Kapteyn Laboratorium

der Rijksuniversiteit te Groningen
Landleven 12 - Postbus 800 - 9700 AV Groningen - Nederland
telefoon (050) 116695 - telex 53572 stars nl

THE NEW GIPSY DATA STRUCTURE FOR IMAGES

J.P. Terlouw

Summary:

The advent of more astronomical data coming from different instruments is gradually making the current GIPSY data structures obsolete.

I present here an approach to solving this problem by adopting a completely general new data structure.

The main purpose of this document is being a subject of discussion in the working group which is investigating the requirements of the new GIPSY. It has not the pretention of being a complete system design.

Groningen, March 1985.

1. The existing ("old") data structure

The existing data structure originates from 1975 and was designed by Ekers and Terlouw for the STER data reduction package for Westerbork data on the Cyber {1}. With some minor changes it has later been transferred to the PDP-11/70 to become a part of GIPSY {2,3}. Basically it consists of two components: a 3-dimensional structure ("set") for images ("subsets") and per image a header describing that image. Some elements of these headers are in fact descriptors for the whole 3-dimensional structure and are present in every image header thus introducing redundancy.
The structure of the headers is positional; every element in the header is meant to store one specific parameter. This approach necessitated extending the header with new elements required for other data than Westerbork, while other elements lost their meaning but still were physically present because of the positional structure of the headers. For a program it is often difficult to find out that a header element has lost its meaning.
The structure does not allow for other than 2- or 3-dimensional data, while not all dimensions are treated in a uniform way. The third dimension is treated differently from the first two dimensions.

2. The new data structure

In my opinion the most important shortcoming of the present data structure is the lack of generality. This has led to a number of unpleasant experiences when attempts were made to use GIPSY on other data than Westerbork for which the system was designed. Patches were necessary to be able to use the software for optical, IRAS and even VLA data. Some parts of GIPSY now deserve the name "patchwork".
These problems have convinced me of the necessity of a completely general new structure; not only an improved structure accomodating the astronomical parameters we now know of. Such a new structure again would consist of two parts: a data part and a header (or descriptor) part. The data part would be an N-dimensional array (as a file on disk), with no further structure than its N-dimensional shape. This immediately implies that all further structure must be defined in the header. This can be done by imposing a hierarchy on the header which is only determined by the N-dimensional structure, not by a physical interpretation of this structure. On the highest level of this hierarchy it would be at least necessary to provide information to enable the interpretation of the dimensions.
The generality would further be guaranteed by the requirement that all header elements are accessed by name instead of position. If certain parameters would not be present, this fact would easily become known by a program.

3. The header

The header normally consists of two types of data:
- constants which give information about the nature of the data described by the header;
- algorithms which can be used by programs processing the data.

The algorithms can be either explicit or implicit. The present data structure only knows implicit algorithms. An example is the transformation from coordinates in the file ("grid units") to R.A. and Dec. In this case the algorithm is the instrument used (e.g. "WSRT") and parameters to this

algorithm are the file coordinates together with the contents of a number of other header elements such as grid spacing.
It may be worthwhile to investigate the possibility of having explicit algorithms in the header. This probably would only work for relatively simple formula's. The coordinate transformation example probably would not be a good candidate for an explicit algorithm, but a transformation from a file position to a velocity or frequency might. A problem to be solved if explicit algorithms would be used is the fact that one often also needs the inverse algorithm. This could be done by specifying both algorithms.

An important requirement is that redundancy should be avoided as much as possible. In a hierarchical structure this means that information must be present at the highest possible level. The generality and flexibility requirements however dictate that exceptions should be possible. This can be accomplished with a header access procedure which first probes at the level at which a program asks the information and goes up in the hierarchy until the information is found or proved to be not present. (This going up could even extend to the user which then would act as the top-level header.)

The proposed hierarchical header should in principle allow for subheaders for each place in the hierarchy. Now the question rises: how many different kinds of headers are possible for an N-dimensional structure. If we limit ourselves to headers for 0...N-dimensional "orthogonal" substructures within the whole structure (pixels, lines, planes, boxes, etc.), there will be 2**N different kinds of headers. This follows from the fact that for all possible substructures each of the N dimensions can be either defined or undefined. Extreme cases are "pixel" headers (all dimensions defined) and the top-level header for the whole structure (all dimensions undefined).
A 3-dimensional "data cube" would have at most 8 different headers and a 4-dimensional structure 16. Though these numbers are higher than the ones we are used to, the total amount of space occupied by all headers for a given structure is not necessarily large due to the fact that all information is present at the highest possible level thus avoiding duplication.

