General Rectangular FishEye Views for 2D Graphics

Uwe Rauschenbach, Stefan Jeschke, Heidrun Schumann
University of Rostock, Computer Science Department, D-18051 Rostock, GERMANY

Keywords. Mobile computers, Fisheye views, Focus-and-context techniques, Graphical displays.

Abstract. This paper describes the RECTANGULAR FISHEYE VIEW for the combined presentation of 2D raster images, 2D vector graphics and text. A detailed presentation of the area of highest user interest (the focus) is integrated into a lower-detail context display providing an overview. Screen space is saved by downscaling (distorting) the context. The geometric layout of the view is assembled using rectangular regions. Three different context modes are proposed to provide for computational scalability. Performance and quality implications of two different implementation options are compared.

1 Introduction and Related Work

Ultra-portable computers have become more powerful and popular in recent years, and applications developed for desktop computers are being ported to these small devices. Mobile computers suffer from limited screen space compared to stationary equipment, however. *Focus-and-context techniques* can be used to decrease the screen space demands of graphical and non-graphical information displays. Such techniques trade off the two conflicting user goals *detail* and *overview*.

In visualization research, *fisheye views* have been proposed to reduce the space demands of very large graphical layouts for presentation on workstation monitors. The nonlinear magnification homepage [1] contains many publications regarding this topic. Fisheye views combine a detailed presentation of the area of highest user interest (called the *focus*) with a downscaled and distorted *context* display which provides an overview. Such, they maintain a high degree of detail and fidelity in interesting areas, combined with display space savings for the context.

Most fisheye techniques (e.g., hyperbolic viewers [3] or nonlinear magnification [2]) use different distortion parameters for every display pixel. Since this requires high computation power, it may not be suitable for many resource-limited mobile devices. Here, techniques are preferred which use the same distortion over large, easily describable areas to save computing resources. The *rubber sheet* technique [6] meets these requirements dividing a graphical display into vertical and horizontal stripes, each of which may be assigned a different scaling factor.

In [4, 5], we have proposed a technique called the RECTANGULAR FISHEYE VIEW which is tailored to the presentation of large raster images on small screens combined with support for the wavelet-based transmission of these images over low-bandwidth links from a remote image server. This scheme propagates the screen space savings achieved by distorting the context to the transmission system, transferring only those image data which are required for display. The tight coupling with the transmission control imposes some restrictions. To support wavelets, scaling factors have to be powers of two which restricts the context definition and allowed zooming only in these steps. Furthermore, only raster images are supported.

This paper describes a generalisation of the

RECTANGULAR FISHEYE VIEW for the display of raster images, 2D vector graphics and text on small devices. By disregarding the transmission, we are able to provide smooth zooming and greater flexibility for the context definition. Three context definition modes with different quality and computational complexity will be distinguished and their advantages and disadvantages will be shown. We will briefly discuss interaction techniques and opportunities for performance optimisations for devices with limited computing power. Furthermore, we consider how to implement the context modes and compare their visual quality and the resulting response times.

The paper is structured as follows: After describing the basic idea of the RECTANGULAR FISHEYE VIEW in section 2, we are going to discuss the three context modes and their underlying geometry computations in section 3. The sections 4 and 5 deal with interactivity and implementation issues, respectively. Finally, section 6 discusses some results, and section 7 provides a summary.

2 The RECTANGULAR FISHEYE VIEW

Before we introduce the various context modes, we first describe the general layout of the RECTANGULAR FISHEYE VIEW.

The proposed scheme maps a graphical layout onto a *canvas window* in a space-saving manner. A *logical coordinate system* must be provided in which the graphical layout can be specified. In this coordinate space, a rectangular region called the *logical window* defines the part of the layout to be displayed. The forward mapping (denoted as *fisheye transformation*) is required for graphical output and converts logical coordinates into device coordinates. The respective inverse mapping is needed for converting device coordinates to logical coordinates during graphical interaction. Fig. 8 illustrates this.

Fisheye techniques display a defined part of the graphical layout (the *focus region*) at a high degree of detail. In the RECTANGULAR FISHEYE VIEW, a rectangular focus is used since this simple geometry provides a good orientation for the user and requires only a modest computation effort. All parts of the layout in the focus region are scaled uniformly. The scale factor (denoted as *zoom factor*) can be modified interactively by the user to control the tradeoff between overview and detail.

The display space in the canvas window not occupied by the focus is used to present the remaining parts of the graphical layout as context information. This is achieved by grouping one or more *context belts* around the focus. Each belt is composed of several *grid rectangles* which are downscaled during the fisheye transformation in order to save screen space. Fig. 1 illustrates the proposed belt scheme[1]. The scale factors of the context belts are chosen such that the resulting RECTANGULAR FISHEYE VIEW always fits exactly into the available space of the canvas window.

