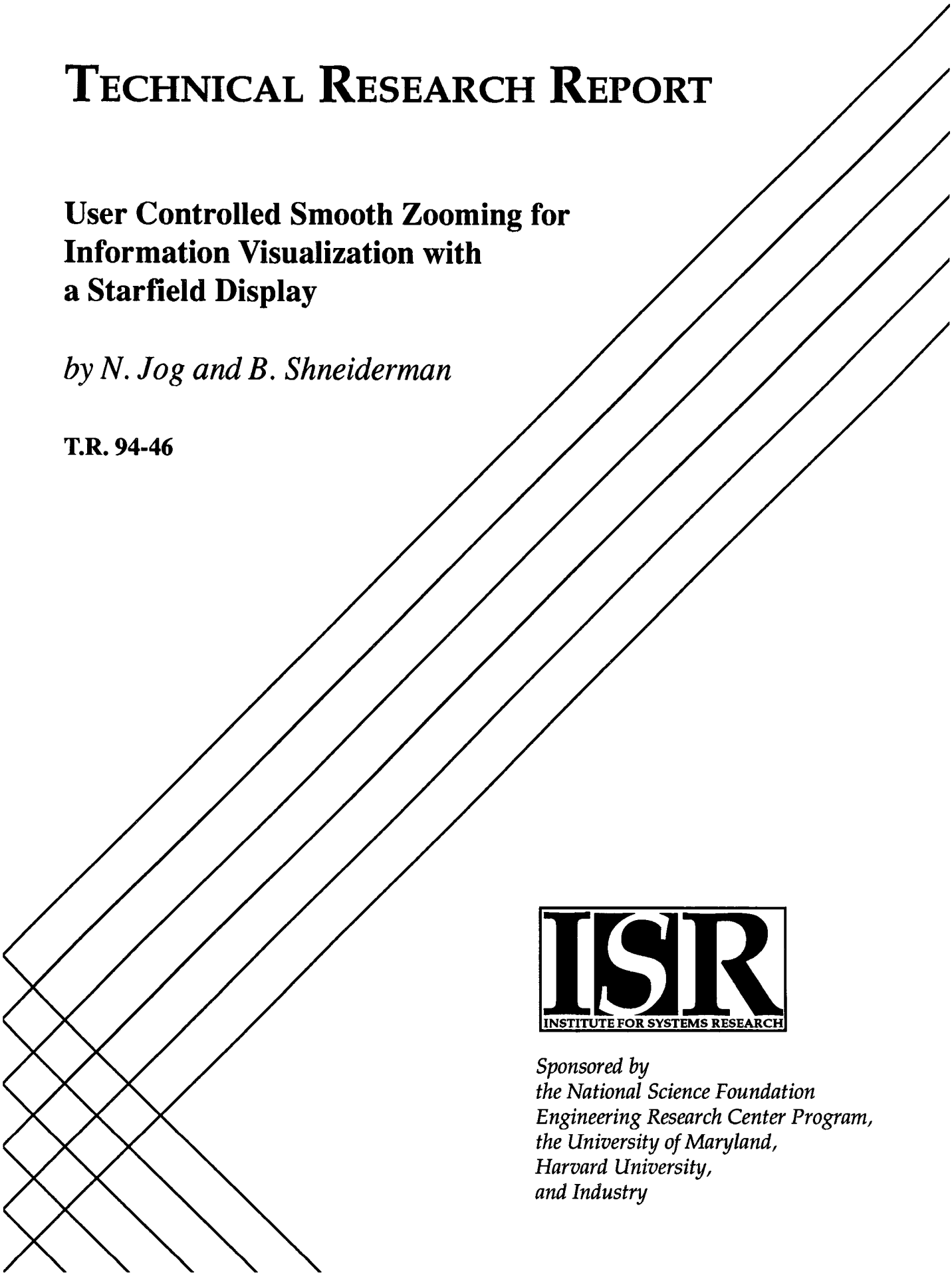
TECHNICAL RESEARCH REPORT

User Controlled Smooth Zooming for Information Visualization with a Starfield Display

by N. Jog and B. Shneiderman

T.R. 94-46



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

CAR-TR-714
CS-TR-3286
ISR-TR-94-46

June 1994

User Controlled Smooth Zooming for Information Visualization with a Starfield Display

Ninad Jog [1] and Ben Shneiderman [2]

Human-Computer Interaction Laboratory
Center for Automation Research
[1]Department of Computer Science
[2]Institute for Systems Research
University of Maryland, College Park, MD 20742-3255
ninad@cs.umd.edu, ben@cs.umd.edu

Abstract

This paper discusses the design and implementation of user controlled smooth zooming of a starfield display. A starfield display is a two dimensional graphical visualization of a multidimensional database where every item from the database is represented as a small colored rectangle whose position is determined by its ranking along ordinal attributes of the items laid out on the axes. One way of navigating this visual information is by using a zooming tool to incrementally zoom in on the items by varying the attribute range on either axis independently - such zooming causes the rectangles to move continuously and to grow or shrink.

