Using Visualization for Visualization: An Ecological Interface Design Approach to Inputting Data

H. Wright[a,*], C. Mathers[a], J. P. R. B. Walton[b]

[a]*Department of Computer Science, University of Hull, Hull HU6 7RX, UK*
[b]*Numerical Algorithms Group Ltd, Jordan Hill Road, Oxford OX2 8DR, UK*

Abstract

Visualization is experiencing growing use by a diverse community, with continuing improvements in the availability and usability of systems. In spite of these developments the problem of how first to get the data in has received scant attention: the stock approach of pre-defined readers and programming aids has changed little in the last two decades. This paper proposes an entirely new way of inputting data for scientific visualization that employs rapid interaction and visual feedback in order to understand how the data is stored. The approach draws on ideas from the discipline of ecological interface design to extract and control important parameters describing the data, at the same time harnessing our innate human ability to recognise patterns. Crucially, the emphasis is on file format discovery rather than file format description. In contrast with conventional approaches, the method can therefore still work even if nothing is known of how the file was originally written, as is often the case with legacy binary data.

Keywords: File organisation, Information search, Ecological interface design, Pattern recognition, Scientific visualization

1. Introduction

Visualization provides scientists, researchers and engineers with an invaluable tool for understanding their data. Since coming to the fore in 1987 [1], work to improve the usability of visualization systems has variously addressed the problem of data representation [2], technique selection [3], and satisfying certain goals or interpretation aims [4–6]. More recently, improvements have been made in securing the provenance and reproducibility of visualizations [7, 8] and in tackling the complexity of using visualization software [9, 10]. However, the problem of data input continues to receive comparatively little attention, even though it is estimated by some expert practitioners to consume up to 90% of the effort when visualizing clients' data [11].

In this paper we present a completely fresh approach to inputting visualization data that uses visualization at the earliest stages, even before the file structure is completely understood. After summarising conventional approaches we show how the prin-

*Send correspondence to h.wright@hull.ac.uk

cipal features of ecological interface design (EID) – work domain analysis, the abstraction hierarchy of means-ends relations, and constraints arising from the work domain – contribute to defining a mechanism for inputting scientific data destined for visualization. This approach encourages thinking about the constituent activities of file input as a set of distinct methods, rather than the traditional approach of constructing self-contained recipes. We go on to describe our software implementation of these methods and illustrate the approach with two interpretations of binary data carried out *a priori*. The paper concludes with a review of the intended scope of the work and its contribution to both the scientific visualization and EID disciplines.

2. Existing Approaches to Inputting Data

Data intended for visualization falls into two broad types: it exists either 1) in some pre-defined format or 2) in some user-defined or unpublished format. Herein may lie the reason for the lack of attention paid to this problem, since the former situation is generally considered to be resolved whilst the latter appears unresolvable in general. It is indeed tempting to assume that inputting data according to some pre-defined standard is a straightforward matter, but sometimes it is not. Visualization software that supports multiple file formats usually requires a different reader function for each format, so the development effort required can be substantial. In the field of chemistry alone, for example, there are over fifty different file formats in use [12].

There are different types of visualization software but all employ essentially similar approaches to the support of pre-defined formats. Turnkey visualizers that are designed to work without customisation by the user generally have a number of file formats that they can import; systems begin by supporting a small subset of definitions which grows over time with each new release. The class of general-purpose, customisable tools known as modular visualization environments (MVEs; [13]) all include a set of reader modules for common, pre-defined file formats, though this set may differ from one MVE to another. As with turnkey systems, it is common for the number of reader modules to increase as a system matures due to the contributions of vendors and aficionados.

Where the generating application writes data in some user-defined, unsupported or unpublished format, the functionality of the visualization system must be extended, either by programming code to read the file or describing its format using a built-in tool. For example, IRIS Explorer's API allows users to incorporate their own code into the visualization system by encapsulating it within a series of 'wrappers' [14]; VTK [15] provides various reader classes that the user can extend to fulfill their needs. Examples of built-in tools for file input are IRIS Explorer's QuickLat [14] and DataScribe [14] tools, the Data Prompter and General Array Importer which are components of Open DX [16], and the AVS/Express Add File Import Tool [17].

Help is therefore at hand, but tackling the problem at the programming and file description level ignores a key element: users of scientific visualization often know something of how their data may look, or have useful clues as to how it is stored or was generated. They may be able to recognise when the representation is faulty, but such recognition usually begins only after the data has

been read in, when the visualization process itself is underway. Making better use of partial knowledge *during* file input has motivated a new approach—a mechanism to input data on a per-solution basis, incorporating interaction combined with visual feedback to guide the process. This philosophy is in complete contrast with existing methods that require prior knowledge of the file's format in order to make any attempt at reading it. Furthermore, because the process is intentionally iterative there is no tendency for the whole reader to collapse entirely when any small detail is overlooked. The result is a set of tools which, whilst they can be used in the conventional way to apply known formats, can also be used to mine visually for unknown file storage parameters. This includes binary files for which, if nothing is known, nothing can easily be gleaned. It is this latter property which proves the most valuable, in some circumstances yielding complete solutions for file input problems that would otherwise be completely intractable. We describe our method as an ecological approach by analogy with the process control industries, where operators oversee and adjust outputs by direct interaction with information coming from the production environment. The next section first describes EID in its familiar context and then goes on to apply it afresh to data file input.

3. Inputting Data the Ecological Way

3.1. Principles of ecological interface design

The EID framework was devised to reduce the rate at which human errors arise during the control of complex systems and to mitigate their effects should they occur [18]. The framework recognises three cognitive control mechanisms, namely skills, rules and knowledge (SRK; [19]), which in turn give rise to skill-based, rule-based and knowledge-based behaviour (SBB, RBB, KBB). The EID approach aims to support interaction at the lowest appropriate level of control, whilst at the same time providing support for higher levels, as needed [18]. Thus for the most part the user will simply be reactive to signals provided by the display (SBB), and preferably will act on the display directly. At a higher level the use of rules may be prompted by the emergence of familiar scenarios (RBB) or, more rarely, problem-solving activity (KBB) will be undertaken on the system as a whole [20]. In this way, efficiency can be maximised during normal operations, whilst at the same time sufficient information remains available to address unusual situations safely.

A valid question at this point is whether EID, conceived in order to reduce the effects of human error, is ever going to be applicable to the problem of file input. If we adopt the conventional approach of a file reader working from an established format then the answer is surely 'No', since the reader will either work if the format is correctly understood or fail if not. The human has little to do in this situation, other than to look for another reader. However, the approach we will take is an unconventional one of testing various hypotheses about the type of the data, the size of an array and the meaning and use of the variables. Such a *trial-and-error* approach may well already be the technique of last resort for inputting unknown file formats but can be impossibly slow when the parameter search space is very large and the support tools are weak. As such we can see parallels with the principles of EID: the need for efficiency (in this case, to test many hypotheses); the need to correct a previous assumption or action;

and, the need for visibility of the current (possibly imperfect) overall system state.

Application of the EID approach in the process control industries typically results in the integration of a number of different display elements. Sensor data, constraints and targets may well appear both graphically and numerically, and support direct interaction with the observation surface (via touchscreen or mouse) in order to control the plant. Displays are therefore configured to allow the system state to be perceived directly and, furthermore, in a way that affords correct (or corrective) action (see, for example, [21]). This property of goal-oriented behaviour resulting from perception of the environment is central to the well-known ecological theory of visual perception [22], from which EID in turn derives its name. A key difference, however, is that the natural ecology (our observed surroundings) is visible whereas the system state is normally invisible. Much of the work of designing a specific ecological interface therefore lies in characterising system variables and their inter-relationships; accomplishing this task for the case of data file input is the subject of the next two subsections.

