

Fisheye Interfaces—Research Problems and Practical Challenges

Mikkel Rønne Jakobsen and Kasper Hornbæk

Department of Computer Science, University of Copenhagen
Njalsgade 128, Building 24, 5. floor
DK-2300 Copenhagen S, Denmark
`{mikkelrj,kash}@diku.dk`

Abstract. Fisheye interfaces give access to a large information structure by providing users with both local detail and global context. Despite decades of research in fisheye interfaces, their design and use are not well understood. To foster a discussion of fisheye views and their theoretical foundations, we identify five challenging areas in designing fisheye interfaces.

Key words: Information visualization, fisheye interfaces, focus+context

1 Introduction

A problem in many computer programs is that users can view only a small part of a large information structure. To make sense of a particular part, users may need to see that part in the context of the larger structure. Also, users may need to see details in parts of the structure that are not closely located. However, it is physically demanding to navigate an information structure in order to bring different parts into view (e.g., by scrolling) and cognitively demanding to assimilate the details that are viewed one at a time.

One user interface approach to addressing this problem is the fisheye view [12]. The fisheye view provides users with both local details and global context in a single view. According to the general formulation by Furnas [12], such a view can be generated by showing only those parts of the information structure that have a high degree of interest given the user’s current focus.

Despite decades of research in fisheye interfaces (see [11] for a review), their design and use are not well understood. In his 2006 follow up paper on fisheye views, Furnas [15] noted that it is not clear what exactly we mean by focus and context and that essential questions about what information fisheye views should provide to users remain unanswered. Further, Lam and Munzner [27] remarked about focus+context interfaces that “we do not know when, how, or even if they are useful.” Little advice is found in the literature on how to design fisheye interfaces. Designers are challenged with quantifying degree of interest and deciding how to distort the view without disrupting the user’s work. Possibly because of these challenges, fisheye techniques are largely absent in

widespread user interfaces. In short, fundamental research problems persist and fisheye interfaces remain very challenging to design.

In this paper, we describe these research problems and practical challenges concerning the design of fisheye interfaces. We hope to foster a discussion of fisheye views and their underlying concepts. One aim is to get away from point designs towards strong-hypothesis experiments [34]. Also, a long-term goal is the development of guidelines for the design of fisheye interfaces.

We first describe the background and motivation for fisheye interfaces and describe their design based on the current state of research so as to facilitate a discussion. Next, we describe our work on fisheye interfaces in programming in which the problems and issues that we discuss emerged. Last, we discuss the research problems and practical issues centered around five areas in the design of fisheye interfaces.

2 Related Work

Fisheye interfaces are one of several techniques for addressing the problem of working with information structures too large to fit within the display. Other techniques address the problem by providing information in multiple views separated either spatially (overview+detail) or temporally (zooming interfaces). By showing the parts simultaneously in a single view, fisheye interfaces aim to decrease the strain on memory associated with assimilating distinct views of the information structure, and thus potentially improve the user's ability to comprehend and manipulate the information structure [11].

Furnas [13] gave three motivations for balancing local detail and global context. First, local detail is needed for local interactions with a structure (e.g., editing a paragraph in a document). Second, global context is needed to interpret local detail. Third, global context is needed for navigation: “to tell the user what other parts of the structure exist and where they are”.

Whereas Furnas [12] has mainly been concerned with what to show in the view, other research has focused on how to distort the view so as to seamlessly integrate focus and context. An early use of distortion is the Bifocal Display, in which a central ‘close-up’ region, showing items in full detail, is surrounded by ‘demagnified’ regions on either side showing all other items in less detail [37]. Similar distortion-oriented views were examined by researchers at Xerox PARC in the Perspective Wall [30] and the Document Lens [36]. Interest in fisheye views has not waned in recent years; in fact, explorations of fisheye views applied to web browsers [3], calendars [6], and programming environments [21, 23] have been accompanied by empirical evaluations.

Below we describe fisheye interfaces and design aims mentioned in the literature. In describing fisheye interfaces, we distinguish between the *selection* of information, based on the degree of interest formalism originally proposed by Furnas, and the *presentation* of information. Earlier research has made a similar distinction [15, 24, 32].

