UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO RONCONI RICHTER

Combining Fisheye Lens and Dimensional Stacking to Visualize High Dimensional Data

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. João Luiz Dihl Comba

Porto Alegre
December 2018

“An investment in knowledge always pays the best interest.”

— BENJAMIN FRANKLIN

ACKNOWLEDGMENTS

I would like to thank my family for the given opportunity to pursue my interests and dreams.

To my friends, who were always there for me, no matter the hardship, and always supported and encouraged me.

To professor João Comba, who guided, helped and stayed with me through this intense period of learning and work, and to professor Filip Sadlo, who contributed with essential information and ideas.

Finally, to all people involved in the research in which this work is based on.

ABSTRACT

As the amount of data stored throughout the world increases, the need to process, analyze and visualize it increases as well. There are many challenges to developing techniques for visualizing large quantities of data, specially in ways that are intuitive and allow for real time interactions. An existing technique to display multivariate data in a 2D space is called dimensional stacking. It maps multiple dimensions inside one another, in a discrete 2D plot. This technique, however, generates an image that increases dramatically in resolution as the number of dimension increases. This limits the number of dimensions plotted concurrently. While it works for a limited dimension amount, it becomes ineffective for more dimensions.

In this work, we propose a distortion technique for an image generated by the dimensional stacking, in order to solve the dimension limitation problem. By combining a fisheye lens distortion technique, the dimensional stacking can be used for visualization of data sets with a higher number of dimensions that would be possible without any distortion, or with traditional operations, such as panning, cropping and zooming.

We describe the process to generate both techniques and how to combine them, and then implement it in a web application that allows loading multiple datasets, has real-time interactions and parameter configuration. To evaluate the effectiveness of the application, we test and analyze different datasets. Then, we compare how well the application performed in relation to the properties of the datasets, such as the number of data instances and the number of dimensions.

Keywords: Dimensional stacking. fisheye lens. visualization tool. rubber sheet stretching. cartesian fisheye.

Combinando Lente Fisheye e Empilhamento Dimensional para visualizar dados de alta dimensão

RESUMO

Conforme a quantidade de dados armazenados pelo mundo cresce, a necessidade de processar, analisar e visualizá-los também amplia. Existem diversos desafios para desenvolver técnicas para visualizar grande quantide de dados, especialmente de formas que são intuitivas e permitem interações em tempo real. Uma técnica existente para exibir dados multivariados em um espaço bidimensional é chamada de empilhamento dimensional. Ela mapeia múltiplas dimensões, dentro delas mesmas, em um plano 2D discreto. Essa técnica, porém, gera uma imagem que aumenta de resolução dramaticamente conforme o número de dimensões aumenta. Isso limita o número de dimensões traçadas concorrentemente. Enquanto ela funciona para um número limitado de dimensões, ela se torna ineficiente para mais dimensões.

Nesse trabalho, é proposto que seja aplicado uma técnica de distorção em uma imagem gerada pelo empilhamento dimensional, com o objetivo de resolver o problema da limitação de dimensões. Através da combinação da técnica de distorção da lente *fisheye*, o empilhamento dimensional pode ser usado para a a visualização de conjuntos de dados com um número mais elevado de dimensões do que seria possível sem nenhuma distorção, ou com operações tradicionais, tais como deslocamento, corte e amplificação.

Nós descrevemos o processo para gerar ambas técnicas e como combiná-las, e então as implementamos em uma aplicação web que posibilita carregamento de múltiplos conjuntos de dados e possui interações em tempo real e configuração de parâmetros. Para avaliar a efetividade da aplicação, são conduzidos testes e analise utilizando diferentes conjuntos de dados. Então, é comparado como a aplicação se desempenhou com relação às propriedades dos conjuntos de dados, como o número de instâncias e o número de dimensões.

Palavras-chave: empilhamento dimensional, lente fisheye, ferramenta de visualização, pano de borracha, fisheye cartesiano.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

DS	Dimensional Stacking
FL	Fisheye Lens
ABV	Alcohol by volume

CONTENTS

1 INTRODUCTION

The amount of data being stored is increasing dramatically over the years, along with the need to process and interpret it (JACOBS, 2009). The usage of data visualization methods for the objective of mining large amounts of data is increasing, being used not only for displaying results made by previous computations, but also in the process of cleaning irrelevant, incorrect or corrupt data from a given dataset (GRINSTEIN; TRUTSCHL; CVEK, 2001).

When it is needed to visualize and understand large amounts of data, such as multivariate datasets with many dimensions, it is often difficult to do it in a way that is easy and intuitive for a person. Traditional tabular methods are many times used by general purpose softwares for data visualization. These methods, however, show data in simple numerical form inside tables and multiple tabs. With large multivariate datasets, it becomes difficult to find relevant data, discover patterns, filter data and navigate through it.

Many techniques were developed to display multidimensional data inside a 2D space, such as the dimensional stacking (LEBLANC; WARD; WITTELS, 1990), that allows the visualization of multiple dimensions at the same time. One property of this method is that as the number of dimensions of the visualization increases, it quickly becomes difficult to see them, due to the exponential growth of lines created, reducing the size of each data element to the degree that makes it invisible to the naked eye.

Navigating through large amounts of data is a challenge as well. When scrolling through a very large table, a user can easily go to other parts of it, but the precision to find a specific location may be lacking. When panning, the user might feel disoriented and can lose track of his position, or of other parts he or she may be interested in.

An existing technique for data navigation is the Fisheye Lens. Given an image created by some visualization method after processing a dataset, the fisheye lens creates a main, "focus" region, in which a detailed view of the data is shown, as well as an outside region of context, that display only a condensed version of the data. The data inside the outer region cannot be seen in its entire detail, but it provides the user a clue for what kind of information might be present on the periphery of the central region. The fisheye lens dynamically change the position and size of the blocks and lines generated by the dimensional stacking, reducing the size of blocks farther from the focus region and increasing the size of blocks closer to it. If using the proper variation of the fisheye lens, the 2d properties of the dimensional stacking image are kept, while also enhancing the ability to see a specific region, and a higher number of dimensions inside it.

This work suggests, describes and implements a combination of the dimensional stacking visualization technique with a variation of the fisheye lens distortion technique. The fisheye lens allows the visualization of a dimensional stacking image with a number of dimensions that it would otherwise be unfeasible.

2 RELATED WORK

Tools with the purpose to visualize and navigate through data can be seen as a combination of two components: the *representation* of data and the *presentation* of data. *Representation* refers to the way a dataset is mapped to visual form inside the tool's screen. *Presentation* is how the representation of the data is manipulated by the user and how the visualization behaves, emphasizes different parts and organize itself. A simple example is a 2D plot for a sine wave. The representation is the lines of the cartesian plan and the sine function being displayed visually. The presentation is the size of the time region that is being displayed, the axes legends, the thickness of the line, the interactions the user may be able to perform, zooming in and out of the plan, or cropping specific ranges of the axes. (CARPENDALE; MONTAGNESE, 2001)

Many visualization tools have been created for low dimensional data (2D or 3D), such as data of physical phenomena (WARD, 1994). Higher dimensional data plotted with these tools may cause overlap of information in the same region, creating cluttering and issues with understanding the visualization. Multiple plotting techniques (WONG; BERGERON, 1994) were developed to map multivariate data to bivariate space such as scatterplots (HANNA; RAO; ATHANASIOU, 2010), parallel coordinates (INSELBERG, 1990), hierachical axis (MIHALISIN GAWLINSKI, 1990), dimensional stacking (LEBLANC; WARD; WITTELS, 1990) and brushing (BECKER; CLEVELAND, 1987).

Given that screen size is many times a constraint in the process of visualizing large scale information, distortion-oriented presentation techniques have been developed aimed to distort not only the view in order to expand a small area and analyze it in full detail, but also to see the whole visualization around the focused area, giving the user a global or local context of the information.