3.2. An abstraction hierarchy of means-end relations

The starting point for any EID approach is first of all to carry out a work domain analysis (WDA), whose aim is to present the system at different levels of abstraction, the so-called abstraction hierarchy (AH). Levels are linked by means-end relations that answer the questions Why?, What? and How? to achieve a particular goal [23]. For example, interpreting a byte sequence in a particular way (the 'How?') is the means to the end of understanding the numerical values in a file (the 'Why?') and the concept (the 'What?') relating the two is the primitive type of the data (Figure 1(a)). The means-end relations are not anchored to any particular level: manipulating the number and sizes of an array's dimensions, for example, is the means, along with the length of the chosen primitive type, of ensuring it accounts for all the values in a file (Figure 1(b)). To enable flexible support in unanticipated scenarios, the AH aims to capture all such links and, importantly, to show how decisions may interact. Work domain analysis is thus very different to straightforward task analysis, which more closely reflects the format-descriptive approach of conventional file readers.

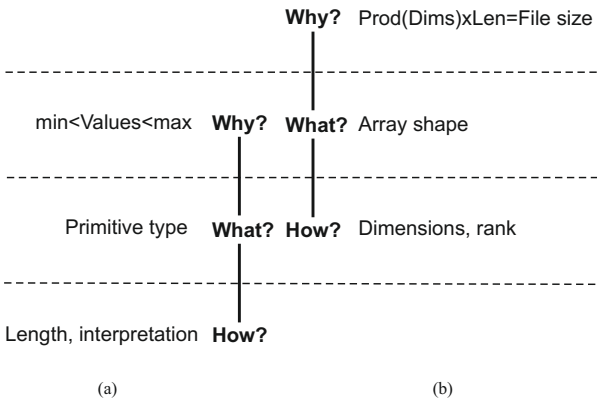


Figure 1: A selection of the purposes and functions pertaining to data input, organised as means-end relations. Means-end relations answer the goal-oriented questions 'Why?', 'What?', and 'How?' (cf. [23], Fig. 7.10).

In process control five levels are identified ranging from the topmost, functional purpose, through to the lowest, which is the physical form of the plant itself. For the current application the functional purpose (FP; Figure 2(a)) is trivial to identify – it is simply the overall description of the task, namely 'File input'. The focus of the present approach is to interpret binary

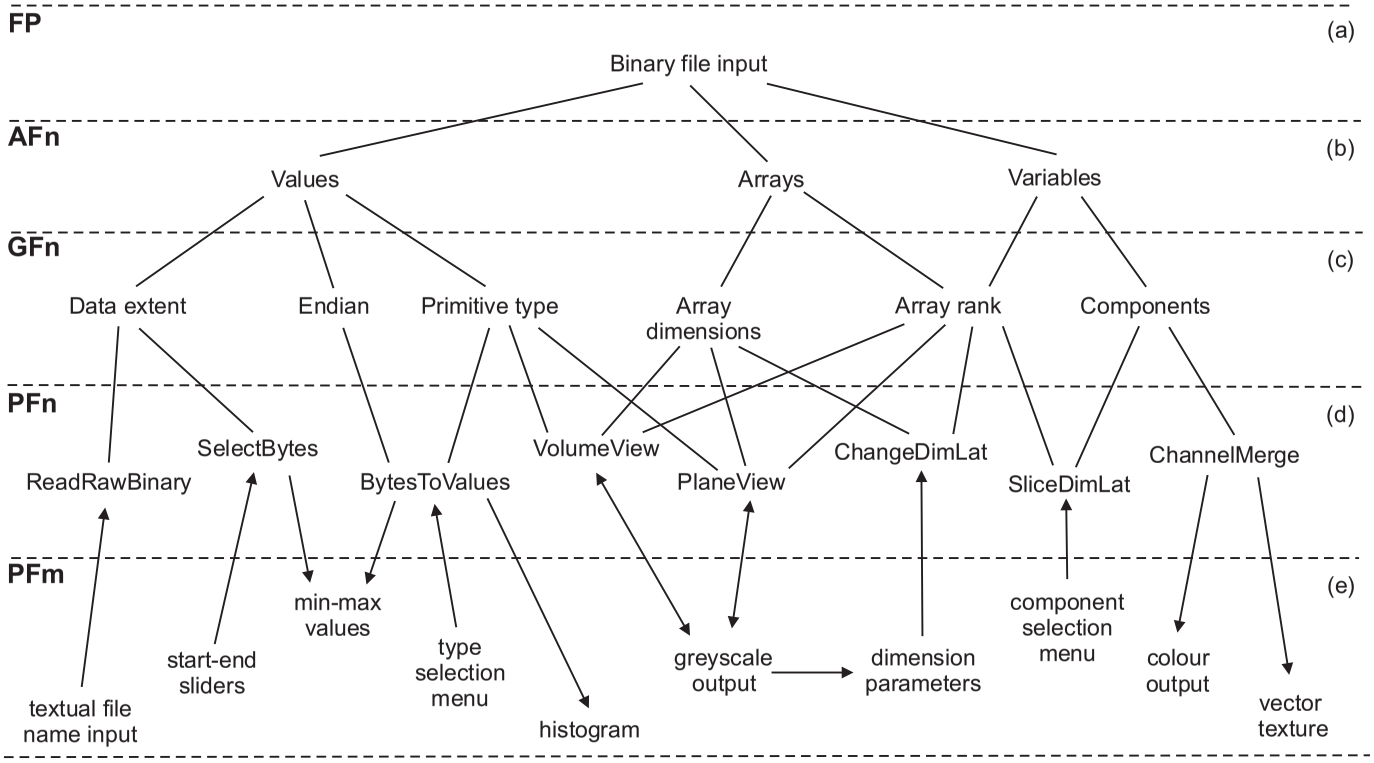


Figure 2: The abstraction hierarchy of means-end relations for the file input work domain. The three upper levels develop an increasingly specific description of the functions involved. Level four, the Physical Function level, depicts the most important of the software methods that realise these functions, whilst the lowest level (Physical Form) contains the control mechanisms applied to these methods. The software implementation of the approach is described in more detail in 4.

data, as opposed to the far easier problem of reading text files, so we can immediately refine this to 'Binary file input'. This defines the 'Why?', but What? does a binary file consist of, usually? Conventional approaches assume we already know whether we are reading e.g. 'float' or 'double' values, whether gridded data is stored in row-major or column-major format and whether there are single or multiple values associated with each variable. In abstract terms we are therefore interested in the make-up of values in the file, the parameters governing how these values are organised into arrays and, if the data is multivalued, the make-up of variables from these values. These objects reside in the next level of the AH and, as in process control, we term this the abstraction function level (AFn; Figure 2(b)).

To populate the next lower level of the hierarchy we pose the questions 'How values?', 'How arrays?' and 'How variables?'; the answers to these questions provide us with objects for the generalised function level (GFn; Figure 2(c)). For example, to find the values will involve finding the extent of the data within the file, whether it is stored with least significant or most significant byte first (its 'endian'), and knowing the primitive type. Working out what variables are in the file may be more involved: for example, an RGB image file is conventionally described as a 2D arrangement of three data values but it might initially be read as a 3D, single-valued array. In this case a reduction in the rank of the array holding the constituent values is needed, possibly followed by a re-ordering step to match the application's convention for red, green and blue components.

Moving the means-end relations down a level, the objects occurring in the generalised function level (now the 'Whats?') characterise the various unknowns when interpreting a data file: they will be familiar to anyone who has tried to program a file reader. Asking 'How?' gives us the software methods in the level below, the physical function level (PFn; Figure 2(d)). Here we find, for example, the means to specify the primitive type in order to interpret values, and the lattice manipulators (a 'lattice' is IRIS Explorer's term for an array) that help with changng array rank, re-dimensioning arrays and extracting components of variables. Separating the different requirements for file input into distinct methods is important because it conveys the flexibility to tackle different aspects of the task independently of one another according to need. For instance, whereas a conventional file reader may be constructed from the outset to read a 2D array with multiple values, in the EID approach this same sequence of values might first be interpreted as a 3D array of single values whose rank then has to be reduced. Similarly a 2D array of long integers might begin life as twice the number of short integers, until the error is detected and the primitive type is reinterpreted. This ability to prioritise the examination of just parts of the work domain is an important feature and, because the levels are linked by means-end relations, the AH is especially suited to this problem-solving approach [23].