To get a feeling of flying through the data, users should be able to track the motion of each rectangle without getting distracted by flicker or large jumps - conditions that necessitate high display refresh rates and closely spaced rectangles on successive frames. Although the use of high-speed hardware can achieve the required visual effect for small databases, the twin software bottlenecks of rapidly accessing display items and constructing a new display image fundamentally retard the refresh rate. Our work explores several methods to overcome these bottlenecks, presents a taxonomy of various zooming methods and introduces a new widget, the zoom bar, that facilitates zooming.

The 10-year old Human-Computer Interaction Laboratory (HCIL) is an interdisciplinary effort within the Center for Automation Research. The main participants are faculty, staff, and students from the Department of Computer Science, Department of Psychology, and College of Library and Information Services at the University of Maryland, College Park, MD.

For single copies
or a list of all HCIL
technical reports
please write to:

Teresa Casey
Human-Computer
Interaction Laboratory
A.V. Williams Building
University of Maryland
College Park MD 20742

email tcasey@cs.umd.edu

User Controlled Smooth Zooming for Information Visualization with a Starfield Display

Ninad Jog† and Ben Shneiderman*
Human Computer Interaction Lab
Institute for Systems Research
University of Maryland
College Park MD 20742-3255 USA
e-mail: ninad@cs.umd.edu, ben@cs.umd.edu

† Department of Electrical Engineering
* Department of Computer Science

ABSTRACT

This paper discusses the design and implementation of user controlled smooth zooming of a starfield display. A starfield display is a two dimensional graphical visualization of a multidimensional database where every item from the database is represented as a small colored rectangle whose position is determined by its ranking along ordinal attributes of the items laid out on the axes. One way of navigating this visual information is by using a zooming tool to incrementally zoom in on the items by varying the attribute range on either axis independently - such zooming causes the rectangles to move continuously and to grow or shrink.

To get a feeling of flying through the data, users should be able to track the motion of each rectangle without getting distracted by flicker or large jumps - conditions that necessitate high display refresh rates and closely spaced rectangles on successive frames. Although the use of high-speed hardware can achieve the required visual effect for small databases, the twin software bottlenecks of rapidly accessing display items and constructing a new display image fundamentally retard the refresh rate. Our work explores several methods to overcome these bottlenecks, presents a taxonomy of various zooming methods and introduces a new widget, the zoom bar, that facilitates zooming.

KEYWORDS

starfield display, smooth zooming, animation, double buffering, zoom bar, bucket methods, dynamic queries, information visualization, focal line.

INTRODUCTION

The exploration of large multidimensional databases is greatly facilitated by presenting information visually and letting the user dynamically query the database using filtering tools that cause continuous visual updates at a rate of at least 15 frames per second [5]. Such dynamic query applications typically encode multidimensional database items as dots or colored rectangles on a two-dimensional scattergram, called a starfield display, with ordinal attributes of the items laid out along the axes. Geographic applications arise as natural candidates for dynamic queries by representing latitude and longitude along the axes - thereby making the starfield display a map of locations. Other databases can also exploit the starfield display paradigm by mapping two ordinal attributes along the axes and using a third to color code the dots. Additional attributes can be controlled by widgets such as sliders and buttons.

Many applications can employ Visual Information Seeking (VIS) principles [1, 13, 9] to facilitate rapid information browsing and empower users to find patterns and exceptions at a glance. VIS principles encompass direct manipulation, rapid query filtering using sliders and buttons, immediate and continuous visual updates of results, tight coupling - i.e. interrelating query components to preserve display invariants and zooming the starfield display to reduce clutter.

Unlike traditional applications such as image browsers that do zooming in large fixed stages, zooming a starfield display should be incremental and flicker-free so that users can track the motion of each rectangle. This gives users a feeling of flying through the data instead of getting disoriented by sudden large changes in view [3]. This paper deals with the design and implementation of such smooth zooming on a prototype dynamic queries application, the FilmFinder [1].

Whereas zooming an arbitrary image in real time necessitates computations at every pixel, zooming a starfield display is a simpler problem because computations have to be done only for the colored

rectangles - not for the background - and there are just hundreds to thousands of rectangles as opposed to a million pixels.