2.1 Selection of Information

Based on the idea that users are not equally interested in all parts of an information structure, the degree-of-interest (DOI) function was suggested by Furnas for selecting information that should be shown in the fisheye view [12]. Furnas decomposed DOI into an *a priori* component—the global structural, intrinsic importance independent of the user’s current interaction—and an *a posteriori* component—the interest specific to the user’s current interaction with the information structure. The DOI of some element x , given the current focus point \cdot , can be formally defined as

$$DOI(x|\cdot) = F(API(x), D(\cdot, x)), \quad (1)$$

where $API(x)$ is the a priori importance of x , and D is the distance of x from the current focus. Using an additive function, a point’s degree of interest thus increases with its a priori importance and decreases with its distance to the user’s focus:

$$DOI(x|\cdot) = API(x) - D(\cdot, x) \quad (2)$$

A Priori Importance. Elements in an information structure may be intrinsically more important than others, which Furnas [12] suggested may be particular to the global structure: “a notion of high versus low resolution, or degree of detail, or grossness of a feature, generality, etc.” For instance, the heading of a document section may give a more general indication of the section’s content than the paragraphs in the section, the first level headings are of less detail than second level headings, etc.—the outline view of Microsoft Word thus uses the global structure of a document to allow sections of the document to be hidden. To calculate the DOI of an element, some measure of a priori importance must be defined. As an example, for a tree structure with root node r , API of a node x can be formally defined as

$$API(x) = -d(r, x), \quad (3)$$

where d is the distance measured as the number of edges between the node x and the root node r in the tree.

Distance To Focus. The second component of DOI is specific to the user’s current interaction with an information structure—some elements may be closer to the user’s focus and thus more important than others. Distance can be defined in different ways: for example, in a city map the distance might be defined geometrically (measured in meters) or as travel time (measured in minutes) either by walking, driving, or using public transit. For a tree structure, the distance $D(\cdot, x)$ can be defined as the number of edges between the node x and the node \cdot (the focus point).

2.2 Presentation of Information

Several approaches have been investigated for the presentation of information in fisheye views.

Filtering. The fisheye view suggested by Furnas [12] used filtering to show only elements with a degree of interest above a given threshold c :

$$DOI(x|\cdot) > c \quad (4)$$

Distortion. Distortion-oriented techniques can be used to demagnify less interesting parts of an information structure to balance detail and context. Distortion can formally be described with a magnification function that determines how parts of the visual structure are magnified in the distorted view [29]. Fig. 1 shows the magnification functions for two fisheye views. The x axis represents one dimension of a visual structure (e.g., rows in a table). The y axis represents the degree of magnification of the visual structure in the given dimension. Fig. 1(a) shows a fisheye view where parts of the visual structure are entirely hidden (with magnification factor of 0), while Fig. 1(b) diminishes parts to a readable size with a magnification factor d_r . A magnification factor d_{ur} that diminishes parts to an unreadable size is shown with a dashed line in the figure.

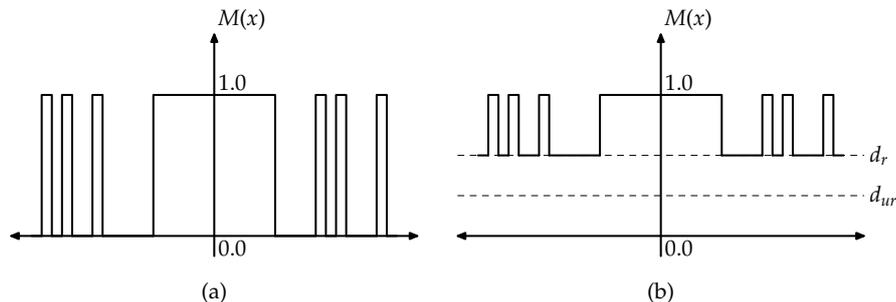


Fig. 1. Magnification functions for two fisheye views using discrete levels of magnification. (a) Less interesting parts of the visual structure are hidden by using a magnification factor of 0. (b) Less interesting parts are diminished by magnification factor d_r .

