

Photi—A Fisheye View of Bubbles

N. I. Churcher

TR-COSC-15/93

The contents of this work reflect the views of the author(s) who is/are responsible for the facts and accuracy of the data presented. Responsibility for the application of the material to specific cases, however, lies with any user of the report and no responsibility in such cases will be attributed to the author(s) or to the University of Canterbury. Have a nice day.

Photi—A Fish-eye view of Bubbles

Neville Churcher*

Fisheye views have recently been employed in a number of application domains to enhance user benefits from graphically presented information. These techniques employ transformations which emphasize regions of interest while suppressing, but retaining, global detail. Applications to date have concentrated on the production of aesthetically pleasing views of essentially static objects such as digraphs. In this paper I investigate the extension of the concepts to interactive applications. The implementation of Photi, a diagramming tool for Smith's database schema design technique, is used as an example.

fisheye views, diagrams, information visualisation

INTRODUCTION

The management of complexity remains a major issue in many areas of computer science. Examples include software engineering, database design and network management.

It has long been recognised that diagrammatic techniques are an excellent means of highlighting important concepts and connections while suppressing details which are not relevant in the current context. The structured programming revolution of the 1970s, followed by the development of information engineering in the 1980s saw the development of techniques for representing concepts such as software architecture, control flow, data flow and data structure.¹

Computerised versions of many techniques emerged as CASE products became widespread. The use of multiple, independently scrollable, windows enabled “wall charts” to be replaced by system views more closely related to the task at hand. However, in many cases, the essential problems of complexity management have not been solved. In order to be truly useful, techniques must be capable of scaling up to “real” problems—sadly, many candidates are limited to text book examples.

The key to effective scaling is often an abstraction concept, such as “levelling” for DFDs, which allows fine detail to be suppressed thereby allowing the user to concentrate on features which are particularly “relevant” at the current level. One major advantage that diagramming tools have over their paper-based counterparts is the ability to recover such detail as required by “exploding” diagram components either *in situ* or in a separate window.

However, there are many diagramming techniques which apparently do not include natural analogues of the of the levelling process.

The fisheye views proposed by Furnas² represented an attempt to develop universally applicable concepts to help HCI workers determine how best to present information in order to combine an accurate and detailed representation of the point of interest (focus) and its “neighbourhood” while retaining important “landmarks” in the remainder. Further, it is necessary to be able to present different views as the focus of interest changes. These concepts are not limited to graphical applications. Furnas presents an example showing how the fisheye view concept might be applied to a structure-sensitive editor.

*Department of Computer Science, University of Canterbury, Private Bag 8004, Christchurch, New Zealand

A number of applications of fisheye views have been made. Examples include the 3D visualisation of a file system,³ the browsing of graphs⁴ and the visualisation of “linear” information structures⁵ and of maintenance data.⁶

The applications cited use fisheye views as a presentation technique for delivering views of an essentially static system. The motivation for this paper was to investigate the suitability of fisheye techniques in interactive environments where the system, as well as the view, is subject to change.

The dynamic nature of diagramming applications introduces a number of additional issues which must be addressed if fisheye views are to become generally applicable. Diagrammatic techniques have much in common and are generally based on the graph structures discussed in reference 4. Diagrams contain components (nodes), and connections (edges), each of which is represented by symbols and lines of the various types appropriate to each technique.

Smith’s method⁷ employs nodes which are “bubbles” represented by labelled ellipses, which may be nested or grouped, and connections represented by two types of arrows. This technique, which is unwieldy in its manual (i.e. pencil and paper) form, has been chosen to illustrate the significant advantages offered by applications using fisheye techniques.

Smith’s method is further described in the next section and the following section gives a brief summary of fisheye techniques. Next is a description of Photi, the Tcl/Tk application I have developed to investigate the application of fisheye concepts to a specific diagrammatic technique. This is followed by a discussion of some of the issues arising in the application of fisheye concepts to diagramming. Finally, some conclusions and directions for future work are presented.