Another distinguishing characteristic of the EID approach to file input is the role of the user in validating the operations in the physical function level. To support this we have to define appropriate interaction mechanisms, and to do that we shift the Why?—What?—How? links down again, this time to centre the 'What?' on the physical function level. This poses questions such as 'How select bytes?', 'How assign bytes to values?' and so on. As before,

the answers to these questions are found in the next level, which now is the physical form level (PFm; Figure 2(e)). For example, viewing data value minima and maxima will often determine whether bytes from the header have been erroneously included, prompting the selection of a different subset of bytes by means of the selector's start-end sliders; viewing a histogram of data may indicate, by the lack of correspondence with the expected values, when a different primitive type selection is needed. Figure 2(e) shows the interaction mechanisms we typically use, together with their relationships to objects in the level above. Upward-pointing arrows depict user input to the indicated software methods; downward-pointing arrows lead to (principally) graphical outputs with diagnostic capability; two-way arrows signify interactive graphics performing both an input and output role.

3.3. Behaviour-shaping constraints

Behaviour-shaping constraints arising from the above analysis are important for placing limits on operators' actions whilst at the same time allowing flexibility of response [23]. Looking at Figure 2 we can see that we need to identify, at various points, which one of a set of choices is correct. For example, when determining how many bytes make up each data value, the user must decide which of just 11 primitive types is most likely to describe the data. The means-end relations show that viewing histograms of the resulting values could provide important evidence when making this decision. In WDA terms, the number of possible choices is constrained because values made up of arbitrary numbers of bytes simply do not occur in the majority of computer architectures. We might thus expect the operator to adjust their choice for this parameter, based on the changing evidence of the histogram, until the values fit some known criterion for the data set under scrutiny.

Another constraint becomes apparent when combining values to make up the different application variables. For example, this might involve extracting red, green and blue pixel values in the correct order, or determining the sequence in which some three-dimensional vector components are written. Provided the number of components, C, of a variable has been determined correctly, there are $O(C!)$ ways to recombine them. For scientific visualization this tends to be tractable since data is usually only moderately multivalued, for example, a volume containing temperature, pressure and a flow vector presents just five separate values to be interpreted.

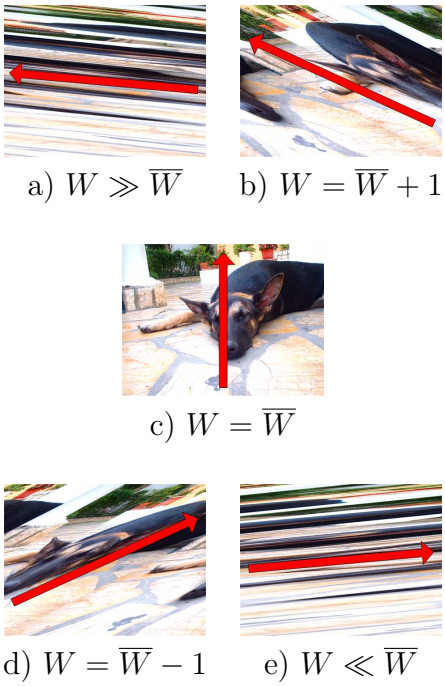
When interpreting values and variables the set of choices is therefore modest in size and the correct interpretation will typically follow without much difficulty. The situation is very different, however, when interpreting array structures. Here, the essence of the problem is to find the rank and extents of the array that holds the values. Let us take a 2D grid of data for illustration, where N data values in total have to be arranged into a rectangular domain of width $\overline{W}$ and height $\overline{H}$. Clearly $N = \overline{W} \times \overline{H}$, so a brute-force search of all possible combinations of trial widths and heights W and H could require the user to scrutinise up to N visualizations, in other words with $(W_i = i, H_i = N/i)$ for $i = 1$ to $i = N$. This is clearly impossible for most files but we can observe that candidate (W_i, H_i) need only be tested if $N = 0 \ mod(i)$, that is, (W_i, H_i) must be factor pairs of N.

Here, then, is a key constraint of the sys-

tem: the area-invariant property of a rectangle which, as it grows in one dimension, must shrink in another to continue to accommodate a fixed number of values. This observation in itself is not enough to arrive at a viable interaction mechanism for finding the right factor pair: an empty rectangle that indicates its allowed sizes when adjusted by the user helps in knowing how many factor pairs exist, but not in choosing which is correct. However, if the values are represented *in situ* within this rectangle, we find that this additional visual feedback renders the problem suddenly quite tractable.

Figure 3 illustrates the approach using an image file of 1228800 elements, whose real width and height, the unknowns of the problem, are 1280 and 960. The sequence shows portions of the images that result as trial widths above and below $\overline{W}$ are used. When the trial value is very far from correct, apparent lines in the image are narrowly spaced, with a shallow negative gradient for $W \gg \overline{W}$, positive for $W \ll \overline{W}$. As the trial width is adjusted the lines widen, reaching a maximum gradient when $W = \overline{W} \pm 1$, before snapping to the correct image of a rather sleepy dog.

More usually after this operation the value found is not $\overline{W}$ itself, but some other value that belongs to an incorrect factor pair. A two-stage approach is therefore generally necessary: first, the rectangle is adjusted as above to find a (usually incorrect) factor pair; second, the correct factor pair is determined from the appearance of this intermediate finding. Figure 4 illustrates this second stage. Here, the trial width and height are 1920 and 640 respectively, compared with the real width and height of 1280 and 960. In spite of this error, three repeats of the subject matter can still clearly be seen in the horizontal direction. Looking at



a) $W \gg \overline{W}$ b) $W = \overline{W} + 1$

c) $W = \overline{W}$

d) $W = \overline{W} - 1$ e) $W \ll \overline{W}$

Figure 3: A sequence of images illustrating the effect of altering the trial width W, which causes the striped pattern to widen, until the image appears.

a magnified version of the display (Figure 5) also reveals less obvious twofold interleaving in the vertical direction, which manifest as apparent stripes of colour running across the image. If we denote the number of these horizontal and vertical repeats as r_h and r_v respectively and the trial width as W, the required actual width of the image $\overline{W}$ can then be calculated as

$$\overline{W} = \frac{r_v W}{r_h} = \frac{2W}{3}$$

In this example the trial image is now made about two-thirds of its current width, and final adjustments towards the real width are made using the oblique line patterns shown previously. The resulting image, now correct in both width and height, looks exactly like Figure 3(c).

Hunting visually for factor pairs is possible due to the extraordinary pattern recog-

Figure 4: A colour image of a sleeping dog, illustrating the horizontal repeating pattern that results when the trial width is an incorrect factor of the total number of elements.

Figure 5: A magnification of Figure 4 illustrating the corresponding vertical interleaved pattern.

nition ability of the human visual system, coupled with the natural spatial coherence of data values that is present in files of scientific data. Provided there are features within the data, each persisting over several rows of the display, distinctive patterns consisting of sloping lines and repeated motifs will always develop when the trial width is adjusted. The sloping lines occur because, in each new row in the display, the start of a feature is offset from its start position in the previous row; over several rows the effect is to smear the feature. Our visual system is always striving to model what we see, so the individual features, all smeared to the same degree at any one instant, are conflated into a set of sloping lines. Repeated motifs occur when trial widths are integer or rational number multiples of the actual width. In these cases the start position of a feature lines up several times in successive rows, or in every second, third, fourth and so on row. Over many rows and many features, more than enough information remains for recognition, as Figure 4 shows.

The next section describes IFIT (Interactive File Input Toolkit), the set of software tools we have implemented to utilise these constraining properties.

4. Software Implementation

The toolkit we have developed is based on the IRIS Explorer MVE, which we recall from the introductory discussion allows code to be added to the released software by means of 'wrappers'. In this way we can re-use existing functions if they are already available or add them if needed, and to the toolkit user the resulting modules are indistinguishable from those provided by the vendor. It should be noted that use of an MVE is not a requirement of the approach

– we could instead have developed a class library of methods and a template calling program.

The modules used fall into two categories: those performing transformations of data and those providing the user with feedback on the current state of the file interpretation.