Sarkar and Brown [6] list desirable features of fisheye views. Regarding the integration of focus and context, they demand that the focus should be fitted smoothly into the context and that the level of detail in the context area should be a smooth function of distance from the focus.

To provide smoothness, the downscaling factor has to be different for every pixel in the canvas window. This requires a large computation effort which poses problems on low-resource mobile devices. Therefore, we propose three different *context modes* with varying computation demands in order to provide *computational scalability*. Such a context mode can be described by the number of context belts and the context scaling method used (see next section).

3 Geometry Computations

Before we discuss the different context modes, we first describe the calculation of the focus and introduce conditions common to all three modes.

[1]For consistency, the focus is represented as belt 0. The remaining belts form the context and are assigned increasing indices with growing distance from the focus.

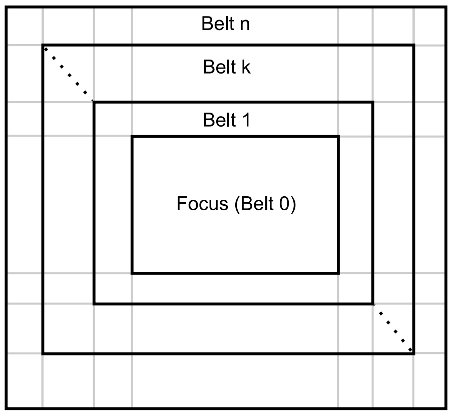


Figure 1: The belts and grid rectangles of the RECTANGULAR FISHEYE VIEW.

3.1 Definition of the focus and the context belts

To specify the fisheye and the inverse fisheye transform, we first introduce some input parameters. For distinguishing between the two coordinate systems, we use variable names starting with c to represent device coordinates and names starting with l to denote logical coordinates:

$lWid$	$\Rightarrow$	Width of logical window
$lHgt$	$\Rightarrow$	Height of logical window
$cWid$	$\Rightarrow$	Width of canvas window
$cHgt$	$\Rightarrow$	Height of canvas window

To describe the belts including the focus, we introduce the notation $lPos_{d,k}$ resp. $cPos_{d,k}$ for the boundary coordinates of the k-th belt ($k \geq 0$) in the four directions $d \in \{left, top, right, bottom\}$ as follows:

$lPos_{d,k}$	$\Rightarrow$	Border of belt k in direction d in the logical window
$cPos_{d,k}$	$\Rightarrow$	Border of belt k in direction d in the canvas window

The user can specify and interactively change the position of the focus boundaries using either logical or device coordinates. In the logical window, these parameters determine which section of the layout the user is interested in most. In the canvas window, they define where and at which size the area of interest is displayed (see also fig. 8). To determine the magnification of the focus, a *focus scaling factor* $Scale_{Focus}$ has to be specified. From these parameters, width and height of the focus can be calculated:

$$\begin{aligned}
cWid_0 &= Scale_{Focus} * lWid_0 && (1)\\
cHgt_0 &= Scale_{Focus} * lHgt_0 && (2)\\
lWid_0 &= lPos_{right,0} - lPos_{left,0} && (3)\\
lHgt_0 &= lPos_{top,0} - lPos_{bottom,0} && (4)\\
cWid_0 &= cPos_{right,0} - cPos_{left,0} && (5)\\
cWid_0 &= cPos_{top,0} - cPos_{bottom,0} && (6)
\end{aligned}$$

Analogously, the widths $lWid_{d,k}$ and $cWid_{d,k}$ of a belt k ($k > 0$) in the direction d are defined as:

$$\begin{aligned}
lWid_{d,k} &= |lPos_{d,k} - lPos_{d,k-1}| && (7)\\
cWid_{d,k} &= |cPos_{d,k} - cPos_{d,k-1}| && (8)
\end{aligned}$$

Given the above definitions and the user defined parameters $cPos_{d,0}$, the logical coordinates $lPos_{d,0}$ of the focus can now be computed as follows:

$$lPos_{left,0} = \frac{cPos_{left,0} \cdot (lWid - lWid_0)}{cWid - cWid_0 + 1} \quad (9)$$

$$lPos_{top,0} = \frac{cPos_{top,0} \cdot (lHgt - lHgt_0)}{cHgt - cHgt_0 + 1} \quad (10)$$

$$lPos_{right,0} = lPos_{left,0} + lWid_0 \quad (11)$$

$$lPos_{bottom,0} = lPos_{top,0} + lHgt_0 \quad (12)$$

The degree of detail in the focus is specified by the user by means of the parameter $Scale_{Focus}$. To scale the context belts, parameters $Scale_{Context,d}$ are used whose exact type is determined by the context mode. Since the focus represents the area of highest user interest, it has to be the part of the display with the highest degree of detail. Thus, the following constraint applies to the scaling factors:

$$\forall d \; Scale_{Focus} \geq Scale_{Context,d} \quad (13)$$

If the constraint 13 is violated (e.g, if $Scale_{Focus}$ is defined too small by the user), another parameter (e.g., the focus size) has to be adjusted until the constraint is satisfied.