SMITH’S METHOD

Smith’s method⁷ has been selected as a case study for the application of fisheye concepts to diagramming techniques. However, the results are applicable to a wide range of techniques in systems analysis and other areas. Diagrams are typically constructed from a similar set of components—symbols, connections and labels—although properties such as symbol shape, component colour, line and text style carry technique-specific information.

The method is a semi-formal synthesis procedure for constructing a database schema from a set of elementary facts. Although it has its weaknesses,^{8,9} Smith’s method is potentially suitable for small to medium sized problems and, particularly with software assistance, is capable of delivering good results while requiring little theoretical knowledge. Effective use of other techniques^{10,11} requires a greater formal background in areas such as data dependencies.

The method consists of three steps. First a dependency list is constructed to record the elementary facts. These include attribute names, single-valued dependencies (SVDs), multi-valued dependencies (MVDs) and composite attributes.

This information is then collated in the form of a bubble diagram. Figure 1 shows the university example described in reference 7.

Individual attributes are denoted by labelled bubbles and groups of attributes (composites) are enclosed by unlabelled bubbles. SVDs are represented by single-headed arrows, MVDs by double-headed arrows. The additional levels of bubbling around some attributes represents the independence of dependencies involving that attribute. For example, the double bubble around attribute **Student** corresponds to the two dependencies *each Student has a given Major and Year* and *Each Student in a given Class and Section has several Exam_Scores*.

Finally, a set of procedures is applied to derive the appropriate relation schemas. Figure 2 shows some of the relations corresponding to the diagram of figure 1. Prime attributes appear raised and non-prime attributes appear sunken for each relation.

Only a handful of references to Smith’s method are to be found in the literature. The method is sensitive to the choice of dependency list and, without computer assistance, becomes unmanageable for all but the most trivial systems. Constructs such as N:M or ternary relationships may not have unique or obvious representations in terms of textual descriptions of SVDs and MVDs. As has been argued elsewhere,⁹ it is better for the user to work at the level of the corresponding diagram and for the dependency lists to be maintained by software.