Some distortion-based techniques use non-continuous magnification (as in Fig. 1) where different parts of the information structure are magnified at discrete levels [10, 19]. For example, Cockburn and Smith [10] compared source code views that demagnified blocks of code to levels that were ‘just legible’ and ‘illegible’. Users preferred the ‘just legible’ magnification level, but performed better with the ‘illegible’ magnification level.

Other distortion-based techniques demagnify the content of dedicated context regions while leaving the content in a focal region undistorted [21, 30, 37]. For instance, the Perspective Wall [30] contains a center panel that shows an undistorted detail view with perspective context panels on either side that shows a demagnified view.

Still other distortion-based techniques use continuous magnification functions to the effect of a photographic fisheye lens by geometrically distorting the view of graphs or maps. For example, The Hyperbolic Browser [28] uses a hyperbolic geometry to uniformly embed an exponentially growing graph structure. Carpendale and Montagnese [9] have described the many possibilities for presentations provided by different types of lenses and distance metrics.

Semantic Zoom and Aggregate Representations. Demagnification can make an element unreadable at smaller sizes. With semantic zooming, popularized by Bederson et al. [4], alternative representations of an element are used at different sizes. A variant of semantic zooming, useful to reduce clutter in the view, is to show the individual elements in a group as one aggregate representation. For instance, DOI Trees [8] represent nodes that are distant from the focus point in aggregate form.

Resolution. Resolution can be used to filter information in the view without the use of elision or distortion [15]. For instance, Baudisch et al. [2] explored a focus+context approach in which a focus region in the center is shown in high resolution and the surrounding context region is shown in low resolution.

2.3 Design Aims

Specific goals in the design of fisheye views are discussed in the literature. One goal relates to one of Furnas' [13] motivations for fisheye views mentioned earlier, that is, to help the user navigate effectively in the information structure. Furnas [14] described two characteristics of effective navigation: First, the user must be able to traverse the structure in a small number of steps. Fisheye views can provide shorter paths to distant parts of the structure by including links to remote elements. Second, visual cues are needed to enable visual search for links in the view that lead closer to the information needed for the user's task. Pirolli et al. [33] suggested that strong information scent improves visual search, whereas crowding of targets in the context region of focus+context displays degrades visual search.

Munzner et al. [31] introduced the idea of 'guaranteed visibility' where highlighted areas of a structure that are important must remain visually apparent at all times. They find that guaranteed visibility relieves users from exhaustive exploration in identifying and comparing differences in large trees.

Zellweger et al. [38] considered different types of information about unseen objects that support different uses in City Lights: information about the existence of objects for *awareness*, physical or informational properties for *iden-*

tification, positional information for *navigation*, and abstract information for *interaction*.

In all, the appropriateness of different techniques for selecting and presenting information in a fisheye view depends on the user’s tasks and how the fisheye view aims to support those tasks.

3 Fisheye Interfaces in Programming

The concerns we discuss in this paper emerged in the design and evaluation of fisheye interfaces that aim to support programming [21, 23]. With the specific goal of helping programmers navigate and understand source code, we have integrated a fisheye view in the Java editor in Eclipse, an open source development platform. Basically, the fisheye view works by assigning a degree of interest (DOI) to each program line based on its a priori importance and its relation to the user’s current focus in the file. Then, lines with a DOI below a certain threshold are diminished or hidden, resulting in a view that contains both details and context.

Below, we discuss the fisheye interface design used in an initial controlled experiment [21], and the design used in a later field study [23], arguing for the changes made to the initial design.

3.1 The Fisheye Interface Initially Studied

In the fisheye interface we studied initially (shown in Fig. 2), the source code view is divided into a focus area and a context area. The editable part of the window, the focus area, is reduced to make space for a context area. The context area uses a fixed amount of space above and below the focus area. Parts of the source code that are less relevant given the focus point are diminished or hidden. The focus point is defined as all lines visible in the focus area. As a result, the context area is updated when the user scrolls the view, but remains unchanged when the user moves the caret within the bounds of the focus area.