3.2 Context modes

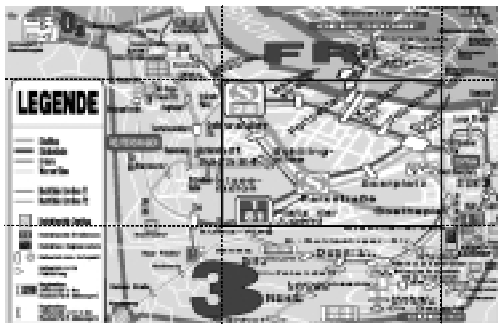
In order to provide scalability with respect to the tradeoff between smoothness and computational requirements, we propose three different context modes.

3.2.1 Uniform context scaling. The uniform context scaling is a straightforward and computationally inexpensive way of mapping the context onto the canvas. Only one context belt is used, which is scaled to fit into the space on the canvas left by the focus. This belt consists of 8 grid rectangles, which are scaled by possibly different factors in X and Y direction. Fig. 2(a) shows an example. The scaling factors are computed as follows:

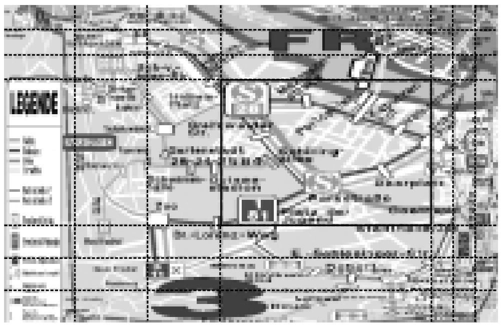
$$Scale_{Context,d} = \frac{cWid_{d,1}}{lWid_{d,1}} \qquad (14)$$

3.2.2 Belt-based context scaling. This context mode uses several context belts and has already been proposed in [5]. By this scheme, displaying less detail with increasing distance from the focus is done in discrete steps. The resulting fisheye view still lacks smooth transitions but can be computed very fast since large areas are assigned the same scaling factor (which may again be different in X and Y direction). In order to decrease the discontinuities, more belts could be used. However, this leads to increased computational effort.

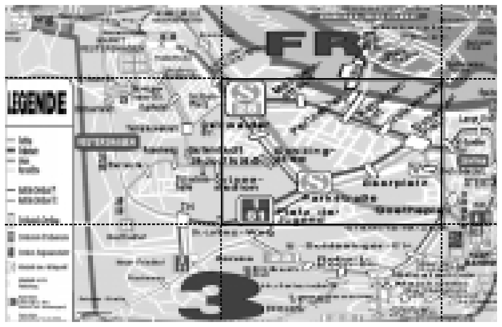
In our system presented in [5], the context is fully configurable with respect to the number and width of the belts. Experiences have shown, that two or three context belts of the same width in the canvas window using scaling factors $Scale_{Context,d,k}$ decreasing by powers of two with increasing distance from the focus provide a suitable context weighting for many applications. On ultra-portable devices, such a design-time context definition offers the advantages of simplified use and reduced computational complexity over an interactive configuration of the context.



(a) uniform



(b) belt-based



(c) non-uniform

Figure 2: Context scaling mode examples.

Fig. 2(b) shows an example with three belts. The parameter $Scale_{Context,d,1}$ of the innermost context belt provides the necessary degree of freedom to exactly fit the contents of the logical window into the canvas window. For the remaining belts, scaling is performed as follows:

$$Scale_{Context,d,1} = \frac{cWid_{d,1}}{lWid_{d,1}} \quad (15)$$
$$Scale_{Context,d,2} = Scale_{Context,d,1}/2 \quad (16)$$
$$Scale_{Context,d,3} = Scale_{Context,d,2}/2 \quad (17)$$

Since the innermost belt is assigned the largest scaling factor, $Scale_{Context,d,1}$ has to satisfy constraint 13.

3.2.3 Non-uniform context scaling. The requirement of a smooth transition from the focus to the context and smooth weighted context scaling can be achieved by using *non-uniform* scaling. The belt layout is the same as in the uniform case - a single context belt is used. Unlike in the former case, the context scaling factor is decreasing continually with increasing distance from the focus boundary. Conceptually, this scaling mode can be seen as composing the context of a very large number of narrow context belts. In order to maintain the geometric properties of the layout in the area of interest, the focus region is still scaled uniformly.

Fig. 2(c) shows an example. The fisheye view features a high visual quality because *non-uniform* scaling greatly reduces discontinuities. However, it is demanding with respect to computing power.

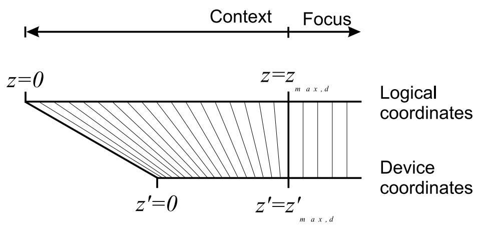


Figure 3: The idea of non-uniform context scaling.