There are different existing approaches aiming to distort the information space in context and focus regions to allow viewing the whole space, and simultaneously some smaller region in more detail. This chapter introduces these techniques, as well as other important concepts related to this project.

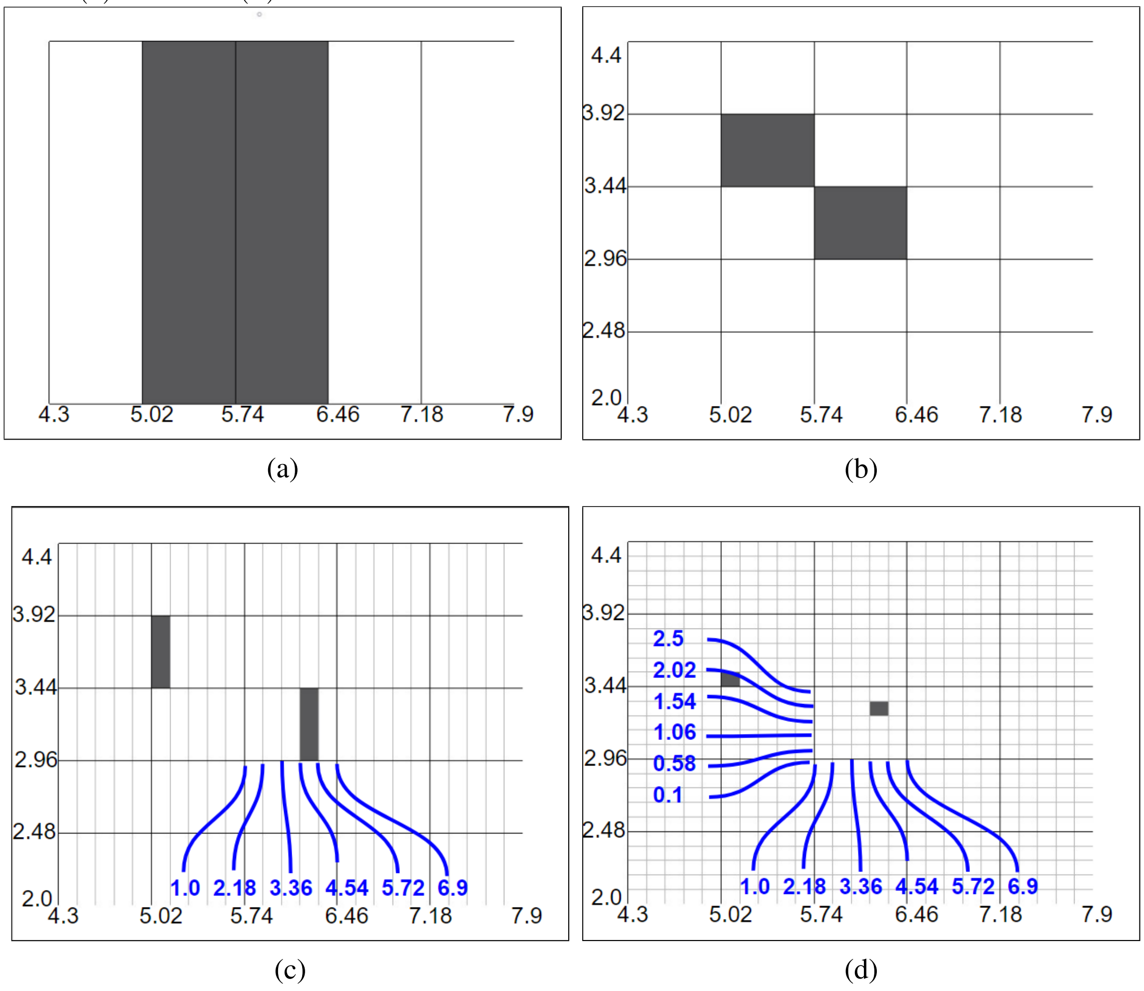
2.1 Dimensional Stacking

Dimensional stacking was initially suggested by (LEBLANC; WARD; WITTELS, 1990) as a way to expand traditional 2D plotting to accommodate potentially any number of dimensions in a hierachical form. It works by discretizing the 2D plot, generating rectangles instead of points for a coordinate, then recursively creating new 2D discrete plots inside each rectangle, for each two more dimensions. The usefulness of this technique depends heavily on the first dimensions selected (KEIM, 2002). If the elements in the first two dimensions are evenly distributed, most of the screen will be populated. Otherwise, the data will be sparse and not make an appropriate use of screen space. In an implementation of this technique, it is therefore interesting to switch the order of dimensions in real-time, to facilitate the selection of the outer (first) two dimensions.

To construct a dimensional stacking visualization, we select the first dimension D_1 of N dimensions $< D_1, D_2, D_3, ..., D_n >$. The order of these dimensions will impact the final position on the visualization. Usually the first dimensions are the "slowest" ones, that is, the dimensions in which information values change the least in each row. D_1 is discretized in B buckets, or sections. Given the screen *width*, we map each section to an x value of $x = i * (width/B)$, with i being the section index within range $[0, B - 1]$, effectively splitting the screen many times vertically. For each value of D_1, we "put in the bucket", that is, create a rectangle that fills the corresponding bucket, given the value discretized in $[0, B - 1]$. Then we select the next dimension, D_2, and split the screen horizontally, splitting each existing bucket in B buckets, generating in total B^2 buckets. We select each value in D_2, discretize it and put in the corresponding bucket, this time in the vertical orientation. A bucket will only be "filled", or painted, if there is at least an entry for both dimensions D_1, D_2 at that bucket. The visualization at this stage will look similar as a regular 2D Cartesian plan, but with rectangles instead of points. Each rectangle (a filled bucket) now serves as a new Cartesian plan for more dimensions. We recursively apply the same steps as before, but for every bucket, instead of the whole screen (LEBLANC; WARD; WITTELS, 1990). This method can be much more easily understood by Figure 2.1 that shows each step in the creation of a dimensional stacking visualization. To account for overlapping of data in the same bucket, we can map the amount of data instances in the same bucket to a color map, and "paint" it.

Note that the first dimension is arbitrarily placed at either vertical or horizontal orientation. The number of buckets reflects the granularity, or dicretization level that one might want from the data. As the number of buckets increases, the more detail there is and difference between values in the same dimension can be seen more easily. For example, if most values in a dimension range from 20 to 30, but there is one value of 200, eight buckets would result in most values being put in the same bucket, making the

Figure 2.1: Examples of the Dimensional Stacking visualization with one (a), two (b), three (c) and four (d) dimensions.



Source: The author, using XmdvTool (WARD, 1994)

difference between them unnoticeable. If the visualization also colors the buckets, the user can notice that there are many values inside a single bucket, and then he or she can increase the number of buckets until these values ranging from 20 to 30 are put in more than one bucket.

2.2 Distortion-Oriented Presentation Techniques

This kind of technique aims to display local detail in some focal region of the visualization, and outside of that region, display it in less magnitude. Given a representation of data, called a visualization, a transformation function is applied to generate a distorted view of this visualization. This corresponds to an uneven mapping of the representation to a new image of different size and aspect-ratio. A magnification function corresponds to an overview of the amount of magnification in each point of the original visualization, and is the derivative of the transformation function. The user can move the focal region to another part of the visualization, and the transformation function, which depends on this focal region, will be applied to the visualization. This will make the new focused region magnified, and the adjacent regions will be demagnified accordingly. This process occurs in real time for the user, and its performance is influenced by the complexity of the transformations, the amount of information being presented and the speed of the platform in which the technique was implemented(LEUNG; APPERLEY, 1994).