THE FILM FINDER

The FilmFinder (figure 1.1 thru 1.3) is a visualization of a database of two thousand movies. Each film has attributes such as its title, director, the year of production, its popularity on a scale of 1 to 10, the list of actors and actresses, length in minutes, category (drama, comedy, horror, etc.) and rating. The year of production (ranges from 1920 thru 1993) is laid out on the x-axis and the popularity (scale of 1 to 10) is laid out on the y-axis so that recent popular movies appear at the top right of the starfield display. Categories are color coded, so that drama films appear as red rectangles, musicals as yellow and so on. The database has more recent movies than old ones i.e. the distribution of data is non-uniform. Additionally, the data is constant over time - i.e. successive invocations of the FilmFinder use the same frozen data set.

Clicking on a rectangle pops up an information card that lists the attributes of the selected film and shows a still picture from it. The number of rectangles that appear in the starfield display can be continuously varied by selecting different attributes of the films (e.g. show just dramas) or by changing the scale on the x and y axes (e.g. show films made between 1940 and 1970).

These query filters are implemented using widgets such as toggles for category and rating; AlphaSliders [2] for the title, actors, actresses, director; a range selection slider [5] for movie length and a new widget called the zoom bar for varying the scale on the x and y display axes.

GLOBAL AND LOCAL EFFECTS OF VARYING ATTRIBUTES

A distinction can be made between query filters that have local effects and those whose effects are global. For example filters such as an AlphaSlider for selecting the name of an actor affect a small number of display rectangles compared with the total number of films in the database, whereas category toggles have a global effect both in terms of the large number of films they affect and the large display area over which these changes take place.

Since a zooming action changes the scale on either of the display axes, forcing a redisplay of all rectangles, the scale change filter is a global-effect filter. The classification of attributes as global-effect and local-effect ones is highly application-dependent. For example if a query such as "Display all films whose directors' names begin with B" were to be supported, the director would be classified a global-effect attribute.

Because changes to global-effect attributes take a longer time to render, special data storage and access techniques have to be employed to speed up the display refresh rate. Thus it is necessary to make a distinction between these two types of attributes at the outset of the visualization design.

THE ZOOMING MECHANISM

Zooming is done by changing the range of attributes on either axis individually - e.g., when the upper limit of years is continuously decreased from 1993 to 1960, the scale on the x-axis keeps increasing. Rectangles representing recently-made movies keep leaving the display and the ones that are within the range grow in size and move so that the display range occupies the entire width of the starfield, looking like a patterned rubber mat getting stretched.

On the other hand, varying any of the non-axis attributes like category or film length causes rectangles to drop out of the starfield or get added to it, but there is no scaling or movement involved.

TYPES OF ZOOMING: A TAXONOMY

Zooming of any image can be 1) Continuous or 2) Discrete. Zooming in discrete jumps is done when drawing the new view involves substantial computations (e.g. zooming an arbitrary picture), or when there is nothing to be gained by doing a continuous zoom, as in changing the view size of text in a desktop publishing application.

Continuous zooming is done when it is important to give the user a feel of flying through a space - say a world of 3-D graphical objects as in virtual reality applications, or in an information visualization like a starfield display. This allows users to get more detail in areas of intense interest and preserve the sense of location in the surrounding items.

Since continuous zooming requires rapid redrawings, the image must consist of simple objects that can be hierarchically structured. It is difficult to do it on an arbitrary image.

Another way of classifying zooming is based on the effects it has over the entire image.

1) Some zooming methods typically use a lens that can be moved over an image. The magnified portion appears within the lens boundaries or in a separate window, while the rest of the image stays undistorted. In either case, there is a sharp discontinuity at the boundaries, so users need to mentally integrate the two views. Ghostview, an X-Window application for viewing postscript files uses such a lens, as does the Magic Lens [4].