In order to achieve a small scaling factor near the window boundary and a large one near the focus, the degree of distortion is controlled by an exponential function with a user-definable parameter ε. Fig. 3 illustrates this for one direction ($d = left$). In logical coordinates, the parameter z always runs on a line from 0 at the boundary of the logical window to the corresponding focus boundary $z_{max,d} := lPos_d$. Analogously, z' is defined in device coordinates. The position z' of a pixel on the line in the RECTANGULAR FISHEYE VIEW is computed from its position z in the logical window as follows:

$$z' = z^{\varepsilon} \cdot \frac{z'_{max,d}}{(z_{max,d})^{\varepsilon}} \quad (18)$$

Given this coordinate mapping, the context scaling can be performed using separable image warping algorithms (cf. section 5). The exponent ε can be specified by the user but may have to be adapted to satisfy constraint 13. To check this constraint, we consider the device coordinates of the first context pixel immediately outside the focus boundary in direction d and transform them to logical space. The variables $z_{0,d}$ and $z'_{0,d}$ denote the position of that pixel on the line between window and focus boundary in the canvas resp. the logical window. Having that, we can compute the context scaling factor to be used for checking constraint 13 as follows:

$$Scale_{Context,d} = \frac{z'_{max,d} - z'_{0,d}}{z_{max,d} - z_{0,d}} \quad (19)$$
$$= \frac{1}{z_{max,d} - z_{0,d}} \quad (20)$$

4 Interaction

Downscaling the context saves screen space but decreases the degree of detail. Thus, the user must be enabled to re-position and re-scale the focus should the area of interest change. These interaction functions provide a means for an intuitive exploration of the layout.

4.1 Interaction functions

The most important functions `MoveFocus`, `JumpFocus` and `DefineFocus` allow to re-position the focus. Where `MoveFocus` denotes small and continuous position changes (e.g., dragging the focus using the mouse), `JumpFocus` allows for fast positioning the focus in another area (e.g., by a mouse click). `DefineFocus` can be used to directly specify a new size and position of the focus.

Already in section 2, we have described that the part of the graphical layout in the focus is mapped onto the canvas using a scaling factor $Scale_{Focus}$. This factor controls the degree of detail in the focus. It can be defined interactively using the `ZoomFocus` function in order to adapt the tradeoff between the aspects *detail* and *overview* to the task at hand:

- A small $Scale_{Focus}$ provides less detail in the focus and accentuates the overview aspect, since the focus region on the canvas is small, leaving more space for the context and thus allowing less distortion.
- A large $Scale_{Focus}$ leads to a great degree of detail in the focus and thus focuses on the detail aspect. There is little space for the context, requiring strong distortion.

By means of the `ResizeFocus` function, the user can interactively modify the size of the focus in the canvas window. There are two different opportunities how this change can affect the parameters of the fisheye transformation.

- First, the size of the focus in the logical window can be increased proportionally, changing $lPos_{d,0}$ and leaving $Scale_{Focus}$ constant. This so called `area` mode allows to enlarge or collapse the part of the layout to be displayed in the focus and thus to adapt the area of interest to changed user needs.
- Second, the focus scale factor can be adapted, leaving the portion of the layout to be displayed and thus the corner coordinates $lPos_{d,0}$ of the focus in logical space constant. This mode is named `magnifier` mode since it provides an intuitive way to changing $Scale_{Focus}$, like changing the focal length of a zoom lens.

On platforms which provide a window system, we must additionally handle the interactive resizing of the canvas window (`CanvasResize`). Since the RECTANGULAR FISHEYE VIEW always uses the full extent of the canvas window, parameter changes must reflect a canvas size change. Again, there are various opportunities to react to a `CanvasResize` event:

- In `area` mode, the focus area is changed proportionally.
- In `magnifier` mode, the focus scale factor is changed.
- An additional opportunity could be to modify the context area in the canvas, thus changing the amount of context distortion.

4.2 Interactivity-related performance issues

Since the RECTANGULAR FISHEYE VIEW exploits distortion in the context, all user interactions involving small incremental changes must provide rapid system feedback in order to support the user in developing a “feeling” for the distorted presentation. However, redrawing the complete canvas may be a time-consuming operation depending on the capabilities of the mobile device and on the context mode (see section 3.2). Thus, the two concepts of *interruptible display refresh* and *ghost feedback* have been developed in order to provide additional computational scalability.

Interruptible refresh updates the display in several steps, one grid rectangle (see fig. 1) at a time, starting with the focus region and moving from there towards the outermost context ring. The display refresh is aborted if after one of these steps a new interaction occurs.