4.1. Transformation modules

Modules in this section of the toolkit allow for the selection and manipulation of the data. Existing modules in the IRIS Explorer release that are useful in IFIT include modules for splitting and recombining variables such as MultiChannelSelect, ChannelSelect and ChannelMerge. Manipulation of raw bytes is hardly ever needed in visualization so here we had to add new modules ReadRawBinary, SelectBytes, and BytesToValues. Choices to be made during the file input process are realised as parameters in the user interfaces of these modules so, for example, SelectBytes has sliders to determine the beginning and end of the byte sequence to be examined, specified relative to the start or end of a file (Figure 6). Other modules implement or apply the constraints identified in Section 3.3: BytesToValues has a menu listing the primitive data types 'unsigned int', 'short', 'float' and so on, so the user can easily experiment with the number of bytes in the data type, and their interpretation (Figure 7). We also added a module ChangeDimLat for generally redimensioning arrays, with sliders for receiving (from an upstream module, typically PlaneView or VolumeView, see Section 4.2), or specifying, up to ten array dimensions (Figure 8). SliceDimLat allows extracting reduced-rank arrays from higher-dimensioned structures, which can then be recombined using ChannelMerge.

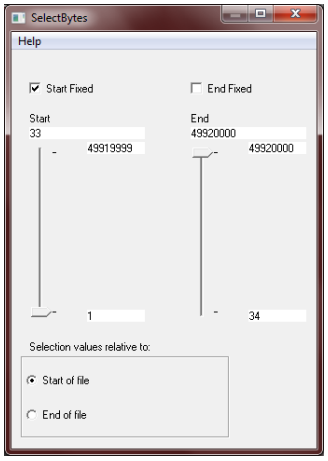


Figure 6: Clipping a 32-byte header from the start of a 50MB binary file. The end selection is here left to run to the full extent of the file, without the user needing to know how many bytes this comprises. Selecting values relative to the end of the file would conversely clip a 32-byte footer.

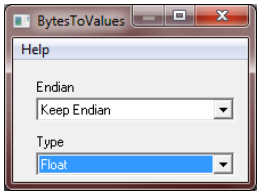


Figure 7: Interpreting values by constraining raw bytes to combine only in particular configurations allowed by the computer architecture.

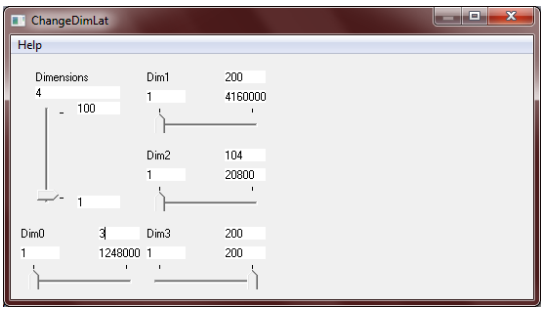


Figure 8: Dimensioning an array of values in order to apply the volume-invariant constraint $W \times H \times D = 4160000$ to a 3-variable data set. This module is typically driven from upstream or can be adjusted by the user.

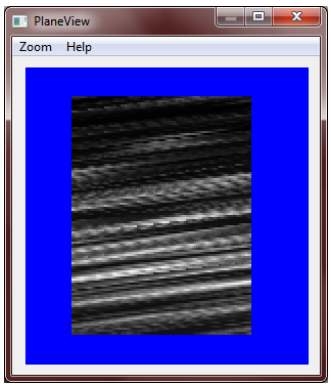


Figure 9: Moving the mouse left or right respectively increases or decreases the trial width, until some recognisable structure is seen. A zoom option allows investigation of artifacts such as the vertical interleaving in Figure 5.

4.2. Feedback modules

Existing feedback modules range from something as simple as the PrintLat and MinMax modules that report numerical values on their interfaces, to IRIS Explorer's Graph3D and DisplayImg modules, which respectively display a histogram and image data. A new module was needed to implement the rectangular area-invariant display, due to the rapid interaction needed with this interface element. This module, PlaneView, thus allows direct interaction via the mouse on the display surface itself, which simultaneously shows the current view of the data, drawn as one greyscale pixel per value (Figure 9). In usage the width of the display rectangle is altered continuously by rolling the mouse pointer back and forth horizontally. In order to be salient to the perceptual task [20], movement to the right increases the width whereas going to the left reduces it. The overall effect is reminiscent of tuning an old-fashioned television receiver by turning its dial — as the required setting is approached, the diagonal stripes gradually resolve into a recognisable picture (or, more usually, a repeated form of it). This combination of visual feedback and rapidly trialling widths means that the eligible factor pairs of N are searched extremely efficiently, more than adequately compensating for the inherent size of the problem. Once the final adjustments to find the correct width and height have been made, as described in Section 3.3, these parameters are used by the ChangeDimLat module to dimension the data array. PlaneView can also be used to interpret list arrays and this is demonstrated in Section 5.2.

Similar patterns to Figure 9 were also seen in 3D data read in using incorrect dimensions, prompting the development of a VolumeView module. VolumeView operates similarly to PlaneView, but the addition of another two tiles extends the methods to 3D. Each tile views a different orthogonal cross-section through the array and presents the same visual patterns that are seen in the PlaneView module. Mouse interaction is analogous to that in the PlaneView module, but now operating in two directions corresponding to the two axes of each view (Figure 10). In addition, animation moves each view back and forth along its respective axis, creating a movie of the data in that axis. If the trial dimensions are correct, the effect is one of moving each slice through the data in question. If incorrect, temporal aliasing occurs similar to that seen at a cinema when the film is run a little too fast or too slowly, causing the frame edge to advance up or down the screen. The speed and direction of the movement seen by the user respectively cue the amount and direction of the correction to be applied to the corresponding trial dimension. As in the case of PlaneView, the power of the method lies in its combination of visual feedback and rapid interaction with the array's trial di-

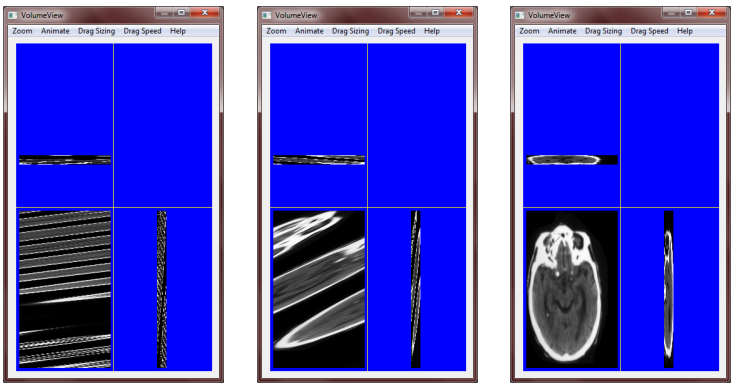


Figure 10: Interpreting an IRIS Explorer lattice-formatted file of a CT scan using VolumeView. The small Z dimension is due to the lower resolution in this direction compared to X and Y.

mensions.

5. IFIT in Use

5.1. Interpreting a flow volume

The first example scenario is a flow volume comprising binary data and no header, which is however known from its file name to be a jet of water captured using particle velocimetry. Two modules have been tried whose help files describe them as binary readers but both rely on having a header in a certain format: accordingly, one gave errors and no output and the other crashed.

An end-user of visualization would not get past this stage but a visualization support person or developer user would probably try writing a program to read the binary file and then look at possible interpretations of the values. By this means the data type could probably be guessed as 'float' but without a header the problem remains as to how to dimension the data. Interpreting the data as a volume with equal sides is a reasonable guess but it gives an isosurface of flow magnitude that is highly confused and fragmented, whereas for a jet of water we would expect a coherent region with different speed to its surroundings (Figure 11).