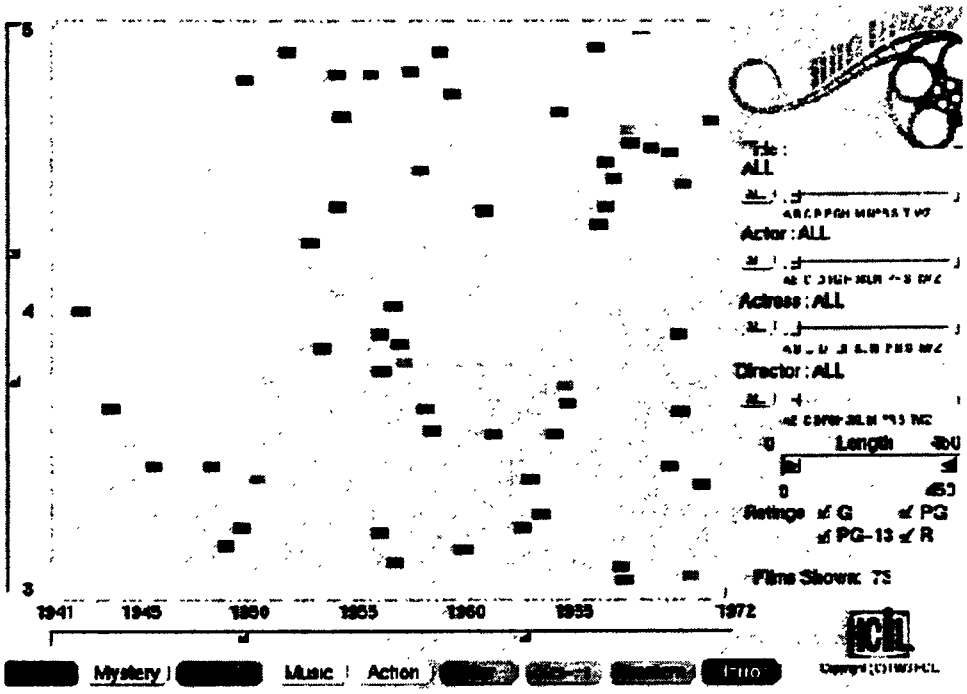


Figure 1.1 The Film Finder showing movies from 1971 through 1972

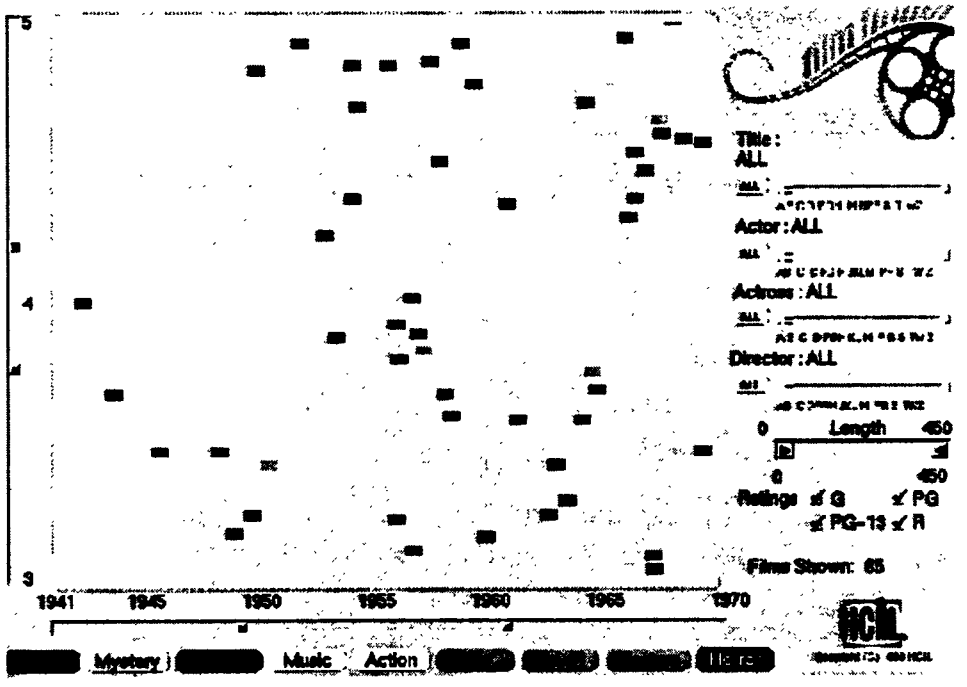


Figure 1.2. Zooming in to display movies from 1941 through 1970

2) With a fisheye lens [11], the selected object is magnified the most, and surrounding objects are progressively diminished in size, giving a perspective view. Fisheye views free the user of the burden of mentally integrating two discontinuous pictures, but they always retain all the information on the screen, making it look cluttered.

3) The third type of zooming changes the scale on one or all of the display axes, causing information to leave or enter the viewing area. This has the advantage of uncluttering the screen, but if it is done in big discrete jumps, the user can feel disoriented.

Any of these three zoom methods can be done either continuously or in big discrete jumps depending upon