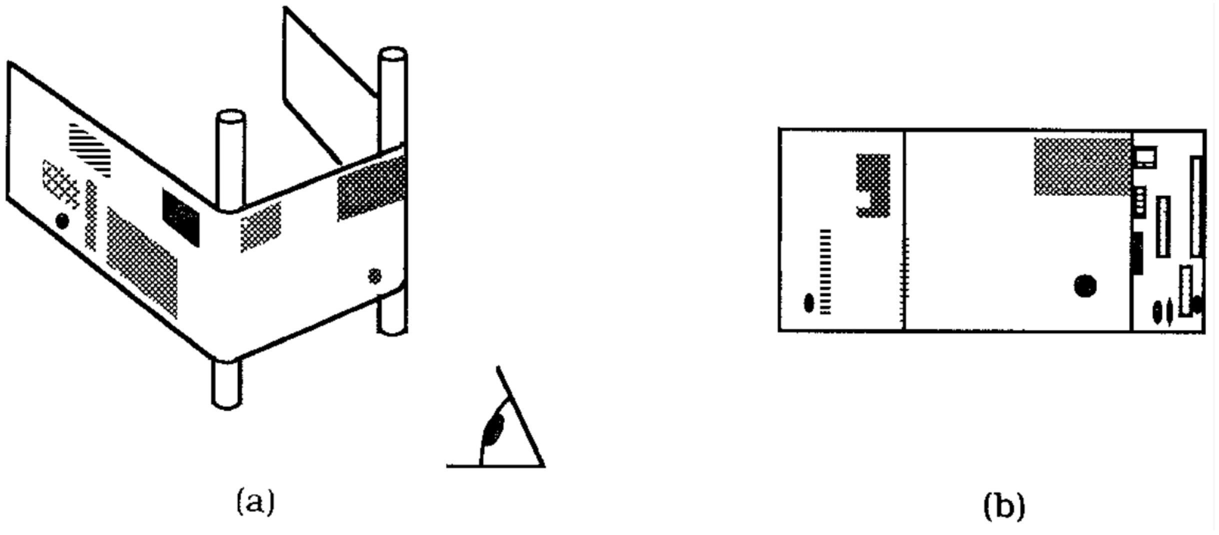
2.3 The Bifocal Display

If we imagine a sheet representing information that is larger than the screen that is used to view it, it would be necessary to scroll through the sheet to examine all the information space. The Bifocal display is a one-dimensional distortion technique that suggests "wrapping" the sheet around two poles, bending it. With an appropriate angle, a user would then be able to see the region between the two poles in full detail and no distortion, and also the whole sheet, but "pushed" to the sides. Even though it will not be possible to clearly visualize information in this distorted region, the user can recognize the presence of information there, or discern which kind of information is there. After noticing a pattern or presence of information on the distorted corners, the user is able to move that item to the region between the two poles to then visualize it in full detail (APPERLEY; SPENCE, 2013). In a visualization with the Bifocal display, the position of an item in the distorted portions of the view does not influence the amount of distortion applied in relation to its neighbors. Items closer to the edges of the screen will be distorted in the same amount as items farther away from the corner, as long as these other items are inside the distorted portion of the view.

2.4 The Perspective Wall

The Perspective Wall was developed to create a different way to display 2D information, integrating detailed views and contextual views in a smooth way. This technique transforms the 2D layout of the bifocal display to create a 3D perspective, creating a wall (MACKINLAY; ROBERTSON; CARD, 1991). Three panels combined compose

Figure 2.2: (a) The original physical representation of the bifocal display; (b) The view generated by the model.



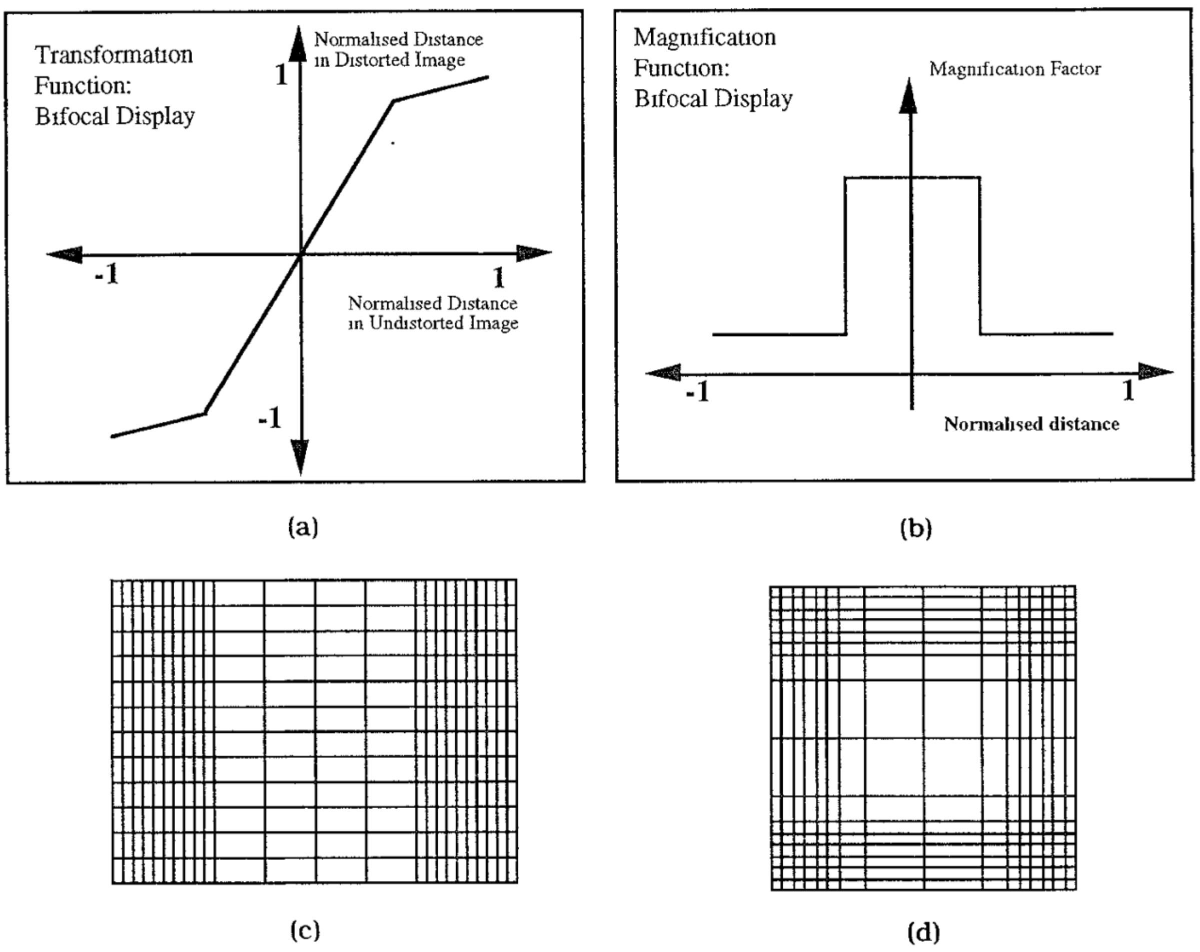
Source: (LEUNG; APPERLEY, 1994)

this wall, creating the illusion of a pentagon shaped box being seen from a side view. One center panel shows detailed information in 2D of a specific portion of the data space, and two lateral panels touch the center panel in an angle, creating a 3D visualization. The lateral, or perspective panels, display the data space in less detail.

A positive characteristic of the Perspective Wall is that when the information is scrolled sideways, there is a smooth transition of the visualization between planes. This allows the user to not lose track of objects being distorted and moved. Displayed elements can be seen in real time bending around the edges of the planes. Another important feature is that the user can change the size of the detailed view (center plane) and the context views (side planes). When there is a lot of information on the detailed view, the user can increase its size to more easily see the information, and when he or she wants to see more context in order to navigate through, the context view can be increased (causing the detailed view to decrease in size). In Figure 2.4, it is clear that the information, originally a 2D information space, still retains its shape in the center plane, while benefiting from the advantages of the Perspective Wall.