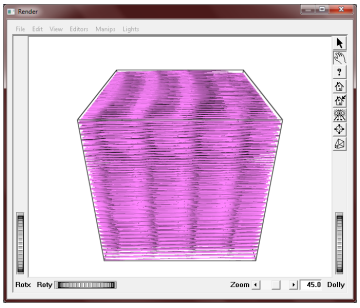
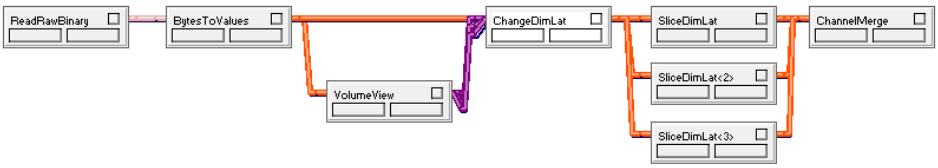


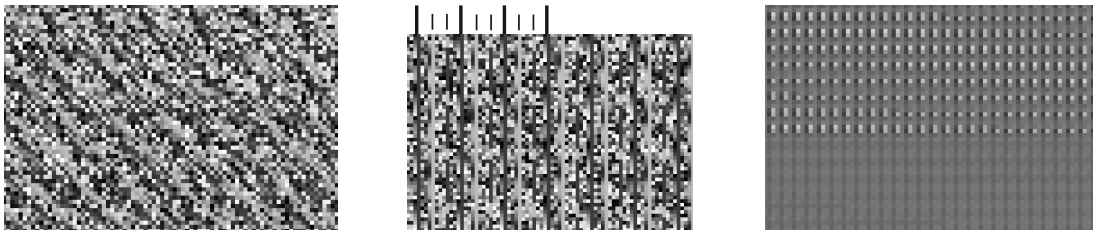
Figure 11: An isosurface of flow magnitude shows fragmented and chaotic appearance that is typical of being wrongly dimensioned.

A few further likely combinations of width, height and depth might be tried but the isosurface takes too long to compute to make an effective search of the enormous space of factor triples that exists for this data.

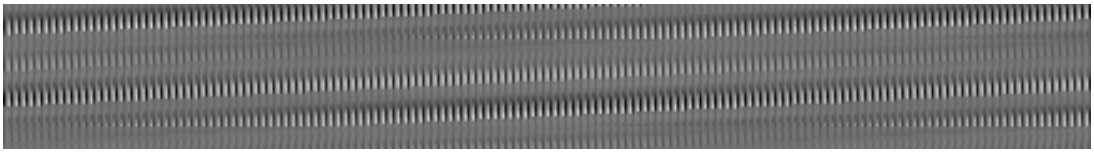
Figure 12(a) shows the IFIT module pipeline created for interpreting this volume of data. The starting point is to read the whole file using ReadRawBinary, which then passes the data to BytesToValues, set initially to give it a trial interpretation of 'unsigned bytes'. In the VolumeView window, arbitrary values for W, H and D reveal a 'pepper-and-salt' texture, left-most in Figure 12(b), that is characteristic of 32- or 64-bit data interpreted inappropriately.



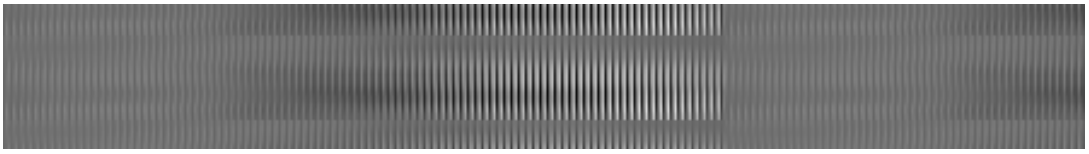
(a)



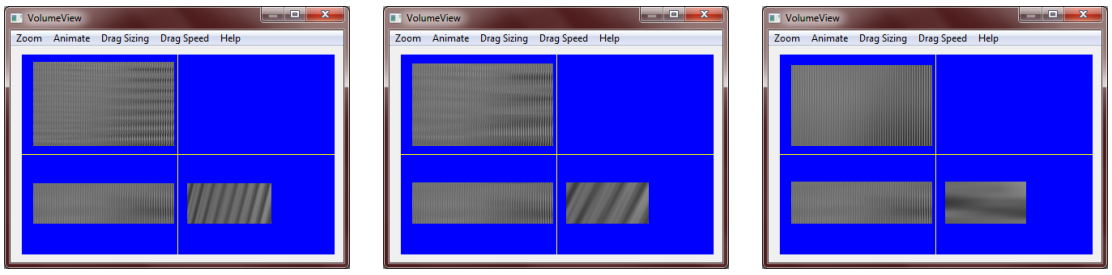
(b)



(c)



(d)



(e)

Figure 12: An IFIT pipeline (a), used first to interpret the primitive type (b), then to find the correct width ((c) and (d)), and finally height and depth (e) of a flow volume.

Slight adjustment of W shows a marked repeat in this texture every 12 bytes overlaying a less obvious pattern every four bytes (Figure 12(b), middle). It is therefore proposed there are three 'floats' per data location, which is consistent with the generating application. Reinterpretation as 'floats' using BytesToValues yields the right-most graphic of Figure 12(b), whose three-fold vertical stripes are characteristic of vector values that are spatially coherent but exhibiting rather different component values. Work now begins on a view of the whole file to determine the correct width, periodically increasing the zoom to check that the vertical fragmentation in Figure 12(b), right, is diminishing. To aid in this, the mouse drag direction is constrained to move only in the horizontal sense. Figure 12(c) shows a typical intermediate where W is a multiple of 3 so the stripes are evident, but an overall sloping line motif shows $\overline{W}$ is still some way off. Figure 12(d) is the first intermediate where W is a rational number multiple of $\overline{W}$: the horizontal discontinuity shows W is 50% too big, but this is now easily corrected and attention turns to finding H and, by implication due to the volume-invariant constraint, D. The mouse drag direction is now constrained to move only in the vertical sense and the visual focus switches to removing vertical repeats. Figure 12(e) shows the whole VolumeView window as H approaches and, finally, matches $\overline{H}$. Note how the sloping lines in the top tile resolve at the same rate as those in the lower right tile. At this stage, $\overline{W}$ is still three times the domain width due to the multiple vector components, but $\overline{H}$ and $\overline{D}$ are correct.

The ChangeDimLat module is prepared in order to create a 4-dimensional array, with Dim0 set to 3 (see Figure 8). Dim1 is connected to VolumeView's output of $\overline{W}$, and divided by 3 on arrival at ChangeDimLat using IRIS Explorer's parameter function facility. The value for $\overline{H}$ flows to Dim2 and $\overline{D}$ to Dim3 (purple wires in Figure 12(a)). Three slices must then be made in the fastest varying dimension, Dim0, in order to extract the volumes of individual vector components, because IRIS Explorer's data model treats the data dimension differently to spatial dimensions. We can think of this stage in terms of re-interpreting the data as three arrays with *rank* 3 and shape $(\overline{W}/3, \overline{H}, \overline{D})$ rather than one with *rank* 4 and shape $(3, \overline{W}/3, \overline{H}, \overline{D})$. These slices are then re-combined by ChannelMerge so they can be processed as components of a single vector. Figure 13 shows the isosurface of magnitude that results, which can be compared with the fragmented version in Figure 11 arising when incorrect dimensions were tried originally.

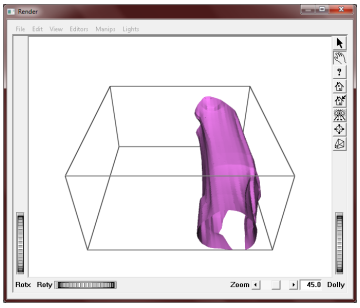


Figure 13: An isosurface of flow magnitude correctly dimensioned c.f. Figure 11.

The pipeline in Figure 12(a) can now be stored with its parameter values intact, in order for it to be re-used later with the same data file.

5.2. Interpreting finite element simulation data

Although one of the simplest arrangements encountered, the gridded data of Sec-

tion 5.1 is very common in scientific visualization and interpretation of any such data file will proceed on the lines described, whether it consists of single or multiple variables, be they scalars, vectors or tensors. Visual feedback can also help with scattered and cell-based data if there are multiple bytes at each node. In this case, rather than find the width, height and depth of the problem domain, PlaneView is used on the file's list arrays to find the number of variables and nodes.