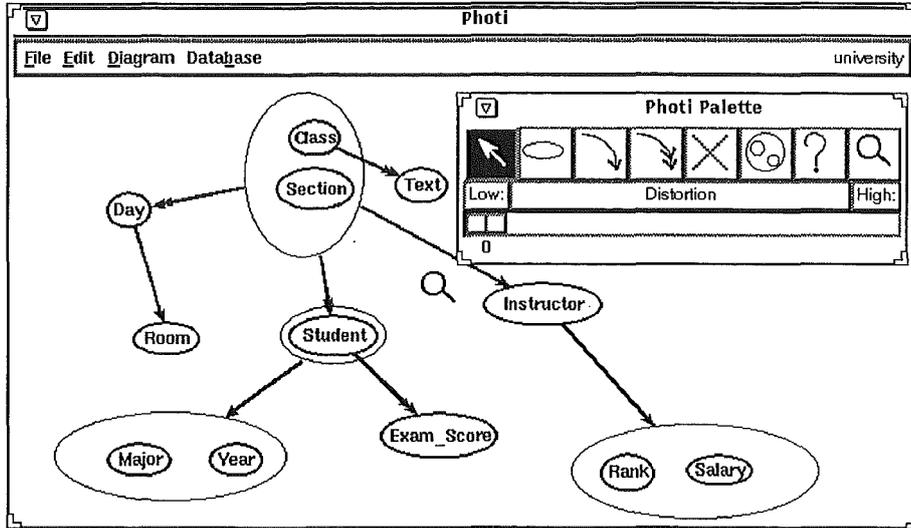


Figure 1: A dependency diagram

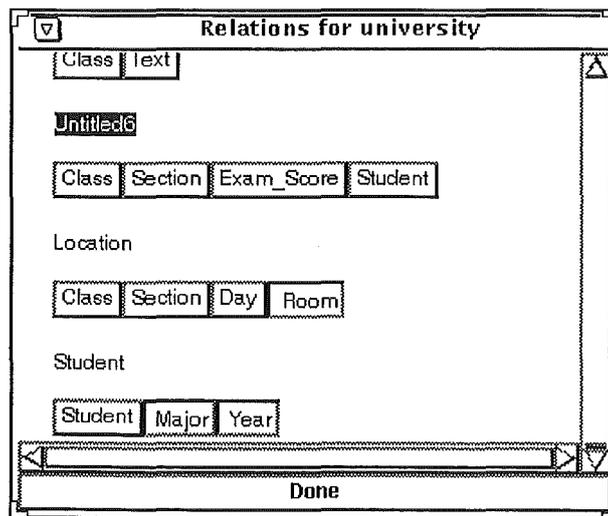


Figure 2: Relations for university example

FISHEYE VIEWS

Information overload is a potentially serious problem in most human-computer interactions. Computer-based information presentation techniques have many advantages—such as scrolling and multiple windows—over their paper-based counterparts. However, these alone have not proved sufficient to provide the power and flexibility required in order to visualise complex information effectively.

The fisheye view approach² is based on the view through a fisheye lens: the entire “world” remains visible; points near the focus are visible in great detail; the level of detail decreases with distance from the focus; some distortion is apparent, particularly at the periphery of the view. A further refinement combines these ideas with those used in the production of maps for tourists: the sizes of the symbols used for major landmarks are larger than those for comparable, but less “interesting”, features and much detail is omitted for clarity.

A key concept is the *degree of interest* function (DOI) which includes contributions from the *a priori* importance (API) of each component as well as contributions depending on its distance from the focus. Fisheye views are then constructed by displaying the components in decreasing DOI order until either all components are shown or a cutoff value is reached. The exact nature of the various functions involved depends on the application domain—a DOI suitable for binary trees will not be suitable for program listings or maps—and the success or otherwise of the application depends critically on an appropriate choice being made.

We consider a rectangular region containing a focal point (x_f, y_f) . It is convenient to work with normalised coordinates (\hat{x}, \hat{y}) whose magnitudes range from 0 at the focus to 1 at the boundary. Spatial transformations of the form

$$\hat{x}' \equiv \mathcal{G}(\hat{x}) \tag{1}$$

are then used to construct the fisheye view. The first derivative $\mathcal{G}'(\hat{x})$ gives the magnification, or scale factor.

The approach used by Sarkar and Brown⁴ refined Furnas’ basic ideas for application to graphs. The spatial transformation used was

$$\mathcal{G}(\hat{x}) \equiv \frac{(d+1)\hat{x}}{d\hat{x}+1} \tag{2}$$

where d is a constant known as the *distortion factor*. The same transformation is applied independently to the \hat{y} coordinates.

Similar transformations have been developed in other fields.¹² For example, the polyfocal transformation¹³ was derived for cartographic application and uses

$$\mathcal{G}(\hat{x}) \equiv \hat{x} + \frac{A\hat{x}}{1+C\hat{x}^2} \tag{3}$$

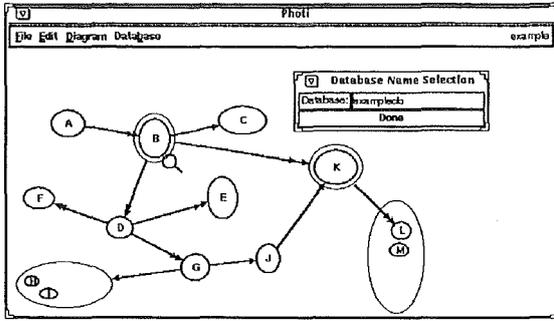
where A and C are constants.

For convenience, transformation functions are assumed to be continuous, monotonic and invertible over the range of \hat{x} under consideration.

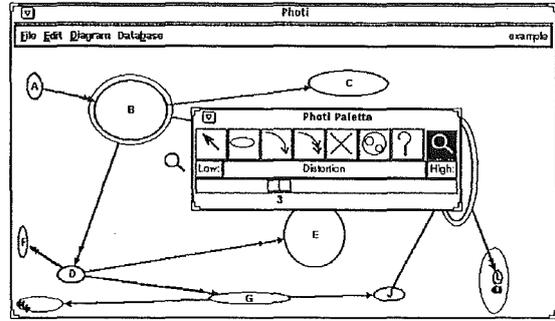
PHOTI

Photi is an implementation of Smith’s method. It was designed as both a platform for experimenting with the application of fisheye transformations and as a teaching tool.