Ghost feedback is a method which can greatly reduce the amount of computation required during interactions like `MoveFocus` involving many small incremental changes. Instead of refreshing the display after each incremental interaction, the display is frozen at the state immediately before the interaction sequence started, and an outline of the new focus is overlayed over it. As long as the user moves or scales the focus, the outline follows on the canvas window, changing its size and position. After completing the interaction sequence, the complete RECTANGULAR FISHEYE VIEW is redrawn using the new parameters. This method is especially suitable for very slow ultra-portable devices.

5 Implementation

For general graphical data the RECTANGULAR FISHEYE VIEW can be implemented in two different ways which differ in the realisable context modes, in the achievable visual quality and with respect to the amount of computation power and memory required.

5.1 Implementation based on rendering pipeline

The first implementation opportunity is based on exploiting the rendering pipeline of the available graphics subsystem for realising the fisheye transformation. For each grid rectangle (cf. fig. 1), the following steps are performed:

- The current grid rectangle coordinates are passed to the graphics system as clip rectangle.
- The viewing transform is set to represent the scaling of the current rectangle in X and Y direction according to either $Scale_{Focus}$ or $Scale_{Context,d}$.
- All those graphical primitives which have a bounding box intersecting the clip rectangle are rendered using functions of the graphics subsystem. The clip rectangle and viewing transform ensure that the correct region of the layout is drawn at the correct scale.

Since non-uniform scaling is usually not directly supported by the pipeline, this restricts the realisable context scaling to the uniform and belt-based modes.

5.2 Implementation with intermediate bitmap

The second implementation alternative uses an intermediate bitmap onto which the content of the logical window is drawn at the scale of the focus in a preprocessing step. A mapping step maps this bitmap onto the canvas window to perform the necessary fisheye transformations, treating each grid rectangle separately. For the mapping, either StretchBlt functions of the graphical subsystem or a separable resampling algorithm proposed by Fant for image warping [7, pp. 153ff.] can be used.

Unlike StretchBlt, Fant's algorithm can perform non-uniform context scaling by applying area sampling to the bitmap. Due to the separability of the algorithm, the exponential function which controls the context scaling can be precomputed and stored separately for the X and the Y direction. This has to be done for each of the nine grid rectangles.

The implementation with intermediate bitmap can support all three context modes if resampling is used. However, it consumes more memory and computing time than the rendering pipeline based method, requiring the combination with ghost feedback on most systems to provide interactive response times.

6 Results

Sarkar and Brown [6] demand smoothness as an important feature of fisheye views. The proposed three context modes (cf. section 3.2) provide this feature to varying degrees, trading off smoothness and visual quality against computational complexity. The following section evaluates this tradeoff.

Smoothness. Using uniform context scaling, there is a sudden drop of the scaling factor at the focus boundary. Belt-based context scaling replaces this large drop by a number of smaller drops, improving the perceived smoothness. Perfect smoothness is provided by non-uniform scaling. Fig. 2 illustrates this.

Computational complexity. This criterion depends on the implementation alternative used. Uniform and belt-based context scaling can be implemented using both alternatives presented in section 5. For non-uniform scaling, only the implementation with intermediate bitmap and resampling can be used. Fig. 4

compares the computing times for using an intermediate bitmap scaled by these two alternatives. It is obvious that the resampling algorithm consumes more computing time than the StretchBlt alternative. This is due to the fact that during resampling, a computationally expensive area sampling operation is performed. In both cases, the refresh rate depends mainly on the size of the intermediate bitmap given a fixed canvas window size. The implementation using the rendering pipeline (see fig. 6 requires more time compared to using an intermediate bitmap with StretchBlt, but less time than using the bitmap with resampling. Unlike the bitmap implementation, the refresh rate depends on the number of graphical primitives. Interruptible display increases the refresh rate in all cases.

Visual quality. The achievable visual quality greatly depends on the context mode and the scaling method used. Both the rendering-pipeline-based and the bitmap-and-StretchBlt-based implementation scale by omitting pixel rows or columns, which leads to inferior visual quality in context areas. Fig. 7 illustrates this for the belt-based context mode implemented using the rendering-pipeline-based approach. A bitmap-and-resampling-based implementation interpolates between neighbouring pixels, achieving higher visual quality as shown in fig. 5 for non-uniform scaling.

7 Conclusion and Future Work

In this paper, we have presented a generalisation of our earlier work [5] on RECTANGULAR FISHEYE VIEWS for image transmission. We have proposed three different modes for scaling the context area and implementation alternatives for these modes. The alternatives differ with respect to smoothness, visual quality and computational complexity. Thus, they provide scalability with respect to the tradeoff between quality and computing time. Looking at the interactivity of the method, we discussed briefly opportunities for handling size changes of focus region and canvas window and looked at methods for speeding up system feedback during incremental interactions.