A DOI function determines if and how much the lines are diminished in the context area. The API of a line is defined by its type (e.g., an indenting statement or a variable declaration) and indentation level. The distance component comprises both a syntactic distance measure similar to the tree distance described in [12] and a semantic distance measure. Semantic distance causes lines containing declarations of symbols that are referenced in the focus point to be more relevant than other lines, including syntactically close lines.

The method used for presenting program lines in the context area is to demagnify lines in descending order of DOI. A line’s magnification level is determined by its DOI relative to the amount of lines yet to be allocated space in the context area. This approach aims to reduce the size of the least interesting lines while showing mainly readable lines in the context area.

interaction. One motivation for this change was to allow the user to directly specify a semantic relation with the caret—we had learned that it was not always clear to users of the first fisheye interface what semantic relation that caused program lines to appear in the context area.

Second, we changed the presentation so that lines are always included in the context area if they have a degree of interest above a given threshold. All lines cannot be shown simultaneously in the fixed amount of space of the context area. However, instead of demagnifying lines prioritized by their DOI, which may result in some lines becoming unreadable or hidden, the context area can be scrolled. The motivation for this change is that all the lines may be important to the user. This design thus aims to guarantee users that the context area contains all the lines they expect to find (e.g., all the occurrences of a variable the user has selected).

```

12 package org.eclipse.nebula.widgets.gallery;
37 public class DefaultGalleryItemRenderer extends Abs
41     boolean dropShadows = false;
78     public void draw(GC gc, GalleryItem item, int i
95         if (itemImage != null) {
100             size = getBestSize(imageWidth, imageHeigh
101
102             xShift = (width - size.x) >> 1;
103             yShift = (useableHeight - size.y) >> 1;
104
105             if (dropShadows) {
106                 Color c = null;
107                 for (int i = this.dropShadowsSize - 1
108                     c = (Color) dropShadowsColors.get
109                     gc.setForeground(c);
110
111                 gc.drawLine(x + width + i - xShif
112                 gc.drawLine(y + yShift + dropShad
113             }
114         }
115     }
152 }
154     public void setDropShadowsSize(int dropShadowsS

```

Fig. 3. The fisheye interface evolved for use in a field study [23]. Less interesting lines are hidden in the context area by using a magnification factor of 0. However, all lines with a degree of interest above a given threshold are included in the context area. In the example shown here, the bottom context area contains more lines than can be shown simultaneously. The context can be scrolled to view lines that are not initially shown.

3.3 Findings From User Studies

Overall, the results from our studies attest to the usefulness of fisheye interfaces to programmers. Participants in a controlled experiment preferred the fisheye in-

terface to a linear source code interface [21]. Participants in a field study adopted and used the fisheye interface regularly and across different activities in their own work for several weeks [23]. The fisheye interface does not seem useful in all tasks and activities, however. Participants in the experiment completed tasks significantly faster using the fisheye interface, a difference of 10% in average completion time, but differences were only found for some task types. Although the results indicate usability issues, they also suggest that some tasks were less well supported by the fisheye interface. In addition, data from the field study showed periods where programmers did not use the fisheye interface, and debugging and writing new code were mentioned as activities for which the fisheye interface was not useful.

Specifically, the fisheye view enables programmers to perform some tasks with less physical effort compared with a normal linear view of source code. This was found in the controlled experiment by analyzing in detail how participants' performed tasks with a fisheye view compared with a linear view. Using a fisheye view, participants directly used information in the context area or navigated with sparse interaction; they read program lines in the context area or clicked in the context area to jump to a particular line. We saw this behavior also in observations of programmers using a fisheye interface in real-life work.

Consistently across our studies, we found that lines semantically related to the user's focus were the most important—such lines were the most frequently used and were the most often mentioned by participants as a benefit of the fisheye interface.

4 Research Problems and Practical Challenges

Several concerns came out of the studies of fisheye interfaces in programming that were described in the previous section. We discuss five areas that present both fundamental problems to research and practical challenges in the design of fisheye interfaces.