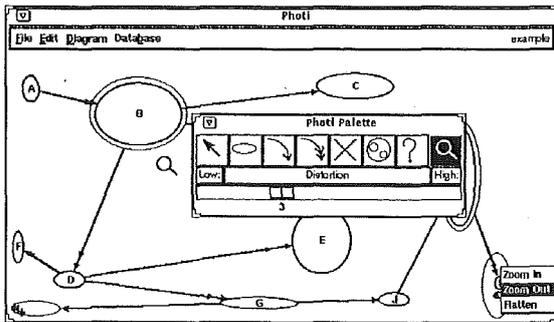
Photi’s palette contains a typical assortment of tools for (in left to right order in figure 1): moving and resizing bubbles, creating bubbles, creating SVDs and MVDs, deleting objects, grouping bubbles and examining object detail. The rightmost palette tool (represented by the magnifying glass icon) is used to move the focal point—whose position on the drawing area is marked with the same symbol—to a new location. The palette also contains a slider which is used to adjust the value of the distortion factor d in equation 2. A value of zero corresponds to the undistorted



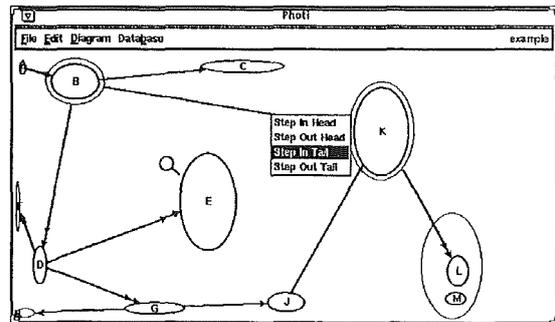
(a) Original "flat" form



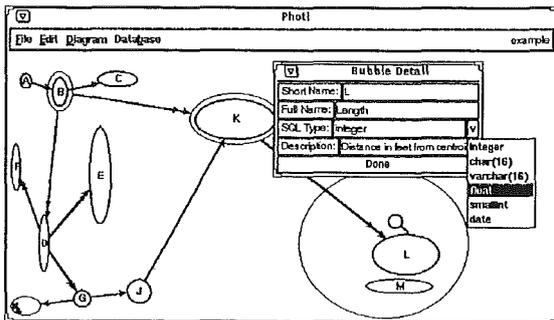
(b) Distorted view ($d = 3$), focus unchanged



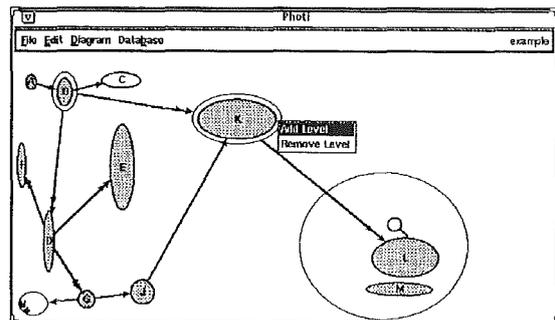
(c) Before zoom operation



(d) After zoom, focus unchanged, dependency management



(e) Focus moved, distortion unchanged



(f) Prime bubbles highlighted, level management

Figure 3: Example from reference 7—illustrating distortion and refocussing effects

or “flat” case. Adjustment of d or the position of the focal point triggers transformations which move and scale all objects according to the new focal point, using the current distortion value.

Figure 3 illustrates a number of Photi’s capabilities using an example diagram from Smith’s paper.⁷ The undistorted view is shown in figure 3(a) with the focal point located near the bubble labelled **B**. Leaving the focus unchanged while increasing the distortion produces the view of figure 3(b). Figures 3(c) and 3(d) show the effect of a zoom operation centred on the upper left corner of the popup menu. The bubble group containing **L** and **M** has been expanded while retaining the emphasis on the region surrounding the focal point. The zooming operations are described in a subsequent section on multiple foci. The effect of moving the focal point, while keeping the distortion constant, is shown in figure 3(e) while figure 3(f) shows the effect of highlighting prime bubbles as described elsewhere in connection with non-geometric transformations.