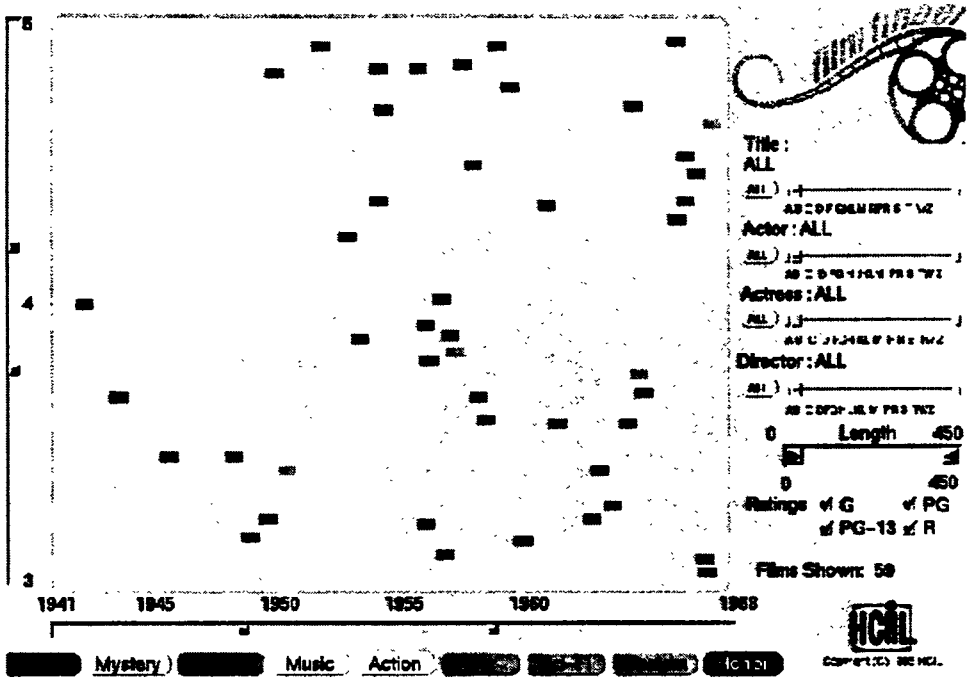


Figure 1.3. Further zooming in shows movies from 1941 through 1968

the complexity of the image. The FilmFinder uses the third method in a continuous manner. Figures 1.1 thru 1.3 show three successive views of the starfield display when zooming is done along the x-axis by decreasing the upper range boundary of the year of production.

PURPOSE OF ZOOMING

Zooming an information visualization display can have twin purposes. When used on images or hierarchy-oriented diagrams like network node-link diagrams [12], successive views can reveal previously hidden detail. For example, each node in a node-link diagram of a network may function as an icon of a subnetwork, so that zooming in would reveal the details of the sub network as another node-link diagram one rung lower in the hierarchy of networks. This use of zooming is akin to magnifying an image at a selected point.

The other use of zooming is to reduce visual clutter by filtering out data points that lie outside the new zoom range. The FilmFinder uses zooming to achieve the latter effect. As a consequence of zooming, rectangles that overlapped partially in the zoomed out view get spread apart, making it easier to click on them.

To make the zooming seem more realistic, rectangles change size as they are zoomed, but the change is bounded to prevent them from shrinking into nothingness or growing to occupy a large part of the screen. It is also easier to select a larger rectangle than a smaller one, and the amount of zoom can be gauged by looking at the size of the rectangles.

Zooming is trickier to do than panning because of the following reasons.

1) Panned objects translate by the same amount without changing in size but zoomed objects change size and also move by different amounts depending upon their distance from the focus. This means that zooming exacts more extensive geometrical recomputations than panning.

2) The amount of translation of each zoomed rectangle in either the x or y direction is a function of both the current range boundaries, making it impractical to precompute increments and store in a lookup table.

ZOOM BARS

Whereas existing widgets for range selection or discrete categories were adequate, none of them was suitable for zooming. Zoom widgets such as user-draggable lens tools are suited for applications where zooming takes place in jumps over small image areas, but they don't work when zooming is continuous over the entire screen and is triggered by changes to a range boundary.

We tried using a pair of buttons for increasing and decreasing each range boundary, but the use of these buttons proved confusing to users - not only were they unsure of whether a button decreased or increased a boundary, but it was also hard for them to see the link between changing a range boundary and the concomitant zooming effect.

We overcame this deficiency by developing a new widget called the zoom bar (See figure 2). A zoom bar is a slider with two thumbs, each controlling one of the extremes of a range. When the right thumb is moved, the upper range boundary is increased or decreased, causing a zoom out or a zoom in by changing the scale on the corresponding display axis. Similarly, the left thumb controls the lower range.

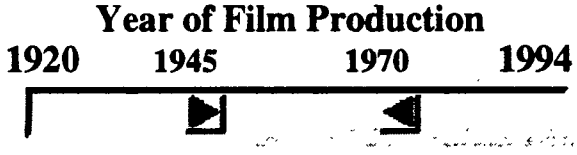


Figure 2. The zoom bar: The viewing range is varied by sliding either of the two thumbs.

The thumbs can come close together by no more than a minimum separation - this separation defines the maximum zoom in of the view. When the two thumbs are at the opposite ends of the slider, the view is fully zoomed out. The scale of each of the x and y display axes is varied individually by its own zoom bar.