4.1 Is a priori importance useful and if so, what for?

In an information structure, some information may be inherently important regardless of what the user is focusing on. For instance, significant landmarks in a map can provide useful context for navigation. However, the usefulness of showing a priori important information in a fisheye interface is not well understood. Further, the reasons why information is deemed a priori important may be confusing to users or may not be relevant in all contexts.

Evidence of the usefulness of showing a priori important information in fish-eye interfaces is lacking. In our fisheye interface, lines are intrinsically interesting if they contain higher-level information, such as method declarations. Because lines that are directly related to the focus do not always fill the context area, lines that contain nearby method declarations, which are less directly related to

the user's focus, are often shown (see Fig. 2). However, we found no clear indication that these lines improve performance in navigation and understanding tasks [21]. In real life, programmers did make use of lines containing nearby method declarations, but they seem less useful than lines directly related to the user's focus [23]. Also, Hornbæk and Hertzum [20] compared alternative fisheye menu interfaces, including one in which a priori important menu items outside of the focus region are shown at larger font sizes. Analysis of eye gaze fixations showed that the context region was used more than a fisheye menu interface without a priori important parts, but no performance differences were found. The authors suggest that the context region might have been used more because it contained more readable information. These findings indicate a use of a priori important information in fisheye interfaces. But neither these evaluations of fisheye interfaces, nor other studies we can think of, are able to link performance benefits to a priori importance.

There are practical challenges concerning a priori importance as well. One challenge is that users may become confused if information is important for no clear reason. The reason that section headings in a document are intrinsically important may be clear to users (i.e., they give a general indication of the content of sections). In other cases, the reason why information is deemed important may be less clear. For instance, in a fisheye view of electronic documents, Hornbæk and Frøkjær [19] assigned higher a priori importance to the first and last paragraphs of document sections, which have been found to give better indication of the content of sections. Participants in their study spent less time on the parts of less a priori importance, but they lacked trust in the algorithm. Another practical challenge is that information may be a priori important for various reasons, but determining why it is useful to show in a fisheye view is difficult. A priori importance need not be given by intrinsic or structural properties such as level of detail. In programming, for instance, methods that are invoked many places in the program (a static indicator of importance) or methods that are frequently called (a dynamic indicator of importance) may be more important a priori. Also, dynamic data about the user's previous navigation patterns [18] or user-community popularity data [15] could indicate importance of information that could potentially make it useful to display. However, information that is important for one particular reason may not be relevant in all contexts of use of a particular interface.

In conclusion, research has yet to uncover how a priori importance contributes to the usefulness of showing information in fisheye interfaces. Also, as information may be important for many reasons, designers are challenged with determining a priori importance with respect to varied contexts of use and making clear to users why information is deemed important.

4.2 What does the user focus on?

Although focus is central to fisheye interfaces, very different notions of focus are used in the literature. We see issues concerning (1) how directly and predictably

focus changes result in view changes, (2) how directly the user controls the focus, and (3) how the user’s focus may depend on task or context.

First, fisheye interfaces that automatically balance focus and context depending on the user’s focus can be distinguished from interfaces in which users manually balance focus and context. In Table Lens [35], for instance, the user has full control over which table cells are shown in detail by interacting directly with the cells using the mouse. The effect of the user’s interaction on the presentation is relatively predictable. In fisheye interfaces that automatically change the view based on a DOI function, as formalized by Furnas [12], the effect of the user’s interaction on the presentation is less direct: it depends on how the focus point is given by the user’s interaction, how the distance component in the DOI function is defined, and how the presentation changes based on the DOI. As described in section 3.2, our fisheye design evolved to use a composite focus based on two forms of interaction: (1) a focus area spanning a range of program lines is used to determine which enclosing program structures that form the context to the code in focus; (2) the text caret gives the user explicit control for focusing on a specific variable, thus contributing as context those lines that contain occurrences of the variable. Consequently, the user can expect different types of information to change in the context area when moving the focus area (e.g., by scrolling) and when placing the caret in an element. We think this helps users control the focus and predict what information is shown in the view.