Perhaps the most fundamental of the operations on the diagram layout is the ability to alter the distortion while keeping the focus fixed. This is used to “set the tone” for further viewing. In addition, a number of further operations are activated via the magnifying glass tool:

- Reset the focus while keeping the distortion constant (see figures 3(b) and 3(e)), enabling the user to pan across the diagram examining areas of interest.
- Perform a “zoom” on the current—possibly already distorted—view, allowing several patches of detail to be examined simultaneously.
- Revert to “flat” configuration. This is equivalent to setting d to zero and discards the effects of prior transformations, including zooming.

In practice, these operations have proved sufficiently natural to enable new users to become proficient rapidly.

The “flat” coordinates of all new bubbles are stored and are updated appropriately when the bubbles are moved or resized. When a bubble is created in a distorted view, the inverse transformation is used to compute the corresponding “flat” coordinates.

Component movement introduces a slight complication. One could require that, as a bubble is dragged around using the arrow tool, its size should change as if it had been moved to its present location by a fisheye transformation with the current parameters. Another option would be to delay updating the size until the corresponding mouse-up event.

Repositioning a bubble in Photi does not trigger any resizing operations. It was felt unwise to distract the user by displaying a continuously changing component (as well as incurring the overhead of computing the changes!) as the potential gains are small. Attempting to anticipate the user’s reasons for moving a component seems equally unwise. No major functionality is lost since only one bubble is moved at a time and resizing operations are available with the same tool.

Photi is implemented using Tcl/Tk.¹⁴ Tcl is a powerful and flexible interpreted language while Tk provides a widget set for GUI development. If desired, the interpreter can be augmented with user procedures implemented in C rather than Tcl. In applications such as Photi, this is most appropriate for commands which are called repeatedly in response to mouse motion events. The computation of the crossing points of dependencies and bubbles, which is required frequently when bubbles are moved, has been implemented via a C procedure. Performance is satisfactory (on a Sun ELC a new view resulting from a distortion change is complete in less than 2 seconds for the diagrams shown in this paper) for small to medium sized systems. However, experimentation suggests that significant performance gains are possible as more commands are implemented in C.

Keyed lists are used to maintain details of bubbles, dependencies and relations. Diagrams may be saved in ASCII form for subsequent re-use. Alternatively, the relations may be exported to an Ingres database—providing and illustration of Tcl’s use to “glue” applications together.

DIAGRAMMING APPLICATIONS

The system described by Sarkar and Brown⁴ was directed towards providing fisheye views of read-only information. Diagramming tools have some additional requirements. In particular, it is

necessary to allow the user to create, modify or delete diagram components independently of the transformations used to produce a particular view. Above all, the user must be presented with a “natural” interface if fisheye techniques are to offer any improvement over current methods.

Layout

Another significant difference concerns the layout of the diagrams. The algorithms commonly used for laying out tree or graph structures were developed for applications such as circuit design and are concerned primarily with the adjacency matrix representation of the system. Thus, they concentrate on constructing a “good” layout in terms of constraints such as minimising connection crossings or total connection length. It is not always straightforward to express concepts such as which components “belong together” in terms of such constraints and it is appropriate to attach a high weighting to the component position suggestions available from the user. These include the positions at which components are created or to which they are explicitly moved.

In applications such as Photi, a point-and-click approach is used to create components so some user-specified position preference is available. While some “cleaning-up” may be acceptable—perhaps to snap components to a grid—dramatic changes of component positions are confusing and counterproductive. A single, apparently minor, change—such as deletion of a component—can potentially produce an enormous change in the “best” layout.

Fisheye techniques do not suffer from this problem to the same extent. Users retain the ability to perform localised layout changes in regions of interest while scaled views diminish the adverse impact of features such as line crossings in the remainder of the diagram. Clearly, it is desirable to combine the advantages of constraint based layout algorithms with fisheye views. In practice, this might best be achieved by applying such algorithms to undistorted views and employing transformation functions which preserve desirable layout properties. For example, the transformations used in Photi will not introduce overlaps between components.