The second example scenario concerns the output of a finite element simulation whose file name indicates it contains 2D flow around a cylinder. We would thus expect to find some nodal coordinates, data variables and mesh of elements but the file has been separated from its metadata so the number of nodes, the type(s) of element and the type and number of variables has to be determined *a priori*.

The interpretation begins with an overall view of the file as unsigned bytes (Figure 14(a)), and confirmation that no single $\overline{W}$ and $\overline{H}$ can be found. With arbitrary W and H the file clearly consists of three sections, the bottom and top of which show the same 'pepper-and-salt' texture as seen previously so probably contain 'floats' or 'doubles'. The middle part also carries a texture but with more marked contrast, and this is characteristic of short or long integers interpreted as byte data. Following some adjustment of W this middle section resolves to the 4-fold pattern in Figure 14(b), so clearly the data are 32-bit, rather than 16-bit, integers and are almost certainly the elements comprising the mesh. At the same time, the top section is found to have subtle eight-fold structure within a more marked overall 24-fold pattern (Figure 14(c)). This section is therefore proposed to consist of triplets of 'doubles', probably relating to the node list. These could be data variables or, if the domain is a 2D manifold in 3D, coordinates. The bottom section shows only an 8-fold pattern with no smaller scale repeat evident within (Figure 14(d)). It is therefore not possible to decide at this stage whether these are 'floats' or 'doubles', but if this section also relates to nodes its size in bytes must be one-third that of the top section since the continuity of the 8-fold repeat is unbroken throughout the section. The suggested related sizes of the top and bottom section are now used to 'bracket' the middle section, by setting up two SelectBytes modules – one to clip Nx8 bytes from the start of the file and another Nx24 from the end, where N is specified using a separate slider widget that drives both modules' parameters. The remaining portion of the file is viewed with PlaneView, and the value for N increased until nearly, but not all, of the 'float' and 'double' sections have been removed (Figure 14(e)). Final adjustments of N are not guided visually but instead using the minimum and maximum values of the long integers in the middle section. As the trial value of N reaches 847, these are reported as zero and 846 respectively, giving confidence there are 847 nodes and some unknown number of elements relating these nodes' zero-based indices.

All that remains now is to assign the top and bottom sections to variables or coordinates, and decide whether the middle section, the element list, can be interpreted further. Recall that it is still uncertain whether the bottom section contains 1694 'floats' or 847 'doubles', but the top section almost certainly consists of 847 triplets of 'doubles'. It would be unusual to store coordinates with greater precision than data, so therefore seems likely the bottom section con-

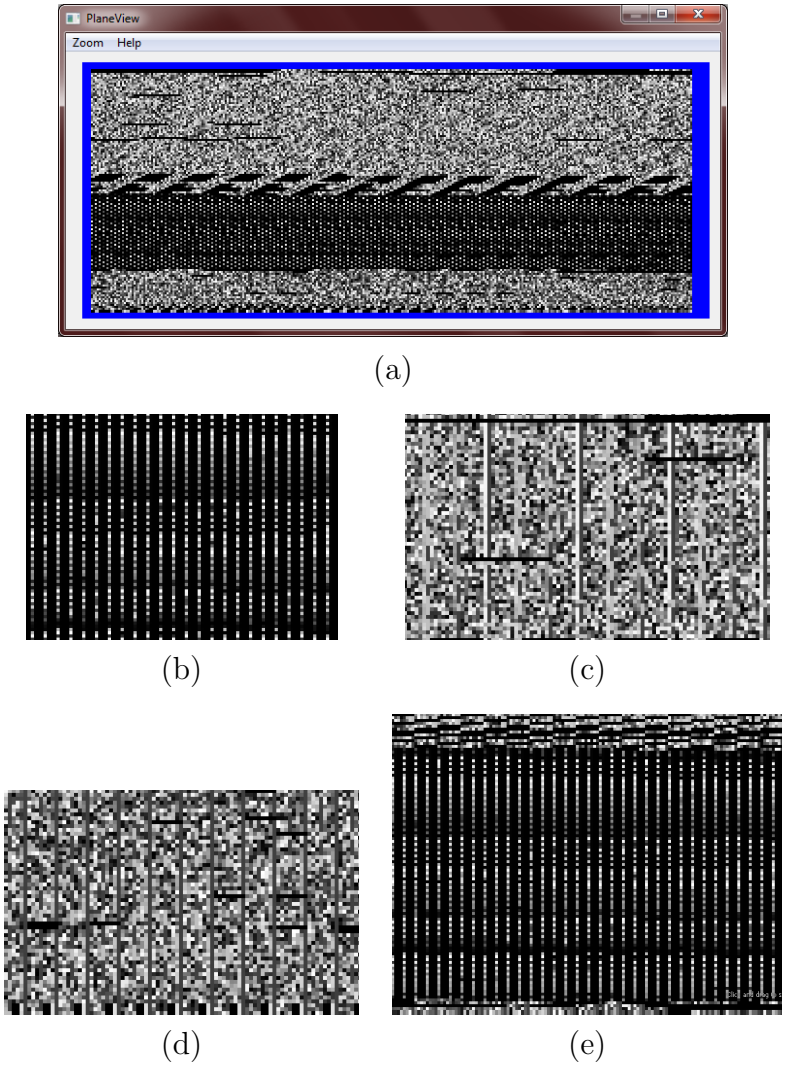
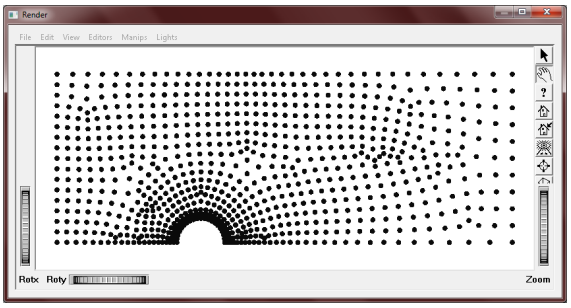
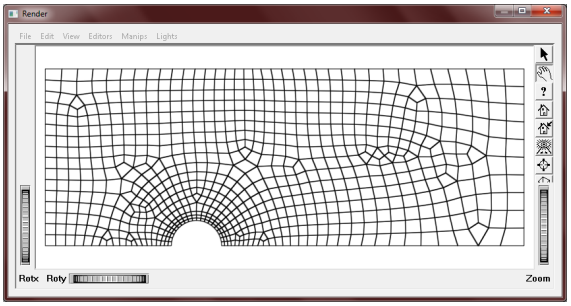


Figure 14: Visualizing a whole file as a sequence of bytes (a), from which it is discovered the middle section contains long integers (b), the top section triplets of 'doubles' (c), and the bottom section 'floats' or 'doubles' (d). Progressively clipping the floating point values away from the top and bottom of the middle section confirms the number of nodes and the likely number of elements linking these (e).



(a)



(b)

Figure 15: Confirming the 'floats' are coordinates (a) and the elements are quadrilaterals (b).

tains coordinates. This hypothesis is tested by plotting the nodes as points, which are seen to fill a rectangle (Figure 15(a)), lending weight to this suggestion. The list of elements linking the points is found from the middle section by observing the number of long integers there is divisible by 4, but not by 3, making it more likely (though not certain) that the elements are quadrilaterals. A mesh constructed using this list and the supposed coordinates looks like (Figure 15(b)). Further investigation of the triplets of 'doubles' in the top section using contours and an arrow plot reveals there is a 2D vector and a scalar rather than a 3D vector.

