proven to be a useful format for dynamical analysis such as visualization and real-time animation. This collapses the T axis into the P-A plane.
- Can the same particle attribute appears multiple times? For example, can mass be zero for gravity, but finite for stellar evolution? Or be quasi asynchronously updated, as was discussed earlier in this meeting.
- The "Greenbank Convention" allows for columns that do not vary to be promoted to become a header variable. `Time` is an obvious example. Makes for great data compression.

7 Future

Some of this design has been taken from an earlier design I sketched out for adding code to AIPS++ to help comparing (particle based) models with (grid/image based) observations. It is possible that you will find this implemented back in AIPS++ at some point in the not too far future.

Acknowledgments

Much of what has been written down here are the results of discussions I have had with Piet, Steve, Jun, Shawn, Dave DeYoung, Tom Quinn, Lars Hernquist, Josh Barnes, and many others over the past few years.

References/Background

- FITS BINTABLE, e.g. `http://fits.nrao.edu`.
- VOTable: `http://vizier.u-strasbg.fr/doc/VOTable/`
- AIPS++: `http://aips2.nrao.edu`
- Folk, M. & Zoellick, B. *File Structures*, Addison-Wesley 1992.
- NEMO Users and Programmers Guide `http://www.astro.umd.edu/nemo`.
- Teuben, P.J. *NEMO Stellar Dynamics Toolbox*, (1995, ADASS-IV proceedings)
- XDF: `http://xml.gsfc.nasa.gov/XDF/XDF_home.html`
- Bryan, G. & Summers, F. (1995) `http://zeus.ncsa.uiuc.edu:8080/data_format/data_format.html`