Example:
Consider a Westerbork data cube with 3 dimensions: l, m, and v. The 3 dimensions lead to 2**3=8 different headers. For the following table we assume that "1" means that a coordinate has a specific value, while "0" means that a coordinate is undefined.

v	m	l	corresponding GIPSY structure	dimensionality
0	0	0	SET	3
1	0	0	map or SUBSET	2
0	1	0	l-v map	2
0	0	1	m-v map	2
1	1	0	row in map	1
1	0	1	column in map	1
0	1	1	velocity profile	1
1	1	1	pixel	0

4. Data access

Data access in the new structure would be different from access in the old structure, but it should still be possible to use old methods via special interface routines.
The new structure would make it possible to make programs more general. For instance the contour plot program CPLOT could be changed so that it could work on any plane in the data cube. By interrogating the header it could determine whether nice frames could be drawn and how they should be drawn, or if the data is not accompanied by sufficient information, decide not to draw a nice frame but only one with grid tick marks. Except for some special applications, this would make the position-velocity map generation program LV superfluous. LV would be replaced by a general data copy and extraction program. The output of the latter program would have exactly the same logical structure as its input. If the new CPLOT was called to produce an l-v plot of this output, it would be given exactly the same user input as in the case it was used on the original data file. The only difference the user would notice is the amount of time required to run the program.
The appearance to the user of the new CPLOT would be somewhat different from the old program. Whereas the old program asks the user the SET,SUBSET to indicate the input, the new program would only ask the SET. Additional information to be supplied by the user would be the kind of plane to be plotted (e.g. "L-M" or "M-V"), the coordinate(s) of the plane and the area within the plane. (Sensible defaults should of course be present.) The area could be given in a number of different formats, also depending on the kind of plane to be plotted.

5. How could the new structure be implemented?

I propose to use two files for each structure: one to contain all pixels in the N-dimensional structure and one to contain all header items. Header items will be implemented as records which can be accessed by specifying a keyword together with a set of coordinates which determine the place in the hierarchy. A special value for a coordinate can be used to indicate that this specific coordinate is not relevant. Again using a 3-dimensional Westerbork "data cube" as an example, a SET header item would be accessed by a keyword and three coordinate values which all have the value "irrelevant". A SUBSET header element would have "irrelevant" values for the l- and m-coordinate and a specific value (SUBSET-number) for the v-coordinate. For a pixel header element all three coordinates would have a value.
Because all information in the header will be stored at the right place in the hierarchy, I do not expect that we will need a variable record size. For instance there is no need to have a list of velocities for each SUBSET in the SET header, but all SUBSET headers could have the information relevant for that particular subset. If, however, all channel maps would have the same velocity, or if the velocity of each map could be expressed by an algorithm, this would only be stored in the SET header. Note that exceptions are possible. If an extra map would be added to the SET, which would not obey the general rule, a keyword item for velocity could be written in that particular SUBSET header.
If it would be necessary to have a variable record size, I think this can be implemented at a limited cost. The only things I can think of that might require this, are user comments and a processing history.

Data access would be performed by a set of routines which are a generalization of the present READXY/WRITXY pair {3}. These would enable a program to read/write parts of any shape (0...N-dimensional) from the data file.
Depending on the possibilities of the operating system under which the new

GIPSY will run, there could also be routines which map the data file onto the program's address space, so that the data can be accessed by the program as if it was an array in central memory.

6. Basic data access routines

Create (id, ndims, dims, error)

id	identification	i
ndims	number of dimensions	i
dims	array containing the dimensions	i
error	error return	o

Exist (id, ndims, dims, error) {logical function}

id	identification	i
ndims	number of dimensions	io
dims	array receiving the dimensions	o
error	error return	o

Delete (id, error)

id	identification	i
error	error return	o

Read-raw (id, pixaddr, npix, data, error)

id	identification	i
pixaddr	address of pixel in file	i
npix	number of pixels	io
data	array to receive pixel values	o
error	error return	o

Write-raw (id, pixaddr, npix, data, error)

id	identification	i
pixaddr	address of pixel in file	i
npix	number of pixels	io
data	array with pixel values	i
error	error return	o

Read (id, lowdim, highdim, npix, data, error)

id	identification	i
lowdim	array with coordinates specifying low-address corner	i
highdim	array with coordinates specifying high-address corner	i
npix	max. number of pixels to read resp. actually read	io
data	array to receive pixel values	o
error	error return	o

Write (id, lowdim, highdim, npix, data, error)

id	identification	i
lowdim	array with coordinates specifying low-address corner	i
highdim	array with coordinates specifying high-address corner	i
npix	max. number of pixels to write resp. actually read	io
data	array with pixel values	i
error	error return	o

The routines Read and Write can be made more powerful if "lowdim" is allowed to have a higher value than "highdim". This would introduce the possibility to "mirror" the data. Another improvement could be obtained by allowing the caller to specify the order in which the coordinates are applied. This would give the possiblity of transposing the data.
Because the coordinate arguments are specified in terms of file coordinates, there should be utilities which construct coordinate arguments from physically meaningful parameters.

7. Basic header access routines

Read header item (id, keyword, ndims, dims, data, error)

id	identification	i
keyword	name of header item	i
ndims	number of coordinates specified	i
dims	array with coordinates specifying level for this item. The first element contains the highest level coordinate. If element =-1, this coordinate is irrelevant. If ndims < number of dimensions in file the remaining coordinates are assumed to be -1.	i
data	header item contents (CHARACTER)	o
error	error return	o

If this routine cannot find the requested item at the specified level, it will search at higher nodes in the hierarchy until it finds the item there or until it knows the item does not exist.