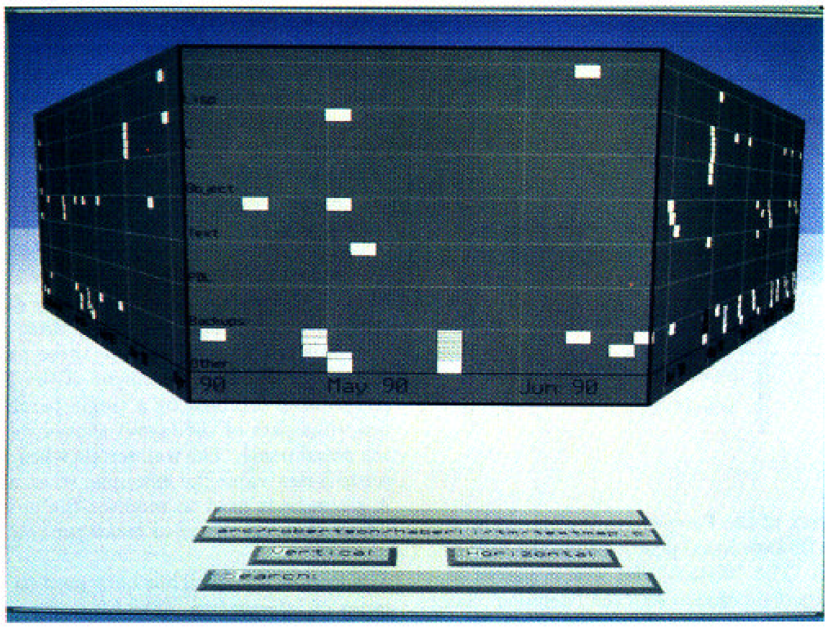
The implementation works by passing three times through the list of information (2D vectors and text with 2D coordinates) and drawing the three planes.

Figure 2.3: The Bifocal Display's (a) transformation function and its (b) corresponding magnification function, as well as its application on (c) one and (d) two dimensions.

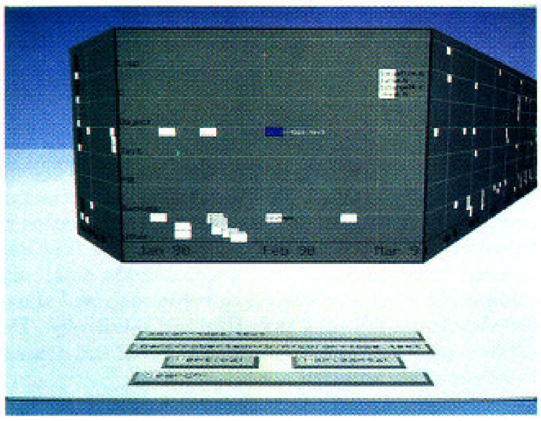

Transformation Function: Bifocal Display
Normalised Distance in Distorted Image
1
-1
1
Normalised Distance in Undistorted Image
-1
Magnification Function: Bifocal Display
Magnification Factor
-1
1
Normalised distance
(a)
(b)
(c)
(d)

Source: (LEUNG; APPERLEY, 1994)

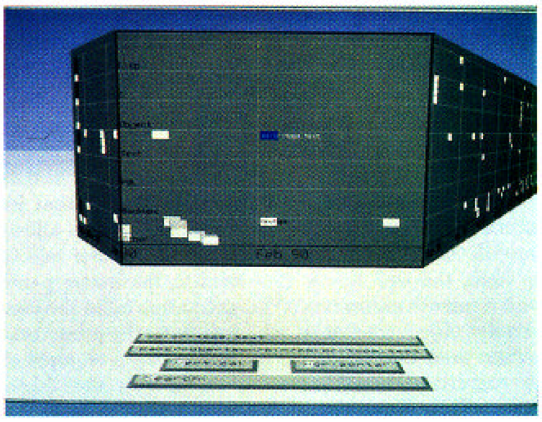
Figure 2.4: The perspective wall being focused in three different locations.



Mackinlay Plate 1



Mackinlay Plate 2



Mackinlay Plate 3

Source: (MACKINLAY; ROBERTSON; CARD, 1991)

2.5 Fisheye Lens

This technique was created as a different approach to show files to 2D structures with a "fisheye lens", that displays things near the center of the view with high detail and magnitude, while also displaying the whole structure, but with less magnitude as the distance from the center increases. A simple fisheye model is proposed by (FURNAS, 1981). To create this structure, three properties are necessary: a focal point, distances from the point to any other region, and the amount of detail. This model assumes the interaction occurs with a single point in the structure, called the focal point. There has to be clear defined distances between the focal point and any other point on the structure. For every point, there has to be a level of detail, LOD(x). This refers to a degree of generality, or approximation of the information at that point. For example, in a mapping software, a straight highway may be displayed on the view. When zooming in, it is revealed that the highway is in reality slightly curved at that point.

With these three components, FURNAS defined a function associated with a fisheye view, called Degree of Interest. Given a current point, let

$$DOI(x|.) = F(LOD(x), D(.,x))$$

where:

".": focal point,

LOD(x): level of detail of point x,

D(.,x): distance between focal point "." and x.

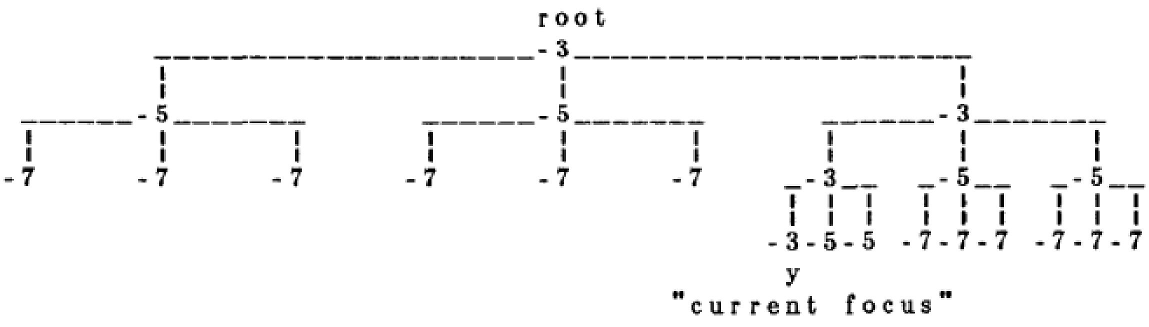
Finally, *x* is only displayed if it is above a threshold *k*.

2.6 Graphical Fisheye Lens

To generate a fisheye view graphically, every vertex's position is distorted based on the distance to a focal point and its original position. The vertexes of lower interest, that is, the ones farthest away from the focal point, will be "pushed" to the corners of the screen, leaving more space for vertexes closer to the focal point, which will be magnified.

For each vertex v: the coordinates of v in the fisheye view is a function of its oroginal coordinates and the coordinates of the focal point, the size in the fisheye view depends on its original size, coordinates, the focal point coordinates and an assigned

Figure 2.5: A fisheye model for trees. The Degree of Interest is greater in regions closer to the focus point.