Non-geometric transformations

The transformations involved in the generation of a new fisheye view are of two fundamental types. Geometric transformations describe the repositioning of objects. Equations 1–2 give those used in the present work. Other transformations involve the re-computation of size and detail for each object, given its new location.

The non-geometric transformations re-size each item according to a scaling function which should satisfy any restrictions appropriate for the diagrams in question, such as the requirement that the scaled components should not overlap.

In the case of diagrammatic techniques there may be both continuous and discrete components. Component labelling is an example of the latter category: as the component size decreases (continuously), the size of the font used for labels is reduced to the next available size, typically by multiples of some discrete unit such as point size. Eventually, the component may become “too small” and its label may be omitted. Information about the colour, shape and border thickness of components is also likely to be treated with the use of cut-offs.

The computation of visual worth for a component involve factors such as its importance (API), size in “flat” co-ordinates, amount of detail (and its granularity) which can be shown meaningfully (cutoff points) and position (after spatial transformation).

For Smith’s method, it would seem reasonable to base the a bubble’s importance on its *complexity*, involving the number of levels, group structure and numbers of SVDs and MVDs. Another scheme might assign higher API values to prime attributes than to non-prime attributes. Photi treats all bubbles as equal. Part of the justification for this is the fact that it is not known at bubble creation time whether a bubble will remain prime, or how many levels and dependencies it will have—thus establishing a true *a priori* importance is not realistic.

A more appropriate technique for diagramming applications would be to provide *a posteriori* importance facilities. In the case of Smith’s method, these might include highlighting prime bubbles or bubble chains corresponding to individual relations. Figure 3(f) shows prime bubbles emphasized by shading only—further emphasis could be added by boosting their sizes.

The specific form presented in reference 4 avoids component overlap, includes contributions from the API and has thresholds for detail and inclusion. However, this introduces additional interface complexity—the user controls the view by adjusting five separate parameters.

One major advantage of fisheye views is immediacy—if the user has to spend time “tweaking” the view then it is debatable whether any substantial improvement over the more common approach of scrolling (possibly multiple) windows has been achieved.

The computation of visual worth is likely to vary significantly between different diagrammatic techniques and a generally applicable system must be capable of handling a variety of styles. One must also realise that factors contributing to importance may be weighted differently depending on the individual user, task or system being examined. A major advantage of Tcl/Tk is the ease with which the parameters, and even the functional form, of the various transformation functions may be modified.

Photi displays all components, dispensing with cutoffs at the expense of cluttering the perimeter of the diagram with small objects, since the very presence of a dense region may trigger further investigation by the user. Cutoff schemes will depend on technique-specific factors such as symbol type or current rôle. For example, bubbles which are not determinants and have no levels might be safely suppressed if they become smaller than some cutoff size.

One must also consider the possibility of different scaling factors for the x and y directions leading to loss of component shape preservation.

Multiple foci

Applications of the single focus fisheye view indicate that it is well suited to situations where the access to the detail of a component and its near neighbours, in the context of a background of other components, is important. However, the nature of diagrams is such that users frequently wish to examine simultaneously several “patches” of detail. The fisheye counterpart to a split-screen view is the use of multiple foci. The transformations are applied independently, with the visual worth of a component being determined by superimposing their effects.

There are some potential drawbacks with this approach. In particular, spurious regions of high magnification may arise through the “interference” of the transformations. This effect depends on the form of the transformation (and hence magnification) function.¹² The polyfocal projection,¹³ whose transformation function is that of equation 3, avoids the problem because, unlike equation 2, the corresponding magnification function is non-monotonic.

Photi side-steps this inconvenience by providing a “multiple focus substitute” via the popup menu shown in figure 3(c). The zooming operations leave the focus unchanged but cause a further transformation to take place at half the current distortion—regarding the current, possibly already distorted, view as “effectively flat”—with its apparent focus at the current mouse position. The popup menu is used to distinguish clearly the user action for initiating this command from that for simply changing the focal point.

A more sophisticated approach might allow for different distortion parameters for the transformations about each focus. However, the additional demands this places on the user are likely to negate the possible benefits. In practice, the ability to flatten and refocus conveniently, combined with the zooming operations, has proved adequate.