The RECTANGULAR FISHEYE VIEW can save a large amount of screen space for graphical layouts – in typical scenarios, between 95 and 75%. Especially on mobile computing devices with small displays, they may enable graphical applications known today on desktop computers and workstations. To be content with the limited computing power of these devices, the scalability of the method has to be exploited by the system designer.

We have implemented the discussed alternatives under Windows 98 as a set of C++ classes which can be integrated into existing graphical applications, adding the fisheye transformation to the graphical output and input pipelines. Future work involves looking at particular applications and porting the experimental system to ultra-portable devices running Windows CE.

Acknowledgement

This work was supported by The German Science Foundation (DFG) under contract no. Schu-887/3-3.

References

[1] T.A. Keahey. The nonlinear magnification homepage. http://www.cs.indiana.edu/hyplan/tkeahey/research/nlm/nlm.html, Oct. 26 1998. Version 1.5.

[2] T.A. Keahey and E.L. Robertson. Techniques for nonlinear magnification transformations. In *Proc. IEEE Symposium on Information Visualization, IEEE Visualization*, pages 38–45, October 1996.

[3] J. Lamping, R. Rao, and P Piroli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. CHI*, 1995.

[4] U. Rauschenbach. The Rectangular Fish Eye View as an efficient method for the transmission and display of large images. In *Proc. IEEE International Conference on Image Processing (ICIP99)*, Kobe, Japan, Oct. 25-28 1999.

[5] U. Rauschenbach, T. Weinkauf, and H. Schumann. Interactive focus and context display of large raster images. In *Proc. WSCG'2000, The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media*, Plzen, Czech Republic, February 7-11 2000.

[6] M. Sarkar, S.S. Snibbe, O. Tversky, and S.P.. Reiss. Stretching the rubber sheet. In *Proc. ACM Symposium on User Interface Software and Technology*, 1993.

[7] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, 1990.

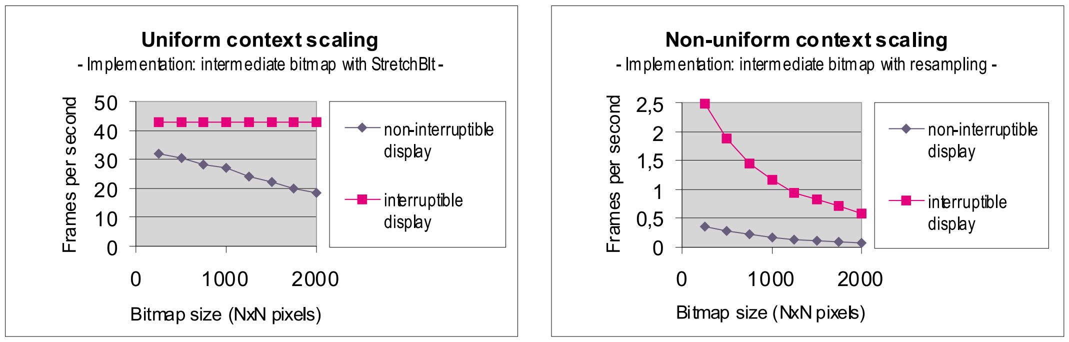
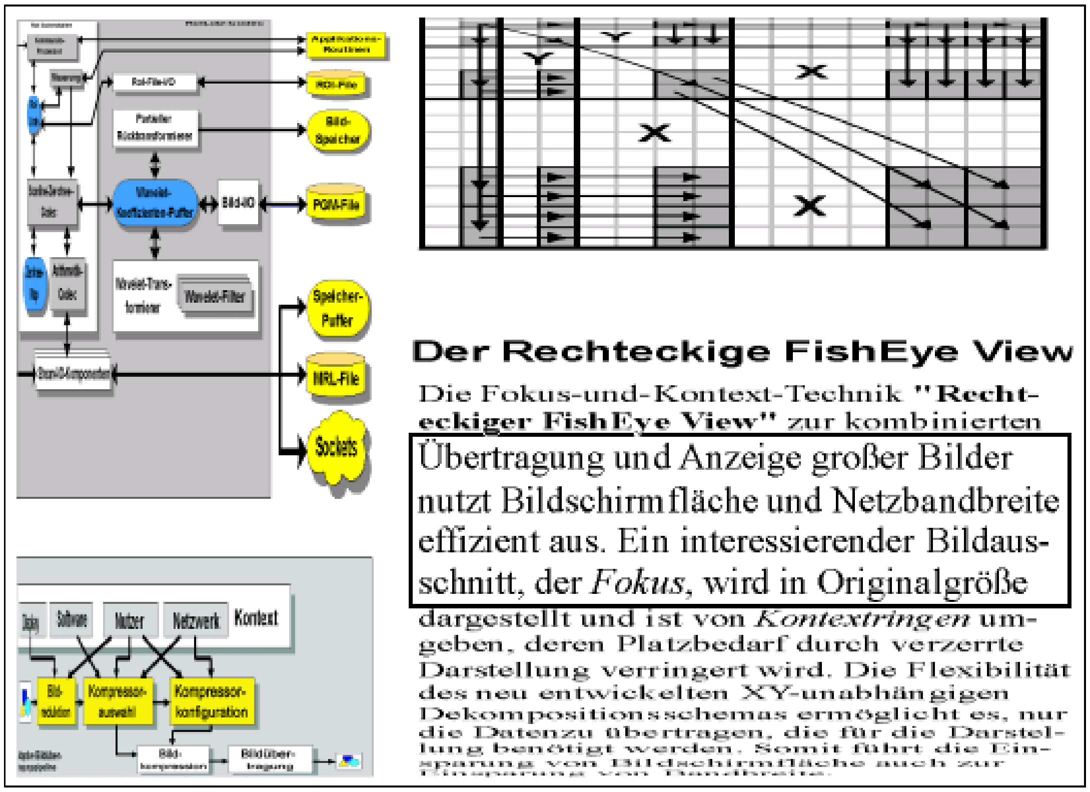