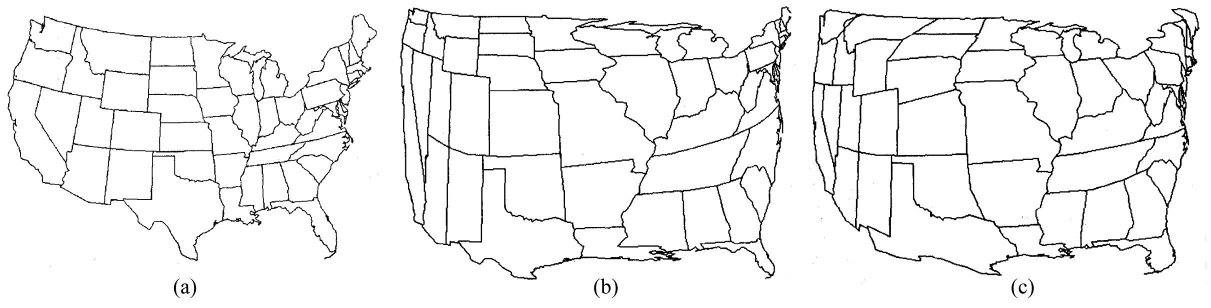
Source: (FURNAS, 1986)

value called API. This value describes its importance in relation to other vertexes in the visualization.

In the process of generating this visualization, a transformation function is applied to the vertexes, mapping their new coordinates in a way that magnifies and demagnifies them, according to their distance from the focal point. If the transformation function handles x and y axes independently, the fisheye view will preserve vertical and horizontal differences between vertexes. If two vertexes v_1 and v_2 have the same value on the y axis in the original view, the application of the transformation function will change their value, but they will remain equal. This is useful for tabular views, because it will not throw off their relative distances and impact the easiness of understanding of the view. This kind of transformation function that handles axes independently is called a *cartesian transformation*. When the transformation function maps the vertexes coordinates depending on both x and y at the same time, it generates a view closer to the "natural" interpretation of the fisheye lens. It creates a sphere-like illusion of the original view using the polar coordinate system (SARKAR; BROWN, 1992). This transformation function is called the *polar transformation*.

When handling some types of graphical views it can make more sense to use the polar transformation. For example, a geographical map of a country with the application of a cartesian transformation might massively distort the x axis while slightly distorting the y axis, for a particular fisheye view. This will distort the original shape of some state so much that the state's shape in the fisheye view will be barely recognizable. A polar transformation with the same magnification intensity will still distort massively the state, but not as much one axis in relation to the other, since a position of a vertex v in the

Figure 2.6: The United States map in its (a) regular form, with (b) cartesian transformation and (c) polar transformation.



Source: (SARKAR; BROWN, 1992)

polar transformation depends on both the x and y axis. This way, the state's size will be changed, but its shape, not as much, retaining a user's ability to recognize it at a glance.

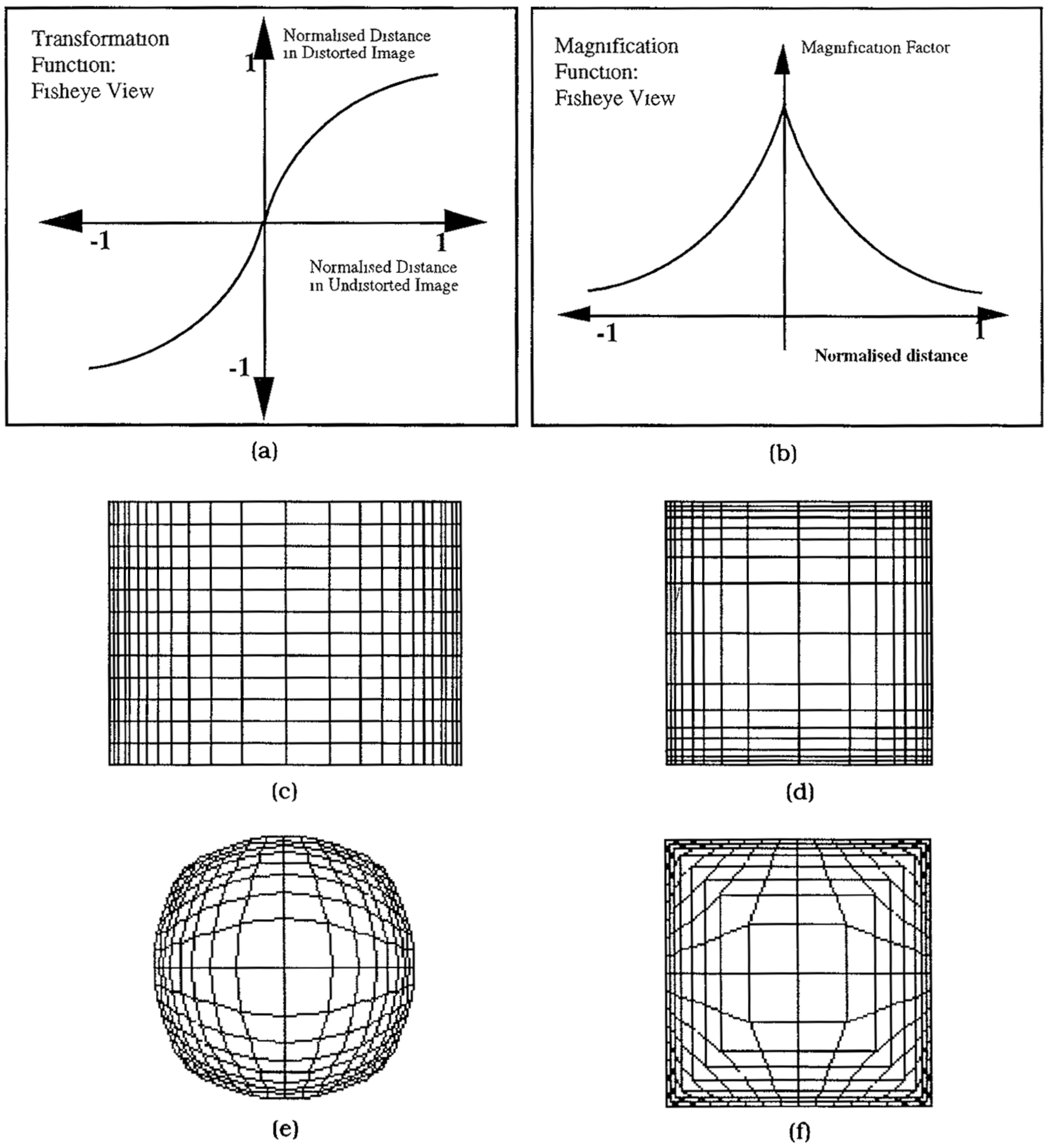
2.7 Rubber Sheet Stretching

This distortion technique is based on the metaphor of the view being a rubber sheet mounted on a frame. By pushing a ball behind the sheet towards it, the area on contact with the ball gets stretched, magnifying the region and exposing a greater level of detail. The analogy of the rubber sheet is in fact not enough, because it has to behave not exactly as an elastic material but to perform also in ways that does not reflect the physical world (JR; GRIFFIN, 1985).

This method has many benefits (SARKAR et al., 1993):

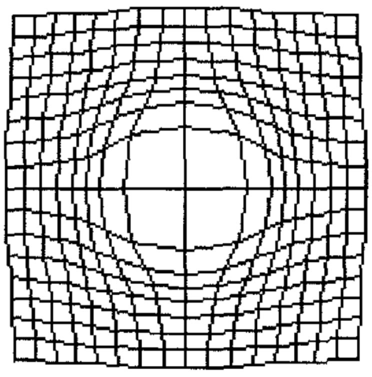
1. The sheet is entirely seen at the same time, providing context.
2. It provides integrated context and focus, since the most distorted area is the focus, and the adjacent regions get less distorted the farther they are located.
3. It is possible to precisely adjust the size of the focus region by changing the size of the ball. This way, the user can select any region of any size to see in more detail.
4. The user can use multiple balls, enabling him or her to select multiple regions of focus at the same time.
5. The only distorted region is the focus region and the context region maintain its original shape. This method makes it clearer for the user to see the referential location and shape of the overall map. Additionally, by reducing the size of the ball, the user can see the entire map in its original form, effectively removing any

Figure 2.7: (a) A traditional transformation function for the fisheye lens; (b) its magnification function; (c) the cartesian fisheye lens application in one dimension; (d) in two dimensions; (e) a polar fisheye lens application; (f) a normalized polar fisheye lens.