Second, very different mechanisms have been investigated in the literature for controlling the focus. Although a mouse or pointing device is probably the most frequently used for determining the user’s focus [5, 6, 10, 16, 17, 28], mechanisms range from tracking of eye gaze [1] to registering where the text caret dwells for a period of time [25]. It is unclear how different mechanisms compare (e.g., in directness or accuracy) in controlling the focus point in fisheye interfaces.

Third, a user’s task and information needs may change during use of a fisheye interface. But information related to the user’s focus may not be useful across tasks. Thinking of a user’s focus in other terms than the DOI component, Mylar [25] builds a task context from multiple focus points by assigning DOI to elements in the view in which the text caret dwells for a period of time. Janecek and Pu [24] propose a DOI function composed of one or more weighted distance functions that the user can configure using sliders. However, the required effort of users in configuring the DOI function to meet changing information needs seems counter to the idea of fisheye interfaces that automatically change depending on the user’s focus.

In conclusion, the user’s focus—a central component in fisheye interfaces—is relatively indefinite and important problems are related to its use.

4.3 What interesting information should be displayed?

A degree of interest function may result in varying amounts of interesting information as the user’s focus changes. However, it is not trivial to visually present varying amounts of information in a view of a given size.

A specific instance of this issue in fisheye views of tree structures is the sibling overload problem where a tree node has many children all with the same DOI [26]. In answer to the problem, Koike [26] proposes fractal views in which the offspring of the nodes with lots of children will be the first to disappear. Fractal views may thus often, but not always, keep the information shown in the view constant. However, this approach may not be useful in all tasks and might confuse users about what information is left out of the view. In our study of fisheye interfaces, we found that programmers read lines directly in the context area, for instance to find a particular use of a variable. However, a large number of lines may contain occurrences of the variable that the user focuses on and not all can be shown simultaneously in a readable size. Using magnification would cause at least some of the occurrences to be unreadable. Our fisheye design thus allows users to scroll in the view of context information to access more lines than can be shown simultaneously. This design aims for predictability in that users can expect to access all information related to their focus in the context area—a scrollbar gives indication of the amount of lines in the context area. However, the scrollable context area adds complexity to the interface and does not strictly guarantee visibility of all related lines.

The issue of representing a varying amount of information may be further complicated when the fisheye interface balances different types of information that are useful in different tasks. If combined in one fisheye interface, visual representations of different types of information may conflict. Instead, a transient visualization, called up temporarily close to the user’s focus, might show a representation of only the information needed to support a specific task without changing the permanent interface [22].

In conclusion, presenting varying amounts of information in a view of fixed size may be a problem if the information structure does not scale meaningfully to different sizes.

4.4 Do fisheye views integrate or disintegrate?

Fisheye interfaces distort the visual representation of information. When used in combination with applications that support rich interaction with the visual representation, distortion may interfere with those interactions. Research into the problems of integrating fisheye interfaces in real applications is, however, limited. With rare exceptions [6, 23], fisheye interfaces explored in the literature are mainly designed as standalone systems. Table Lens [35], for example, demonstrates an innovative fisheye technique that facilitates exploratory analysis of large data tables. However, it is not clear how the functionality in Table Lens could be combined with the rich set of features in modern spreadsheet applications.

In our work on integrating a fisheye view in a source code editor, we tried to take into account the diversity of editor use in real-life programming work. In the fisheye interface that we initially studied, we used a static division of focus and context in a source code editor for at least two reasons. It provides stability in the context area, which changes only when scrolling the view, but remains stable

when moving the caret within the focus area. Moreover, it allows users to interact with the editor as they normally do, including scrolling, searching, and editing. Even so, user study results suggest that the fisheye interface does not support all activities and that some users prefer a plain source code view in those situations. In the fisheye interface we deployed for real-life use, we thus provided an option for switching the fisheye interface on or off and a shortcut for transiently calling up the fisheye interface [23]. Other research support this approach: based on a study of web browser interfaces, Baudisch et al. [3] recommend allowing user's to call up a fisheye interface on demand for specific uses.