5.3. Skills, rules and knowledge revisited

The value of the EID approach in its general formulation can be captured as three principles, namely to 1) "support interaction via time-space signals"; 2) "provide a consistent one-to-one mapping between the work domain constraints and the cues [...] provided by the interface"; and, 3) "represent the work domain in the form of an abstraction hierarchy to serve as an externalised mental model" [20]. In short, it should simultaneously support all three levels of SRK-based behaviours (SBB, RBB, KBB). At the skill-based level there are the mouse interactions directly on the display, moving left and right in response to the changing pattern seen there (Figures 9, 10 and 12). The goal of getting the sloping lines to widen is easily learned and is independent of the particular type of file being tackled, so the response quickly becomes innate. At the same time as this low-level control is operating, other visual information is available for rule-based reasoning. As was seen in the examples in Section 5, misinterpretation of the primitive type causes characteristic textures. When the trial width is a multiple of the number of bytes in the actual primitive type, these textures resolve into vertical stripes as in Figure 12(b) and Figure 14(b) to (d). Eliminating horizontally repeating motifs or horizontal discontinuities invokes another set of rules involving the trial width, with the response enacted according to the analysis in Section 3.3. However, *vertical* repeats or vertical breaks in continuity, provided there are not also any interleaving artifacts, are an indication that the data has another dimension and requires input to VolumeView, rather than PlaneView. Figure 16 shows the output of a simulation of flow, temperature and pressure within a double glazing panel that is initially input to PlaneView. The slices appear stacked on top of one another rather than filling the depth of the volume.

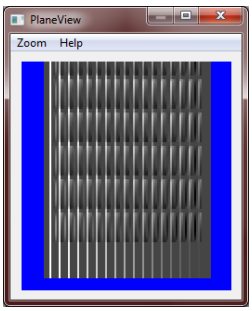


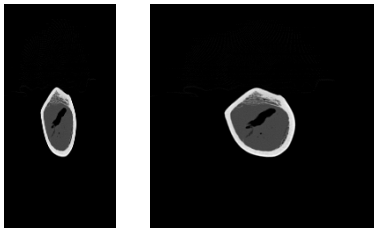
Figure 16: RBB: a simulation output volume initially interpreted as planar data.

(a) (b) (c)

Figure 17: RBB: unexpected 'contours' in blood flow magnitude due to incorrect byte alignment (a). Adjusting the starting point of the byte selection results in (b). Similar contours are seen when this 'short' data is interpreted as 'byte' (c), but this error is distinguished from (a) by also having vertical stripes similar to those seen in Figures 12 and 14.

The data files interpreted in Sections 5.1 and 5.2 did not contain headers but when present they may offset the interpretation from its true starting point. For example, if a byte sequence including the header is interpreted correctly as 'short' data, incorrect byte alignment caused by an odd number of 'byte' types in the header causes apparent contours that are not typical of the data (Figure 17). As well as rule-based behaviour, knowledge-based behaviour played a role in correcting this particular interpretation because the file is a single positron emission scan to measure blood flow magnitudes, which typically vary smoothly through the anatomy.

Knowledge-based behaviour was also key



(a) (b)

Figure 18: KBB: Reading a slice of CT data as half the actual number of double-length values produces distortion (a) in a structure that is known to be roughly circular (b).

in correcting the aspect ratio in the CT data of Figure 18. The data file is a cross-section of femoral bone that should be roughly circular, but a combination of an incorrect dimension and compensating incorrect primitive type gives an elongated shape.

5.4. Discussion

The modules that implement the visual feedback described in Section 3.3 are especially effective for gridded data of the type interpreted in Section 5.1. Provided there is visible variation in the values spanning the domain, they will yield complete solutions even when nothing is known about the file format. Indeed, for this type of data, using IFIT is quicker than compiling a new module to input the file, even if the format is known. As was demonstrated in Figure 10, it is not necessary to interpret header information to find data input parameters governing the array dimensions. However, if this *is* done, the saved pipeline effectively becomes a reader for all files of the type, and not just the particular instance which has been interpreted. We denote these two approaches as providing 'single-use' and 'complete-use' solutions respectively. As was seen in Section 5.2, interpreting cell-based data is less

automatic and requires more problem solving behaviour, but the visual approach enables a strategy to be formulated and carried out. The greater effort required to interpret such a file would nonetheless be well spent if the data was valuable and could not be retrieved by other means.

Visual interpretation can be difficult if the data in the file has periodicity, since it is difficult to separate this from the repetition caused by incorrect factor pairs. Likewise if there is very little variation in values or the variation is rather chaotic the requirement for distinctiveness and coherence, on which the visual tools depend, is not fulfilled. An intriguing possibility if the visual approach became widespread would be to include with all data sets a distinctive inset pattern, in as many dimensions as the data set spans, that acts as a key for interpretation independent of the data itself. This would be the multi-dimensional analogue of the 2D test card that used to be transmitted to aid in tuning an analogue television. For now, if a file yields no visual information of any use when viewed with PlaneView or VolumeView, the transformation modules of Section 4.1 can still be used and the resulting data passed through a standard visualization pipeline for verification. However, in this case the parameters governing the file's organisation will need to be known in advance rather than be discovered. In this case the benefit of the EID approach, that is, work domain analysis and the abstraction hierarchy, lies in expressing file input as a set of distinct methods (Figure 2(d)) that are accessible to a non-programmer user.

Single or multiple values representing heights, CT values, temperature, vector components, and so on, all map to greyscale in PlaneView and VolumeView. These tools use simple display algorithms with O(N) time complexity, so the file interpretation activity proceeds at interactive rates regardless of the ultimate intended visualization technique. More complex and time-consuming techniques such as contouring and isosurfacing are only invoked once the arrays have been dimensioned and variables assigned. The simple display algorithms used have also been made robust to rapid and wide variations in the primitive data type, domain size, and numerical range of values passed to them. The same is not always true of the intended visualization techniques themselves, and is another reason to complete the file interpretation before continuing to the visualization stage *per se*.

At present, vector data is treated as multiple scalars and interpretation proceeds on the lines of Figure 12. Using IRIS Explorer's LatLIC module, we have also used the techniques of Section 3.3 to carry out a proof-of-concept interpretation of a file of vector values rendered to an image as a texture. This may prove to be a more satisfactory approach than treating the vector components individually if the variation required for recognising structure resides in the vectors' directions rather than their magnitudes. However, incorporation into PlaneView and VolumeView themselves would require a real-time algorithm such as [25] to achieve interactivity and remains to be done. Our test data set did not include tensor data but the same general principle would apply, provided an image-based visualization of the tensor field showed sufficient structure, and could be carried out in real-time (see for example [26]). Handling large data sets presents a further concern since IFIT caches all the data in order to produce the characteristic visual patterns, something which is

clearly impossible if a file comprises many gigabytes. To address this would require interpreting a file in sections: for example, inspecting the leading values and first few planes of data in a CT volume would find the header length, primitive type, and volume width and height; the last few planes and trailing values would yield the footer length, if any; combination of these parameters with the overall file size would allow solving for the volume's depth. Once again, implementing a partial-cacheing approach to ReadRawBinary to support this way of working remains an item of future work.

6. Summary and contribution

This paper has described a novel mechanism for inputting data intended for scientific visualization, together with its implementation as a set of modules in the IRIS Explorer MVE. In contrast to current approaches, the toolkit uses both iteration and visual feedback to monitor and correct continuously the parameters that govern the input process. The result is a flexible and multi-purpose facility that doesn't 'fall at the first hurdle', as existing tools are prone to.

6.1. Applications and scope

IFIT has been applied to a number of problems, all of which were real-world data sources provided by a range of users. As well as being successful in importing files in known formats such as DICOM [27], FITS [28], and DEM [29], we have found it to be particularly useful where data files have to be interpreted without detailed knowledge of their formats. These include files written in legacy formats for which the documentation (if originally extant) is unavailable, or files that adhere to a format which has been superseded by an updated version without allowing for conversion from the old format. If the file is binary, such problems can be completely intractable to conventional tools, yet easy to decipher using IFIT.

We have also had experience of a consultancy scenario, in which users provide data files that are to be visualized without giving any description of their formats—either because they didn't know anything about them, or because they were unable to disclose such information. Finally, IFIT has been used to extract data from hardware which stores it in a binary format that is unpublished by the manufacturers. Here, the toolkit gives the user new kinds of access to the data which they have collected—for example, to analyse and visualize it using their own software—but without compromising the intellectual property vested in the format itself.

Our experience in using IFIT is that it is simpler than programming or scripting but harder than using a pre-coded solution. If a reader exists for a file that has been properly identified and conforms to the format, then IFIT is not the tool of choice. However, when a reader must be constructed it presents a viable alternative to current methods. Moreover, for binary files in unknown formats, IFIT may be successful where other approaches fail completely.