Figure 4: Display refresh rates[2] of the implementation using an intermediate bitmap. Left: Uniform scaling by StretchBlt operation of graphical subsystem. Right: Non-uniform scaling by Fant's resampling algorithm.



Der Rechteckige FishEye View

Die Fokus-und-Kontext-Technik "Rechteckiger FishEye View" zur kombinierten Übertragung und Anzeige großer Bilder nutzt Bildschirmfläche und Netzbandbreite effizient aus. Ein interessierender Bildausschnitt, der *Fokus*, wird in Originalgröße dargestellt und ist von *Kontextringen* umgeben, deren Platzbedarf durch verzerrte Darstellung verringert wird. Die Flexibilität des neu entwickelten XY-unabhängigen Dekompositionsschemas ermöglicht es, nur die Daten zu übertragen, die für die Darstellung benötigt werden. Somit führt die Einsparung von Bildschirmfläche auch zur Einsparung von Bandbreite.

Figure 5: RECTANGULAR FISHEYE VIEW displaying a 2D vector graphics layout using a 1024x713 pixel background bitmap and non-uniform context scaling with $\varepsilon = 1.5$. Canvas window size: 512x357 pixels.

[2]Refresh rates were measured on an INTEL Pentium with 200MHz and 32MB RAM running Windows 98, using a canvas window of 800x600 pixels and a focus size of 400x300 pixels.

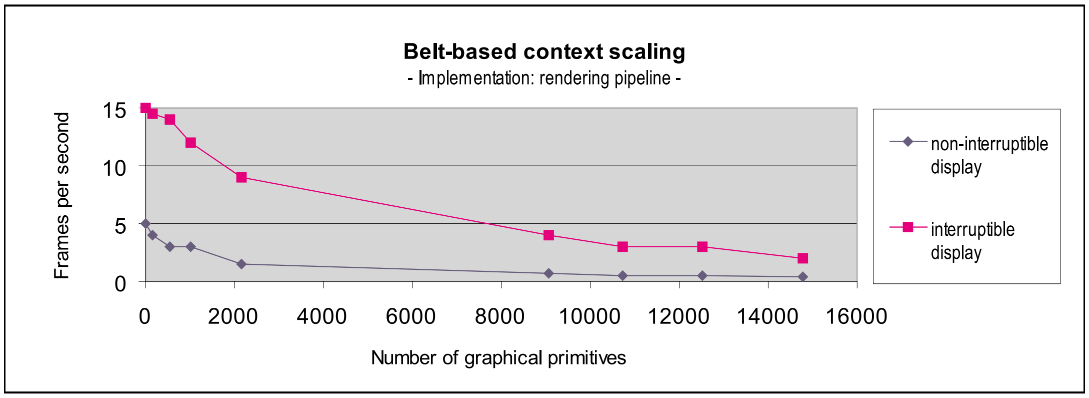


Figure 6: Display refresh rates[3] of the implementation using the rendering pipeline of the graphical subsystem.