Transformation Function: Fisheye View
Normalised Distance in Distorted Image
Normalised Distance in Undistorted Image
1
-1
1
-1
(a)
Magnification Function: Fisheye View
Magnification Factor
-1
1
Normalised distance
(b)
(c)
(d)
(e)
(f)

Source: (LEUNG; APPERLEY, 1994)

Figure 2.8: The application of a rubber sheet distortion on a 2D grid.



Source: (LEUNG; APPERLEY, 1994)

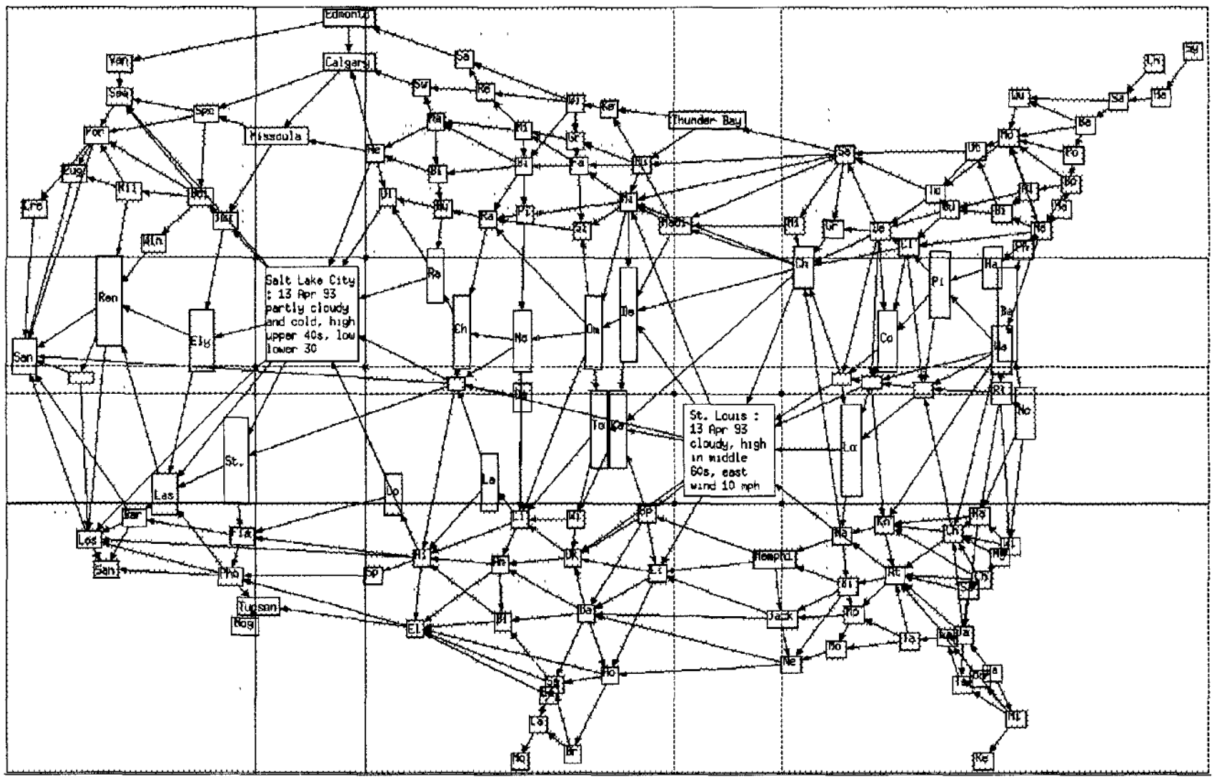
distortion from the view, and then increase again the size of the ball afterwards.

The most simple implementation is the *orthogonal stretching*, that works by placing two horizontal lines and two vertical lines, called *handles*, on top of the visualization. The rectangle formed by these lines represents the lens. When the handles are dragged across the view, the region between the vertical handles get distorted along the x axis and the region between the horizontal handles get distorted along the y axis. The region inside the rectangle is distorted by both axis. Regions outside of the handles return to their original shape and size. In the region between the handles, there are *vertical stretch factor* and a *horizontal stretch factor* associated with each axis. Each point in the original view is mapped to a new distorted point according to the stretch factors.

The intersection point between a horizontal handle and a vertical handle can be "glued together" to preserve the shape of the lens. This feature is called *clamping*, and can be imagined by nailing a pin on the intersection point, attaching the two handles. One disadvantage of orthogonal stretching is that when two handles create a distortion on an axis, the entire region is distorted, and not only the desired rectangle. In Figure 2.9 , we can see that the not only the rectangles are distorted, but the set of rows and columns in it are distorted across the whole view. The orthogonal stretching preserves symmetry and therefore can be more appropriate for layouts with elements in the shape of rows and columns, such as grids, circuits, matrices and spreadsheets (SARKAR et al., 1993).

Another way to view the rubber sheet technique is the *polygonal stretching*. Instead of defining four handles to create a lens, it is created a polygon to represent it. This polygon works as the handle and only the region inside it is distorted.

Figure 2.9: The orthogonal rubber sheet stretching containing two lenses distorting a map of the United States.


Edmonton
Calgary
Van
Sea
Spo
Missoula
Thunder Bay
Salt Lake City : 13 Apr 93 partly cloudy and cold, high upper 40s, low lower 30
St. Louis : 13 Apr 93 cloudy, high in middle 60s, east wind 10 mph
Ren
Las
St.
Los
San
Tucson
Memphis
Jack

Source: (SARKAR et al., 1993)

3 DEVELOPING THE APPLICATION

Although the dimensional stacking technique works for data with more than two dimensions, it has a limitation when the number of dimensions increases beyond single digits, at most. Given the number of dimensions d and the amount of buckets s The total number of columns or rows used to separate blocks is d^s. In a typical visualization in which $d = 6$ and $s = 7$, there will be $2 * (6^7) = 559,872$ lines that are needed to be drawn in the screen. The Table 3.1 shows just how fast the number of lines in the view can increase. Even with a relatively low configuration of $d = 4, s = 6$, the number of lines generated is 4096. For a relatively small screen size of $640px$, the columns and rows would cover the screen entirely, disrupting the visualization. Furthermore, the size of the blocks being put inside the buckets will be less than $1px$. There is always a need to compromise between the number of dimensions and buckets.

One solution adopted is simply not displaying any column or row (JIANG et al., 2016). If s is sufficiently low, it is easy to differentiate between adjacent blocks, and lines separating them may be not necessary. Another possibility is to combine pixelization with dimensional stacking, so blocks are displayed as a single colored pixel, and the separation lines are again not drawn (LANGTON; PRINZ; HICKEY, 2006). These methods do not provide a generic solution to the increasing number of lines problem, and depend on the dataset being used. For higher values of s, adjacent blocks with similar color will appear to be one single block. The separation lines also serve to tell which dimensions each block belongs to, and removing them causes confusion in figuring out what is the values in each dimension for a given block. This also does not solve the issue of blocks being invisibly small. If we limit the minimum size of blocks to $1px$ with pixelization, then the image resolution will start to grow.

The solution proposed in this work is to combine the fisheye lens with a dimensional stacking visualization. First, we generate a dimensional stacking visualization, and apply a fisheye lens distortion on top of it. This view is encapsulated in a web application with real time interaction. The cartesian fisheye lens is the lens used, and every frame,

Table 3.1: Total number of lines created with the dimensional stacking technique given a series of d and s values.

Dimensions	2	2	2	4	4	4	6	6	6
Buckets	3	6	9	3	6	9	3	6	9
Lines	8	64	512	64	4096	262144	216	46656	10077696

Source: The author.

for each object in the generated DS visualization, we apply the fisheye lens function to distort its position and shape. Blocks with distance from the focus point that are less than a threshold k become semi transparent and expose a new level of the DS visualization for two more dimensions. When the focus point moves around the view, as blocks enter and exit the threshold, they dynamically become transparent or solid, showing or hiding inner two dimensions.