When the zoombar is clicked in the channel on a position other than the thumb, the thumb closest to the clicked point snaps to that location. This causes a jump in one of the range values and results in discrete zooming.

Advantages and Limitations

The zoom bar is intuitive and easy to use because of its similarity with a scrollbar. It occupies a small rectangular area (saving on precious screen real estate) and its operation is rapid because of its small size. It is equally well suited for both continuous and discrete zooming.

However, it cannot be used for panning, and its operation is restricted to one display dimension at a time.

AN ALTERNATIVE TO THE CAMERA MODEL

While traditional continuous zoom techniques give the user the impression of flying perpendicular to a plane (the starfield display plane) towards a fixed point, our novel zooming technique gives the impression of a rubber carpet getting stretched and contracted, with the user standing at a fixed distance from it. This fundamental difference arises because our zooming scheme provides for independently alterable zoom factors for the x and y axes.

These different zooming techniques can be described as the view seen through a camera which has a variable focal length and whose position can be changed.

Traditional zoom techniques employ one of two possible schemes. Either the camera stays at a fixed place and the focal length changes over time - as in a real camera, or the camera's field of view remains fixed and the camera moves towards or away from a fixed point. If either case were used in a starfield display, rectangles would appear to move away from the focal point in all directions during zooming in, and appear to converge toward the focal point during zoom out, as shown in figure 3.

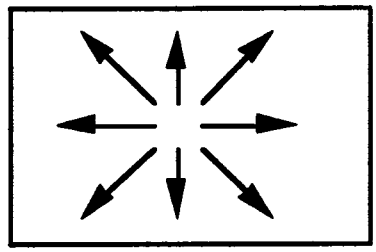


Figure 3. Zooming in causes rectangles to move radially outward from the focal point.

By contrast, zooming in the FilmFinder is done independently in the x and y directions, so there is no single focal point. Instead, there is a *focal line* corresponding to the range boundary that remains fixed. This line could be any of the four sides of the starfield display.

For example, if zooming is done in the horizontal direction and the upper range is increased, the focal line is the left side of the display. Rectangles that lie close to the left boundary move by very small amounts compared with the ones near the right boundary, though all rectangles shrink by the same amount.

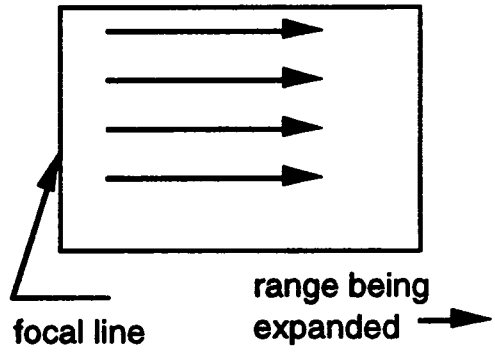


Figure 4. Zooming along x-axis. Arrows show rectangle motion.

As the range increases, rectangles enter the viewing range from the right and start moving leftwards. Figure 4 shows this behavior. In other words, rectangles flatten as they move toward the focal line and elongate as they move away from it.

If a traditional zooming technique with a user-defined focal point of interest is used in the absence of an

additional overview screen, the user may feel lost even if the zooming is smooth. However, when the individual-axis zooming of this paper is used, the user's view is always firmly anchored to one of the sides of the starfield display. We conjecture that this leads to a better grasp of location and thereby improved user satisfaction.

IMPLEMENTATION

The FilmFinder is implemented on a Sun SPARCstation 1+ using Galaxy/C, a platform independent application environment developed by Visix Inc. The FilmFinder can be ported to several platforms - from a slow DOS machine running Windows to a fast Sun, so it becomes imperative to seek software speedups and optimizations to get a rapid display refresh rate instead of relying solely on the use of faster hardware.

Galaxy's powerful object-oriented constructs were used to build customized widgets and other parts of the program. The film data is in flat-file format, taken from the Internet. The entire data is read into a linear array and sorted by several other attributes like length, actors, etc. Each of the individual sorted lists contains pointers (indices) to the linear-array database i.e. the set of all the records are always in the linear array in the order in which they were read in.

ATTAINING SMOOTH ZOOMING

Zooming can be speeded up both by speeding up the data storage and access techniques and by using appropriate display strategies.

Apart from the starfield display metaphor, the display of lighted rectangles can also be thought of as the visualization of a 2-dimensional sparse matrix, with zooming being used along either axis to select a range of indices. This correspondence suggests that many of the techniques used for storing and manipulating sparse matrices can be used for representing the underlying data, thereby speeding up the rendering rate sufficiently enough to maintain the illusion of zooming.

The following techniques are used.