CONCLUSIONS & FUTURE WORK

Smith’s method is an example of a technique which, in its manual form, is intractable for all but the simplest systems but which becomes remarkably attractive with appropriate software support. Experiments with Photi indicate that, with appropriate extensions, fisheye methods are indeed applicable to Smith’s and other diagramming techniques.

The construction or modification of diagrams is essentially a creative task. Serious frustration and potential errors may result from disruption of the user’s train of thought due to poor system performance. Consequently, acceptable response times are likely to be shorter than for static applications. Fortunately, most editing operations are incremental and only changes to the distortion

or focus require re-computation of the entire view. Implementing more of Photi's capabilities as C commands would improve performance. For very large systems some combination of scrolling and fisheye views may be appropriate.

An enhancement planned specifically for Photi, but applicable to other techniques, is the provision of "what-if" scenarios—enabling the user to see the potential consequences of a contemplated change—through the use of multiple drawing windows. More imaginative possibilities include working in periodic coordinates, perhaps wrapped on a sphere so that the diagram appears as if painted on a track ball.

Multiple focus fisheye views have potential applications in groupware systems where several users share a single diagram. The ability to draw attention to regions using relatively localised fisheye transformations offers more scope than the current alternatives such as multiple cursors.

Non-geometric transformations involve factors which may be subjective or method dependent. Variations are possible even in "static" systems. For example, the API for the states on the U.S. map display given by Sarkar and Brown,⁴ might be based on population, contribution to GNP or strength of national guard forces. A minimal "sensible" set of default behaviours should be provided and users must have the ability to customise these. This will be particularly important in the case of fisheye presentation management environments for CASE or IPSE applications where a variety of techniques must be handled in a consistent manner. Further investigation of the distinction between *a priori* and *a posteriori* importance contributions is also required.

Tcl is a suitable language for prototyping and implementation. Its advantages include ease of integration with other applications (the Ingres DBMS in Photi's case) and the ability to allow arbitrarily complex interaction with the user via the direct or indirect entry of new scripts.

ACKNOWLEDGEMENTS

I am grateful to Andy Cockburn for arousing my interest in fisheye transformations, and to the many contributors to `comp.lang.tcl` whose ideas I have borrowed.

REFERENCES

- [1] J. Martin and C. McClure. *Diagramming Techniques for Analysts and Programmers*. Prentice-Hall, 1985.
- [2] G.W. Furnas. Generalised fisheye views. In *Proc ACM SIGCHI '86 Conference on Human Factors in Computing Systems*, pages 16–23, 1986.
- [3] G.G. Robertson, J.D. Mackinley, and S.K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 189–194, New Orleans, Louisiana, April 1991.
- [4] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Proc. ACM SIGCHI '92*, pages 83–91, May 1992.
- [5] J.D. Mackinlay, G.G. Robertson, and S.K. Card. The perspective wall: Detail and context smoothly integrated. In *Proc ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 173–179, New Orleans, Louisiana, April 1991.
- [6] D. Mitta and D. Gunning. Simplifying graphics-based data: applying the fisheye lens viewing strategy. *Behaviour & information technology*, 12(1):1–16, 1993.
- [7] H. C. Smith. Composing fully normalised tables from a rigorous data dependency diagram. *Communications of the ACM*, 28(8):826–838, August 1985.
- [8] S. Neil. Survey of a database design technique and guide to database design methods. Project Report, Department of Computer Science, University of Canterbury, 1988.

- [9] R. P. Wilson. DB Drafter. Project Report, Department of Computer Science, University of Canterbury, 1989.
- [10] W. Kent. Fact-based data analysis and design. *Journal of Systems and Software*, 4:99–121, 1984.
- [11] P. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems*, 1(4):277–298, 1976.
- [12] Y. K. Leung and M. D. Apperley. A unified theory of distortion-oriented presentation techniques. Mathematical and information sciences report A93/1, Massey University, 1993.
- [13] N. Kadmon and E. Shlomi. A polyfocal projection for statistical surfaces. *The Cartographic Journal*, 15(1):36–41, 1978.
- [14] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1993. In Press : ISBN 0-201-63337-X.