3.1 Generating The Dimensional Stacking Visualization

The first stage is to create a traditional dimensional stacking visualization. It is composed of columns and rows of different *line-widths* and colored rectangles. Here it will be described the process for N dimensions and S buckets, with a screen of resolution of $width$ and $height$.

3.1.1 Columns and Rows

To create the buckets, we need to partition the screen in equal slices. First, we arbitrarily choose that the axis in which the first dimension will be put is the x axis. For each dimension n in range $[0, N-1]$, we partition the screen S^{n+1} times. Starting with dimension $n = 0$, we create $S+1$ columns, from position 0 to $width$. $S-1$ columns are for partitioning the screen in equal spaces, and 2 are for the borders of the grid. Each column is spaced by $width/S$ pixels. For $n = 1$, we repeat the process, but now at the y axis, creating $S+1$ rows spaced by $height/S$ pixels. For $n = 2$, we need to create S buckets for every existing bucket, therefore we need to generate $(S^n)+1$ new columns spaced by $width/(S^n)$ pixels. Note that S new columns generated will overlap with previously drawn columns. For $n = 3$, we generate $S^{n-1}+1$ rows spaced by $height/(S^{n-1})$ pixels. Since we started on the x axis, every even n represents a vertical split and we create S^n+1 columns, and every odd n, a horizontal split, and we create the same amount of rows as the previous dimension created columns, that is, $S^{n-1}+1$. We then repeat this for every n until $n = N$. In the end of this process, the total number of lines created is the sum of the number of lines created in each step, from $n = 0$ to $n = N$.

For increased differentiation between which dimension each line represents, the $line-width$ of each row and column varies. The lines in the first outer dimensions are thicker, and get more thinner and in lighter grey for more inner dimensions.

3.1.2 Blocks

A colored rectangle in the DS view represents an instance of data that is located at that coordinate, for every dimension accounted. These rectangles will be called blocks. To place the blocks in the correct buckets, we need to analyze the range of the values of every dimension in the dataset. Given some dimension n, its domain is the range of values from the smallest value to the the biggest value of data instances in n. For every data instance, we analyze the value in each dimension and discretize it accordingly to its dimension domain and the value of S. For example, if the domain of n is $[20, 97]$ and $S = 3$, a data instance of value of 25 in n will be put in the first bucket, and one with value of 80 will be put in the third bucket. If we take into consideration all dimensions, S and a data instance, we can draw a block in the exact bucket.

To discover the exact position to draw the blocks and their size, it is necessary to know the x, y offset, the width and height of a block. The width of a block is exactly the distance between two adjacent columns and the height is the distance between two adjacent rows. Using formulas used to generate the columns and rows, the height of a block is $height/(S^{n-1})$ and the width is $width/(S^n)$. To discover the x position of a block, for every even dimension, we multiply the width of a block in that dimension by the index of the bucket in that dimension. The total sum of these multiplications will be the final x position of the block for a particular data instance. If we execute this process accounting only for odd dimensions, we get the y position.

There is a possibility, and almost certainty, that different data instances will produce blocks in the exact same position. To account for this, we paint the blocks using a color scale.

3.2 Applying the Fisheye Lens Distortion

After the DS view is finished, the position and size of lines, columns and blocks will not change. These values are fixed, and the distortion occurs by applying a function to each object to display the distorted object in the view, but never modifying the values originally generated by the DS algorithm. The fisheye lens is the distortion technique that is used in this work. Instead of drawing the DS view in its original form, a fisheye lens is applied and dynamically distorts objects and display them. For each object, the coordinates x, y of the blocks and $x1, x2, y1, y2$ of lines will be passed through functions $xScale$ and $yScale$ that will receive the coordinate and apply a fisheye distortion.

The fisheye lens expands the size of blocks inside a specific region, allowing more dimensions to be displayed there, and contracts blocks around the region. These blocks will have a reduced size, and that makes it impossible to visualize them clearly. The user just needs to see that there are blocks, and not necessarily their properties, because if the user knows of the presence of nearby blocks, then he or she can just change the focus region to the region containing these blocks. This provides integrated focus and context in the dimensional stacking visualization.

To create a fisheye lens, it is necessary to apply a fisheye scale to each element of the visualization. A scale is a mapping of the axis between the original view and the new distorted view. This mapping takes into consideration the domain of the view and the range of the scaled values. If, for example, it is wanted to map the whole screen in the y axis to only the top half of the screen, the fisheye scale would have the domain of $[0, height]$ and range of $[0, height/2]$. Multiple scales can be used for different parts of the view, and used independently for each axis. These scales utilizes the DOI function as defined by FURNAS.

The algorithm to generate the fisheye scales used in this work is from a d3 plugin (FISHEYE..., 2018), that takes a domain, a range and a distance factor, and generates a fisheye scale.

This work implements two fisheye lenses, the single point fisheye and the rubber sheet stretching, and uses them as a foundation to generate a new variation, called multi-lens fisheye, that it is the actual lens implemented in the final application.

3.2.1 Single Point Fisheye

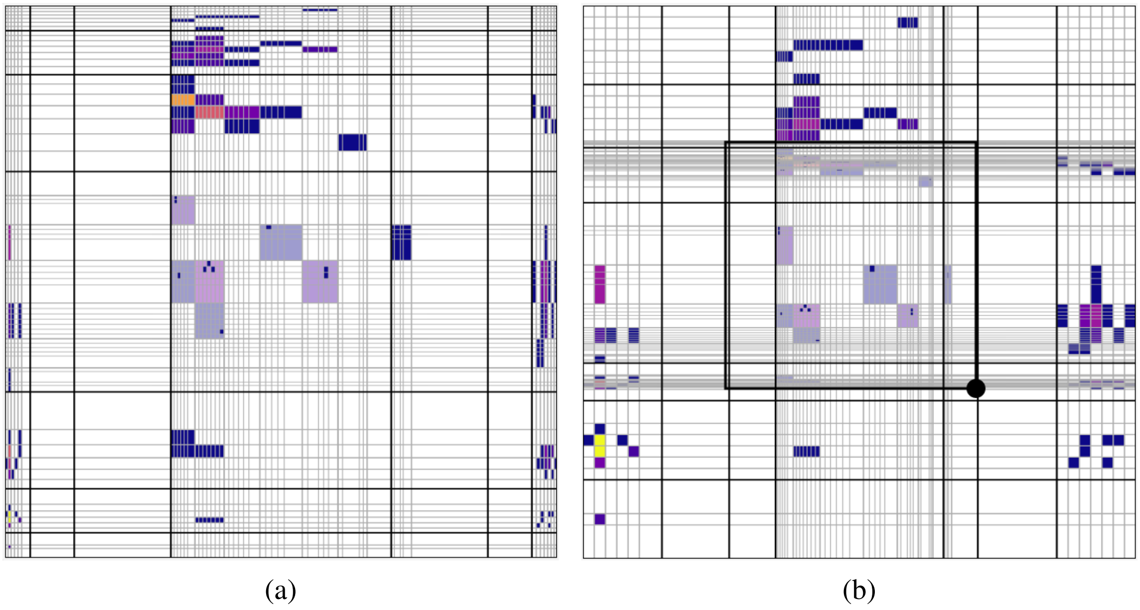
The first fisheye lens developed is the lens in which the focus region is a single point in the view. This point is a 2D coordinate limited by the screen borders. Each line, row and block depends on this point for its coordinates and opacity. We create independent fisheye scales for the x and y axis. We will define a single scale for the entire view, for each axis. The scales map the entire view region for the entire view region. This means that the axes of elements mapped with the scales will be contained inside the view. A column of $x = 0$ or $x = width$ will have the same value after the mapping with the scale, but any different value will be changed. These scales, called $xFishScale$ and $yFishScale$, will distort differently the visualization according to the distortion factor and the focus point.