Write header item (id, keyword, ndims, dims, data, error)

id	identification	i
keyword	name of header item	i
ndims	number of coordinates specified	i
dims	array with coordinates specifying level for this item. The first element contains the highest level coordinate. If element =-1, this coordinate is irrelevant. If ndims < number of dimensions in file the remaining coordinates are assumed to be -1.	i
data	header item contents (CHARACTER)	i
error	error return	o

Delete header item (id, keyword, ndims, dims, error)

id	identification	i
keyword	name of header item	i
ndims	number of coordinates specified	i
dims	array with coordinates specifying level for this item. The first element contains the highest level coordinate. If element =-1, this coordinate is irrelevant. If ndims < number of dimensions in file the remaining coordinates are assumed to be -1.	i
error	error return	o

Read raw header record (id, irec, keyword, ndims, dims, data, error)

id	identification	i
irec	header record number	i
keyword	name of header item found	o
ndims	size of dims	io
dims	array to obtain coordinates The first element receives the highest level coordinate.	o
data	header record contents (CHAR.)	o
error	error return	o

Note: "Write raw header record" is potentially dangerous for the integrity of the header structure and for this reason I think it should not be made generally available.

8. Higher level header access routines

On top of the basic routines a layer of higher level header access routines can be built. The following list is what I think is worth consideration.

- A routine to combine header access with user input. This involves the design of a structured prompt (USRINP keyword) to reflect both the header keyword and the coordinate information. There should be the possiblity to choose whether or not to store the user input in the header.
- Routines for different types of header information: integers, reals, etc. These routines should also apply algorithms which are present in the header or referred to by the header.
- A routine that copies headers from an existing file to a new file. It should be possible to copy selectively in a flexible way.

This list should and will be extended, but it should not become too long, because header access required by application programs will be done through these routines, not through the basic routines.

9. Implementation of the data file

The implementation of the data file would be quite simple. It probably would only consist of a big array on disk filled with pixel values.

10. Implementation of the header file

Though header access will be infrequent relative to access of pixel data, it is important that the implementation is sufficiently efficient. Storing header data in an unstructured fashion in a file and retrieving header records by means of a linear search certainly would not be adequate.

The method I propose is a hash technique. (A description of hash techniques can be found in {4}.)
First all coordinate information in the header access call is packed in a 32-bit integer in such a way that all components can be retrieved later. Except when the number of dimensions is large, the maximum total space available for pixel storage would be about 4 Giga-pixels, which I think is quite acceptable. The advantage of this packing is that only a fixed space in the record, independent of the number of dimensions, need be reserved for coordinate information.
Now this 32-bit number together with the name of the header item is "hashed" to obtain the address of an entry in a table. Every entry in this table points to a linked list of header records. Provided that the table is large enough -in the order of the total number of header records to be stored- the linked lists are short. In practice this means that any record can be retrieved in a very small number of disk accesses (on the average less than 2). This number is independent of the total amount of records stored, but only depends on the ratio (number of records)/(table size). The table need not be present in memory but also can reside on disk. Because the address in the table is known in advance, no search in the file is necessary and the record number in the file containing the required table entry is immediately known. The result of this is that a disk-based hash table adds exactly one extra disk access in the header access procedure. If the header file is mapped onto memory, header access probably would even be more efficient.

A disadvantage of this approach is that header items on a specific level of the hierarchy are not grouped together but may be scattered through the file. Header records are stored in the order in which they are written for the first time. If all header items on a specific level are required, the only way to obtain them is reading all records in the header file.
If this is not acceptable, a combination of a B-tree organisation for coordinate information and hashing for the keywords (still combined with coordinates) might be a solution. A different solution might be maintaining two hash tables and inserting the records in two separate linked lists.

11. Other data structures for GIPSY

This document in principle only has the image data structure as its subject ("image" in a broad sense, encompassing all 0...N-dimensional structures). I believe, however, that the just described approach has enough promises that it can be used for non-image data. As a next step, we should investigate the possibility of having one integrated format throughout GIPSY for astronomical data. Except for images, this format should at least allow for the kind of data which now often is stored in tables.

12. Concluding remarks

To make the new data structure a success, there is more to be done than just making a good structure. Perhaps more important is that programs make a good use of the structure. To do so, programs should not make assumptions about the nature of the data but use the header to obtain the facts. Strange enough this will lead to the removal of astronomy from many programs but having done so these programs will be better tools for doing astronomy.

References:

{1} STER program documentation.
(unpublished, available in library of Kapteyn Laboratory)

{2} GIPSY documents MAPFILE.DC3 and HEADERS.DC2.

{3} GIPSY document READXY.DC2.

{4} D.E. Knuth:
"The Art of Computer Programming", Vol. 3, Chapter 6.4, pp. 506-549.