1) Efficient Storage

The film data items are stored in buckets that are indexed by the year in which they were made [9] as shown in figure 5. Each bucket is a linear array whose maximum size depends upon its year index - for example, the bucket for films made in 1990 is longer than the one for 1930 because the database has more recent films than old ones.

By finding out the number of films per year, the data distribution is known *a priori*. The film distribution is evenly clustered over three ranges. Between the years of 1920 and 1960, there are at most 20 films per year, at most 50 in the next window till 1984 and up to 150 for each of the successive years. Thus the maximum size of each bucket is fixed.

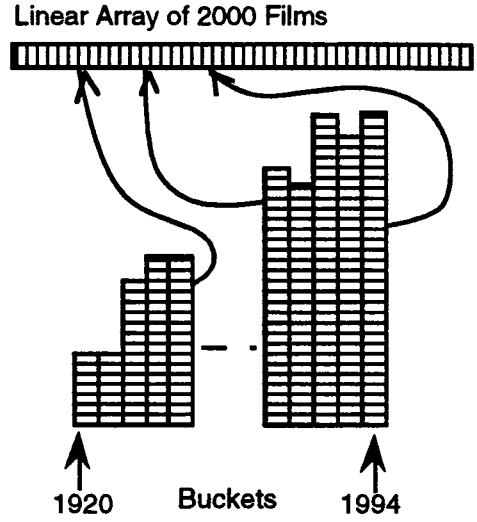


Figure 5. Films in buckets indexed by year. Each bucket has films of all popularities, each position in bucket points to the film's record in the linear array of 2000 films.

The bucket method uses O(kn) space, where k is a constant that depends on the data distribution and n is the number of films. For each valid data slot in the buckets, there are k unused ones. The value of k can be optimized by doing an analysis of the input distribution. For the FilmFinder, there are approximately twice as many bucket slots as there are data items, so k equals 2.

Although an *a priori* knowledge of the data distribution gives an edge by enabling the buckets to be configured as static arrays, this histogramming can be done on the fly and the buckets can be made of dynamically allocated arrays. This strategy would be useful in applications where the variable on a display axis can be altered by the user during the course of the program's execution.

When either the year or popularity range is varied to produce zooming, films from the corresponding buckets are compared for popularity and cached into an array if their popularity lies within the current popularity range. This operation involves one indirection (to address the proper bucket) and two comparisons (to check the popularity range) for each film in the bucket.

Alternatively, the grid file method would use a two-dimensional array indexed by both year and popularity to point to the proper bucket. This method wastes space though - it inherently needs $O(kn^2)$ space and because of the non-uniform distribution of

the FilmFinder data, k will be a large value, and it would be a harder task to optimize k for various 2-dimensional ranges even when the distribution is known *a priori*. It is also slower due to the two indirections. So we use the bucket method in preference to the grid method.

2) Rapid Access

In response to a zoom operation the list of visible rectangles is formed and cached into an array so that subsequent operations on other non-zoom attributes of films (like length, category, rating) can access the compact cache. The way this cache is built and updated differs for zoom in and zoom out.

When zooming in is used to reduce clutter. rectangles appearing in successive views are a subset of those appearing in previous views, so that the rectangles for the new view can be plucked from the old list. Therefore zooming in takes place faster when the number of rectangles being displayed on the screen is small because a shorter list has to be searched, and gets sluggish when it is large.

On the other hand, a zoom out operation always increases the number of visible rectangles by expanding a range on one of the axes. This list of rectangles appearing in the new view can be formed in two ways. An obvious method is to form the list of rectangles lying within the newly expanded range and append this list to the old rectangle list. However, since the positions of all rectangles have to be recomputed regardless of whether they appeared in the old list or not, it is cheaper to ignore the old list and build it afresh by scanning over the entire expanded range.

Data access is of the order of O(number of buckets), which equals O(range size) and as zooming is done, the range keeps changing, and so does the rate of data access. The bucket storage method permits access to each range boundary in constant O(1) time.

3) Double Buffering

The new image is drawn onto an off-screen buffer and copied onto the screen in a single operation so that the flicker associated with erasing and redrawing a large number of rectangles is eliminated. Even though double-buffering increases the total amount of time taken to change a range boundary, it makes zooming smoother by reducing flicker.

There are two approaches to drawing new rectangles: they can either be drawn on a fresh off-screen buffer after painting the buffer in the background color or they can be drawn on the old buffer after erasing the old rectangles. Since the FilmFinder rectangles never cover more than 30% of the screen area when fully zoomed out, it is cheaper to erase old rectangles before drawing the new ones than to paint a large fresh buffer in the background color.

4) Increased Axis Resolution

If successive domain data values on either of the ranges are placed too far apart, the positions of successive rectangles change by a large amount, causing their motion to appear jumpy and discontinuous. The motion can be smoothed by having more display locations than data points - for example the y-axis has just 10 discrete values of popularity but 70 steps in which rectangles move. Although this technique does smooth the motion, it increases the total time needed to change a range.