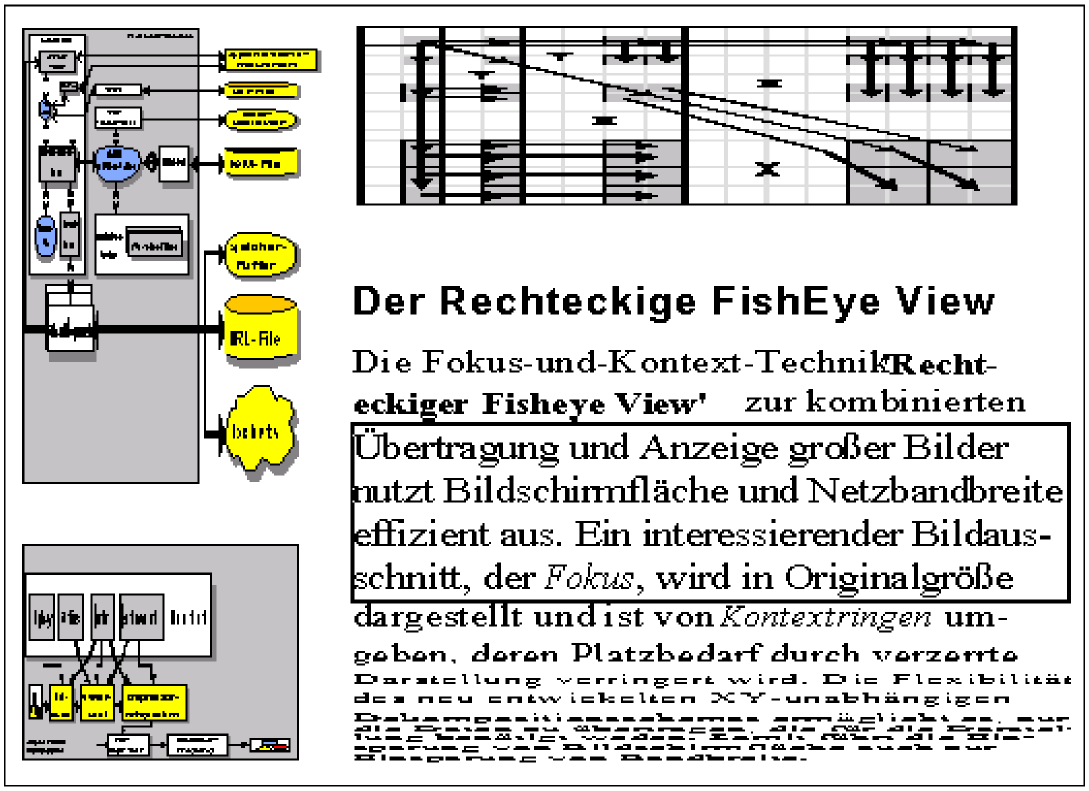

Der Rechteckige FishEye View
Die Fokus-und-Kontext-Technik 'Recht-eckiger Fisheye View' zur kombinierten
Übertragung und Anzeige großer Bilder nutzt Bildschirmfläche und Netzbandbreite effizient aus. Ein interessierender Bildausschnitt, der *Fokus*, wird in Originalgröße
dargestellt und ist von *Kontextringen* um-
geben, deren Platzbedarf durch verzerrte
Darstellung verringert wird. Die Flexibilität
des neu entwickelten XY-unabhängigen
IRL-File
belts

Figure 7: RECTANGULAR FISHEYE VIEW displaying a 2D vector graphics layout using belt-based context scaling with 3 belts.

[3]Refresh rates were measured on an INTEL Pentium with 200MHz and 32MB RAM running Windows 98, using a canvas window of 800x600 pixels and a focus size of 400x300 pixels.

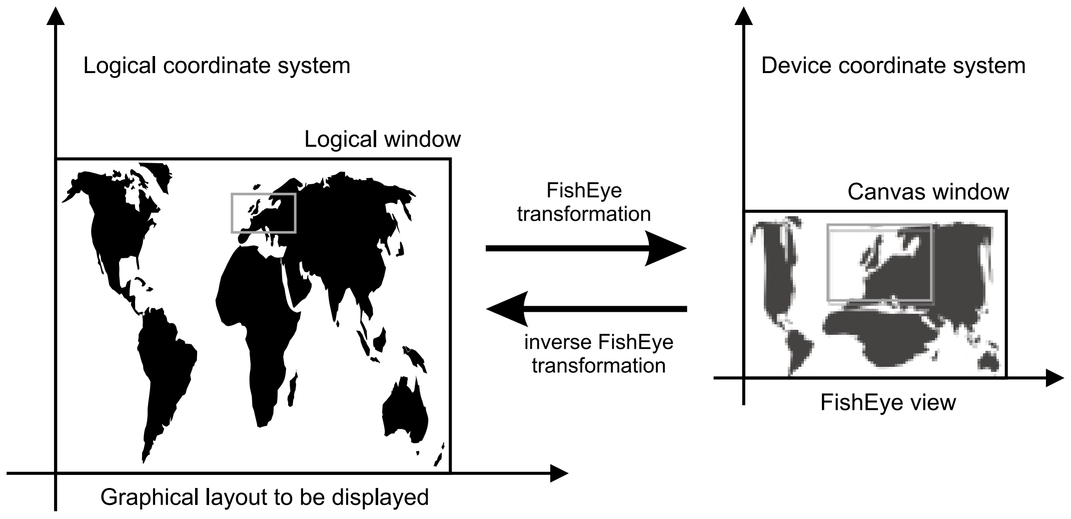

Logical coordinate system
Logical window
Graphical layout to be displayed
FishEye transformation
inverse FishEye transformation
Device coordinate system
Canvas window
FishEye view

Figure 8: Illustration of the fisheye transformation.