6.2. Extending the application of EID

Ecological interface design has a pedigree stretching back more than 20 years [18]. Much work to date has targetted continuous-process systems, describing, for example, the simulation of pasteurisation [24], catalytic cracking [30], and

nuclear power plant [31]. In addition to these classic applications the potential of EID has also been described for increasingly diverse fields, including discrete production management [32], information search and retrieval tasks [33], computer network management [34] and driver assistance systems [35]. The present work demonstrates the application of EID to yet another very different area, further confirming its value for a wide range of tasks.

We implemented our toolkit in an MVE for the purposes of visualization, but the principles are suited to other software genres and, indeed, to other applications. Graphic design, computer forensics, data mining and visual analytics all suffer the same problems of file format specification and data interchange complexity that are seen in visualization, and all potentially are amenable to a visual approach. We hope that our demonstration of the applicability of EID to scientific visualization will encourage other researchers in turn to take a fresh look at their own field.

Acknowledgements

The 'test card' idea was contributed by one of the reviewers; we thank them and all who read the paper for their interesting and helpful suggestions.

Thanks are due to many current and former colleagues at the Universities of Hull and Leeds who contributed data sets for testing, a number of which appear in this paper, including contributions to the extensive test data set that ships with IRIS Explorer, collated originally by Silicon Graphics, Inc. and latterly the Numerical Algorithms Group (NAG) Ltd. We also wish to acknowledge Stanford University and the University of North Carolina for placing many useful data sets in the public domain.

CM acknowledges the UK Engineering and Physical Sciences Research Council and NAG Ltd for supporting a CASE studentship, during which the IFIT software was developed.

References

[1] B. H. McCormick, T. A. DeFanti, M. D. Brown (eds.), Visualization in scientific computing, Comput. Graph. (ACM) 21 (6) (1987).

[2] P. K. Robertson, A methodology for choosing data representations, IEEE Comput. Graph. Appl. 11 (3) (1991) 56–67.

[3] H. Senay, E. Ignatius, A knowledge-based system for visualization design, IEEE Comput. Graph. Appl. 14 (6) (1994) 36–47.

[4] S. M. Casner, A task-analytic approach to the automated design of graphic presentations, ACM Trans. Graph. 10 (2) (1991) 111–151.

[5] B. E. Rogowitz, L. A. Treinish, An architecture for rule-based visualization, in: G. M. Nielson, R. D. Bergeron (Eds.), IEEE Visualization '93, IEEE Computer Society Press, 1993, pp. 236–243.

[6] I. Fujishiro, Y. Takeshima, Y. Ichikawa, K. Nakamura, GADGET: Goal-oriented application design guidance for modular visualization environments, in: R. Yagel, H. Hagen (Eds.), IEEE Visualization '97, IEEE Computer Society Press, 1997, pp. 245–252.

[7] T. J. Jankun-Kelly, K.-L. Ma, M. Gertz, A model and framework for visualization exploration, IEEE Trans. Vis. Comput. Graph. 13 (2) (2007) 357–369.

[8] C. T. Silva, J. Freire, S. P. Callahan, Provenance for visualizations: Reproducibility and beyond, Comput. Sci. Eng. 9 (5) (2007) 82–89.

[9] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Freire, C. T. Silva, Viscomplete: Automating suggestions for visualization pipelines, IEEE Trans. Vis. Comput. Graph. 14 (6) (2008) 1691–1698.

[10] E. Santos, L. Lins, J. P. Ahrens, J. Freire, C. T. Silva, VISMASHUP: Streamlining the creation of custom visualization applications, IEEE Trans. Vis. Comput. Graph. 15 (6) (2009) 1539–1546.

[11] W. Benger, On safari in the file format jungle—why can't you visualize my data?, Comput. Sci. Eng. 11 (6) (2009) 98–102.

[12] H. Rzepa, The Chemical MIME Homepage, http://www.ch.ic.ac.uk/chemime/ [Accessed December 2011].

[13] G. Cameron, Modular visualization environments: Past, present and future, Comput. Graph. (ACM) 29 (2) (1995).

[14] J. P. R. B. Walton, NAG's IRIS Explorer, in: C. D. Hansen, C. R. Johnson (Eds.), The Visualization Handbook, Elsevier Butterworth Heinemann, 2005, pp. 633–654.

[15] W. Schroeder, K. Martin, B. Lorensen, The Visualization Toolkit: An object-oriented approach to 3D graphics, Kitware, Inc., 2006.

[16] D. Thompson, J. Braun, R. Ford, OpenDX: Paths to visualization, Visualization and Imagery Solutions, Inc., 2001.

[17] Advanced Visualization Systems, Visualization Concepts, Chapter 4: Importing Data, http://help.avs.com/Express/doc/help_722/books/vizcons/VizConceptsTOC.html [Accessed December 2011].

[18] J. Rasmussen, K. J. Vicente, Coping with human errors through system design: implications for ecological interface design, Int. J. Man. Mach. Stud. 31 (5) (1989) 517–534.

[19] J. Rasmussen, Skills, rules and knowledge: signals, signs and symbols, and other distinctions in human performance models, IEEE Trans. Syst. Man. Cybern. 13 (3) (1983) 257–267.

[20] K. J. Vicente, J. Rasmussen, Ecological interface design: theoretical foundations, IEEE Trans. Syst. Man. Cybern. 22 (4) (1992) 589–606.

[21] D. V. C. Reising, P. M. Sanderson, Ecological interface design for pasteurizer II: A process description of semantic mapping, Hum. Factors 44 (2) (2002) 222–247.

[22] J. J. Gibson, The Ecological Approach to Visual Perception, Houghton Mifflin, Boston, Mass., 1979.

[23] K. J. Vicente, Cognitive Work Analysis: toward safe, productive, and healthy computer-based work, Lawrence Erlbaum Associates, Mahwah, N.J., 1999.

[24] D. V. C. Reising, P. M. Sanderson, Work domain analysis and sensors II: Pasteurizer II case study, Int. J. Hum. Comput. Stud. 56 (6) (2002) 597–637.

[25] J. J. van Wijk, Image-based flow visualization, ACM Trans. Graph. 21 (3) (2002) 745–754.

[26] E. Zhang, J. Hays, G. Turk, Interactive tensor field design and visualization on surfaces, IEEE Trans. Vis. Comput. Graph. 13 (1) (2007) 94–107.

[27] DICOM, Digital Imaging and Communications in Medicine, http://medical.nema.org/ [Accessed December 2011].

[28] D. Wells, E. Greisen, R. Harten, FITS: A flexible image transport system, Astron. Astrophys. Suppl. Ser. 44 (1981) 367–370.

[29] USGS, The USGS DEM standards, http://data.geocomm.com/dem/ [Accessed December 2011].

[30] G. A. Jamieson, K. J. Vicente, Ecological interface design for petrochemical applications: supporting operator adaptation, continuous learning, and distributed, collaborative work, Comput. Chem. Eng. 25 (7-8) (2001) 1055–1074.

[31] N. Lau, G. A. Jamieson, Ecological interface design for the condenser subsystems of a boiling water reactor, in: 27th Canadian Nuclear Society Annual Conference, Canadian Nuclear Society, 2006.

[32] D. Trentesaux, N. Moray, C. Tahon, Integration of the human operator into responsive discrete production management systems, Eur. J. Oper. Res. 109 (2) (1998) 342–361.

[33] W. Xu, M. J. Dainoff, L. S. Mark, Facilitate complex search tasks in hypertext by externalizing functional properties of a work domain, Int. J. Hum. Comput. Interact. 11 (3) (1999) 201–229.

[34] C. M. Burns, J. Kuo, S. Ng, Ecological interface design: a new approach for visualizing network management, Comput. Netw. 43 (3) (2003) 369–388.

[35] P. A. Mendoza, A. Angelelli, A. Lindgren, Ecological interface design inspired human machine interface for advanced driver assistance systems, IET Intell. Transp. Syst. 5 (1) (2011) 53–59.