FUTURE WORK

Zooming can be made faster and smoother by extending the above techniques. When the screen has a large number of rectangles, draw just half of them alternately, as done in the X-Window application xgas. The buffering technique can be varied: instead of erasing and redrawing rectangles on the buffer, an updated image of the union of the old and new areas of each rectangle can be copied onto the buffer.

An important challenge is to find an upper limit on the number of rectangles that can be displayed before the illusion of zooming fails, and to get a concrete measure of the maximum speed of the rectangles (say 1 cm. per second) that can be tolerated.

A more challenging problem is to visualize a database containing items of the order of 50,000. Such a visualization can be made by displaying a small number of representative rectangles [8] and zooming in to reveal the hidden ones - using zooming for its traditional purpose of revealing more detail. Handling such huge amounts of data would necessitate the use of linked data structures like k-d trees, range trees or quadtrees [10].

A 3-D display of rectangles might be an appealing alternative, but there's the danger of items obscuring each other and of the user getting lost in the "star-tank".

Although the zoom bar is an adequate tool, an alternative tool like a resizable rectangle roving over a miniature overview of the starfield display would permit both zooming and panning to be done with the same widget.

CONCLUSION

This paper discussed strategies to make incremental zooming appear smooth and flicker-free, presented a taxonomy of zooming methods and introduced the zoom bar, an intuitive and rapid widget to facilitates zooming. Smooth zooming of items in a database

visualization was achieved by reducing the data access and display bottlenecks.

ACKNOWLEDGEMENTS

We thank Christopher Ahlberg for designing and implementing the FilmFinder, and for his suggestions and review of the paper. We thank Bruce Chih-Lung Lin for identifying efficient storage and access techniques, Andries van Dam, John Hughes and Marko Teittinen for helping define zooming with the camera model, and David Mount for helpful suggestions.

Thanks are also due to Richard Chimera for numerous suggestions, to Richard Potter for his review and to Visix Inc. for providing us their cross-platform application builder, "Galaxy". Thanks to Teresa Casey and Ara Kotchian for helping with the images. Finally, we thank the Institute for Systems Research for their support. This research was supported by a grant from the National Science Foundation, grant number NSFD CDR-8803012.

REFERENCES

1. Ahlberg, Christopher and Shneiderman, Ben. Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *Proc. of CHI 1994.* ACM, New York, pp 313-317.

2. Ahlberg, Christopher and Shneiderman, Ben. The AlphaSlider: A Compact and Rapid Selector. *Proc. of CHI 1994.* ACM, New York, pp 365-372.

3. Bederson, Benjamin, Hollan, James. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. Submitted to UIST 1994.

4. Bier, Eric, Stone, Maureen, Fishkin, Ken, Buxton William, Baudel Thomas. A Taxonomy of See-Through Tools. *Proc. of CHI 1994.* ACM, New York, 1994, pp 306-312.

5. Carr, David, Jog, Ninad, Kumar, Harsha, Teittennen, Marko, Chimera, Rick and Ahlberg, Christopher. The HCIL Widgets: Motivation, Specification, Development and Use. *CAR-TR-xxx* University of Maryland, College Park 1994.

6. Foley, James, van Dam, Andries, Feiner, Steven and Hughes, John. *Computer Graphics, Principles and Practice.* 2nd edition, Addison Wesley, Reading, MA 1990.

7. Jain, Vinit and Shneiderman, Ben. Data Structures for Dynamic Queries: An Analytical and Experimental Evaluation. *CAR-TR-685,* University of Maryland, College Park 1993.

8. Keim, Daniel, Kriegel, Hans-Peter and Seidl, Thomas. Visual Feedback in Querying Large Databases. *Proc. of IEEE Visualization 1993,* pp 158-165.

9. Robertson, George, Card, Stuart and Mackinlay, Jock. Information Visualization Using 3-D Interactive Animation. *Communications of the ACM, 36,* Number 4, April 1993, pp 57-71.

10. Samet, Hanan. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, MA 1990.

11. Sarkar, Manojit and Brown, Marc. Graphical Fisheye Views of Graphs. *Proceedings of CHI 1992,* pp 83-91.

12. Schaeffer, Doug, Zuo, Zhengping, Bartram, Lyn, Dill, John, Dubs, Shell, Greenberg, Saul, Roseman, Mark. Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchially Clustered Networks. *Research Report # 92/491/29,* University of Calgary, November 1992.

13. Shneiderman, Ben. Dynamic Queries for Visual Information Seeking: A Step Beyond Database Languages. *IEEE Software, 1994.*