Although empirical insight in the adoption of fisheye interfaces in real-life applications is thus far limited, one lesson is that use of the visualization to improve performance for some tasks should not detract from performance for other tasks [6].

4.5 Are fisheye views suitable for large displays?

A key motivation for using fisheye views is that many information structures are large, while the windows for viewing those structures are small [12, 29]. With small windows, fisheye views can help by providing surrounding context to the local details currently looked at. However, it is not clear how the usefulness of fisheye views relates to window size. First, while small displays have motivated some uses of fisheye interfaces [6, 7], small displays may detract from the usability of other fisheye views. In our study of a fisheye view of source code we learned that users' prefer a large view of local details, at least for some tasks, and that a larger display can mitigate the problems that the fisheye view causes. Work context is probably a key factor, but still we are curious: do large displays generally allow for fisheye views that are preferable and more usable compared to other techniques? Second, with wall-sized displays our view of a large information structure is limited mainly by the human visual system. Fisheye views may also in this context provide balance between contextual overview and local detail, but research in this area is lacking. With the rare exception of a study of focus+context screens [2], we are unaware of research in the use of fisheye techniques on large displays. In all, a better understanding of the effect of display size on design and usability of fisheye interfaces is needed.

5 Conclusion

We have discussed research problems and practical issues in the design of fisheye interfaces based on the degree of interest formalism of Furnas [12]. The concerns emerged in our empirical work on fisheye views in programming and we have related the concerns to fisheye research in the literature.

Five areas of concern have been discussed. First, a priori importance is central to the DOI formalism proposed by Furnas, yet its contribution to the usefulness of fisheye interfaces is not well understood. We have contrasted the usefulness of generally important information to that of information directly related to

the user's focus. Second, concerning the user focus component of DOI we have discussed how the user controls the focus, the directness and predictability of the resulting view changes, and how the user's focus relates to the user's task. Third, we have called attention to the problem of determining what should be left out if not all important information fits in the view. Fourth, most fisheye techniques have been researched as standalone systems. We have discussed the challenges of integrating fisheye views in existing interfaces. Finally, we have questioned a key motivation for research in fisheye interfaces: that of viewing large information structures through small windows. However, large displays may potentially benefit from fisheye techniques as well. The effect of display size on the usefulness of fisheye interfaces needs investigation.

In sum, fisheye interfaces continue to present an intuitively attractive interface paradigm. However, as discussed in this paper several fundamental research questions and unsolved design challenges persist.

6 Acknowledgements

Many thanks to Miguel Nacenta and the anonymous reviewers for their constructive comments.

References

1. M. Ashmore, A. T. Duchowski, and G. Shoemaker. Efficient eye pointing with a fish-eye lens. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 203–210, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.
2. P. Baudisch, N. Good, V. Bellotti, and P. Schraedley. Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 259–266, New York, NY, USA, 2002. ACM.
3. P. Baudisch, B. Lee, and L. Hanna. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 133–140, New York, NY, USA, 2004. ACM Press.
4. B. B. Bederson and J. D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 17–26, New York, NY, USA, 1994. ACM.
5. B. B. Bederson. Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 217–225, New York, NY, USA, 2000. ACM.
6. B. B. Bederson, A. Clamage, M. P. Czerwinski, and G. G. Robertson. Datelens: A fisheye calendar interface for PDAs. *ACM Trans. Comput.-Hum. Interact.*, 11(1):90–119, 2004.
7. S. Björk, L. E. Holmquist, J. Redström, I. Bretan, R. Danielsson, J. Karlgren, and K. Franzén. West: a web browser for small terminals. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 187–196, New York, NY, USA, 1999. ACM.

8. S. K. Card and D. Nation. Degree-of-interest trees: a component of an attention-reactive user interface. In *AVI '02: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 231–245, New York, NY, USA, 2002. ACM.
9. M. S. T. Carpendale and C. Montagnese. A framework for unifying presentation space. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 61–70, New York, NY, USA, 2001. ACM Press.
10. A. Cockburn and M. Smith. Hidden messages: evaluating the efficiency of code elision in program navigation. *Interacting with Computers*, 15(3):387–407, 2003.
11. A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):1–31, 2008.
12. G. W. Furnas. The FISHEYE view: A new look at structured files. Technical Report #81-11221-9, Murray Hill, New Jersey 07974, U.S.A., 12 1981.
13. G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23, New York, NY, USA, 1986. ACM Press.
14. G. W. Furnas. Effective view navigation. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 367–374, New York, NY, USA, 1997. ACM Press.
15. G. W. Furnas. A fisheye follow-up: further reflections on focus + context. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 999–1008, New York, NY, USA, 2006. ACM.
16. C. Gutwin. Improving focus targeting in interactive fisheye views. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 267–274, New York, NY, USA, 2002. ACM Press.
17. J. Heer and S. K. Card. Efficient user interest estimation in fisheye views. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 836–837, New York, NY, USA, 2003. ACM Press.
18. W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 3–9, New York, NY, USA, 1992. ACM Press.
19. K. Hornbæk and E. Frøkjær. Reading patterns and usability in visualizations of electronic documents. *ACM Transactions on Computer-Human Interaction*, 10(2):119–149, 2003.
20. K. Hornbæk and M. Hertzum. Untangling the usability of fisheye menus. *ACM Trans. Comput.-Hum. Interact.*, 14(2):6, 2007.
21. M. R. Jakobsen and K. Hornbæk. Evaluating a fisheye view of source code. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 377–386, New York, NY, USA, 2006. ACM.
22. M. R. Jakobsen and K. Hornbæk. Transient visualizations. In *OZCHI '07: Proceedings of the 19th Australasian conference on Computer-Human Interaction*, pages 69–76, New York, NY, USA, 2007. ACM.
23. M. R. Jakobsen and K. Hornbæk. Fisheyes in the field: Using method triangulation to study the adoption and use of a source code visualization. In *CHI '09: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1579–1588, New York, NY, USA, 2009. ACM Press.
24. P. Janecek and P. Pu. A framework for designing fisheye views to support multiple semantic contexts. In *International Conference on Advanced Visual Interfaces (AVI02)*, pages 51–58, Trento, Italy, 2002. ACM Press.

25. M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM Press.
26. H. Koike. Fractal views: a fractal-based method for controlling information display. *ACM Trans. Inf. Syst.*, 13(3):305–323, 1995.
27. H. Lam and T. Munzner. Increasing the utility of quantitative empirical studies for meta-analysis. In *BELIV '08: Proceedings of the 2008 conference on BEyond time and errors*, pages 1–7, New York, NY, USA, 2008. ACM.
28. J. Lamping and R. Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7:33–55, 1996.
29. Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1:126–160, 1994.
30. J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 173–176, New York, NY, USA, 1991. ACM.
31. T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 453–462, New York, NY, USA, 2003. ACM.
32. E. G. Noik. A space of presentation emphasis techniques for visualizing graphs. In *Proc. Graphics Interface '94*, pages 225–233, 1994.
33. P. Pirolli, S. K. Card, and M. M. Van Der Wege. The effects of information scent on visual search in the hyperbolic tree browser. *ACM Trans. Comput.-Hum. Interact.*, 10(1):20–53, 2003.
34. J. R. Platt. Strong inference. *Science, New Series*, 146(3642):347–353, Oct. 16 1964.
35. R. Rao and S. K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 318–322, New York, NY, USA, 1994. ACM.
36. G. G. Robertson and J. D. Mackinlay. The document lens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 101–108, New York, NY, USA, 1993. ACM.
37. R. Spence and M. Apperley. Data base navigation: an office environment for the professional. *Behaviour and Information Technology*, 1(1):43–54, 1982.
38. P. T. Zellweger, J. D. Mackinlay, L. Good, M. Stefik, and P. Baudisch. City lights: contextual views in minimal space. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 838–839, New York, NY, USA, 2003. ACM.