The focus point can be moved by dragging the screen. As the user drags the screen, the focus point moves accordingly, updating the scales. Then, the columns, rows and blocks have their position and size adjusted in real time by the scales. This adjustment occurs using $d3$ to set their new position and size, taking their original values (the ones generated by the dimensional stacking) and passing through the corresponding scale. The distortion factor influences how strong is the distortion around the focus point. The bigger it is, the more "pushing" of the adjacent elements around the focus point to the sides will occur. By scrolling in and out, the user changes the distortion factor and updates the visualization in real time, causing the effect of zooming in and out.

3.2.2 Rubber Sheet Stretching

Another fisheye distortion technique implemented in this work is the rubber sheet stretching. Instead of using a single point as the focus region, we define two horizontal and two vertical lines in the view and apply the fisheye distortion only inside the regions between the lines. This effectively creates a rectangular fisheye lens in the visualization. The development of this distortion was done by deriving from the single point fisheye. The rubber sheet stretching can be imagined as a single focus point, but not all elements being distorted, only the ones inside a defined range. The $xFishScale$ and $yFishScale$ scales are defined only for a limited domain and range. The two vertical lines chosen represent the domain and range for the $xFishScale$ scale, and the two horizontal lines, the $yFishScale$ scale. The elements of the visualization that are outside of the boundaries of the rectangular lens are not distorted whatsoever. They preserve their original form when the user drags the lens, as long as their position is kept outside of the boundaries. Note that since the distortion of the rubber sheet stretching is applied in the x and y axes independently, if an element is inside the boundaries of the lens in only one axis, it will be distorted for that axis. This effect can be seen in Figure 2.9, in which, for example, the cities of Calgary and Memphis are distorted, even though they are not inside any lens. The other difference between the implementations of the rubber sheet stretching in relation to the single point fisheye is that the scales are updated in real time as the domains and

Figure 3.1: The generated view for 6 dimensions and 7 sections, distorted by the single point lens (a) and rubber sheet stretching (b).



Source: The author.

ranges change when the lens changes size or position in the rubber sheet stretching.

The lens, that is the rectangle formed between the four delimiting lines, is drawn in bold, to facilitate telling it apart from the rows and columns of the DS. A black circle is placed on the bottom-right corner of the lens, and serves as a draggable handle to resize the lens. The lens itself can be dragged with the mouse to change its position. An issue occurs when the lens is dragged beyond the boundaries of the view. This causes a portion of the lens to not be visible anymore, hiding all elements inside it from the user. To solve this, it is allowed to move the lens outside of the boundaries of the view only up to half of the lens width or height. When the lens is at that position, all elements that were outside of the view will be inside it, visible to the user.

3.2.3 Multi-lens Fisheye

A new method of applying the fisheye lens has been theorized and implemented in this work. The development of the previous distortion techniques serves as a technical basis to create the multi-lens fisheye, and it is the final technique developed in this work. When a single point fisheye lens is used in a dimensional stacking visualization with few dimensions, the generated image is magnified in a point, possibly increasing the size of a colored bucket until it covers a big portion of the screen, while still preserving the global context. The idea of the multi-lens fisheye is to put a new fisheye lens, covering only a small region of the view, and only inside this lens, exposing a dimensional stacking visualization for two more inner dimensions. More lenses can be added, and for each one, two more dimensions will be added as well. These lens are positioned hierarchically, so that every lens is placed inside another, bigger lens.

The rationale behind this concept is that each time we add a lens, we can increase the degree of how deep we go inside the dimension hierarchy, while also hiding a very big number of lines that would otherwise be drawn, and still maintaining context.

The first lens is always fixed and has the size of the whole view. Then, an indefinite number of lenses are placed on top of each other. The lenses are not just traditional single point lenses placed inside each other, though. Every column, row and cell have to travel smoothly, without abrupt snaps when the lenses move through them. If a single point fisheye lens is placed on top of another, and both lenses are dragged through the screen, an object passing between them will snap to a different position, it will not be seamless. Another problem is that this seamlessness has to occur for any object passing through any combination of lenses in any position. To accomplish this, we create a intricate combination of multiple fisheye scales that changes at every frame. It is necessary four scales for each lens, Two scales for the x axis and two for the y axis. The position of any lens can be described by the formula

$$pos = \frac{m - \frac{s}{2}}{l - 1} * i \tag{3.1}$$

in which pos is both the x and y attributes of the lens, m is the x or y of the focus point, s is the desired length of the smallest lens, l is the desired number of lenses to be generated and i is the index of the lens, between 0 and $l - 1$. The size, or length of any lens can be described by the formula

$$size = w - \frac{i(w - s)}{l - 1} \tag{3.2}$$

in which w is the width or height of the screen. Using this method to calculate the proportions of the lenses ensures that they are evenly spaced in the screen, thus balancing the size of the context regions for each lens, and helping the user to retain local and global context, regardless of which part of the screen is being focused.

To generate the scales, first it is necessary to understand how they are combined and joint. Consider the multi-lens fisheye for two lenses. The first lens fits exactly on top of the whole screen, with $x = 0, y = 0, width = w, height = h$. w and h are the resolution of the screen. The second lens is a single point fisheye lens that has a previously defined length and is placed exactly in the middle of the screen. It has $x = w-(s/2), y = h-(s/2), width = s, height = s$. The second lens is a single point fisheye lens and needs two scales, one for each axis. The first lens, however, requires four scales. For the x axis, one scale is for elements inside the first lens and outside of the second lens, but located on the left side of the second lens, and another scale is for elements on the right side of the second lens. For the y axis, one scale is for elements on the top of the second lens and one for elements on the bottom. The focus point of these scales is the respective edge of the second lens. To pass the position of an element of the dimensional stacking image through its corresponding scale, each attribute of the element pass through a function, that depending in which domain of the scales the attribute is, it returns the mapping done by the correct scale, of that attribute. Consider a column that is on the left-most side of the screen. It is being passed through the left scale of the x axis of the first lens. If the second lens is dragged through the screen and passes through the column, the column will now be subject to the x scale of the second lens only.

To accomplish this with any number of lenses, the function that selects the right scale for that attribute loops through the list of the lenses, and apply the correct scale by calculating the domain that the attribute is in. The domain of any scale is from the edges of the lens to the focus point of the inner-most lens minus the length of a bucket on that dimension. This has the effect that the size of any lens is exactly the size of a single bucket on the outer lens.

When it is generated the dimensional stacking view, every element is being drawn at the screen. Since the multi-lens fisheye focus on a specific region of the screen, we might want to hide details of the context regions. A detail of a region is defined as the number of dimensions of the dimensional stacking visualization shown at that region. The rows and columns of every dimension will be drawn, even when there are no elements to be shown. Each lens should expose only two new dimension. To account for this, the function that selects the corresponding scale for each attribute in each element also receives the information of which dimension that element belongs. If the element, for example, is a block of the sixth dimension, but it is inside a lens of the fourth dimension, the block is not drawn in the screen. This way, the first lens shows the dimensional stacking for the first two dimensions, the second lens for the first four dimensions, and so on.

If the inner-most lens is positioned on top of a region containing blocks, not only the blocks of the inner-most dimensions will be drawn, but also any blocks of outer dimensions. The blocks overlap and breaks the visualization. To fix this, the blocks of the outer dimensions become opaque when inside the inner lens. This creates a clear difference between the opacity of the blocks of different dimensions, and make it easy to tell them apart, even though they are on top of each other.

3.3 The Application's Interface

The application developed to implement the ideas presented in this work is intended to provide a visual presentation of generic datasets, a way to navigate this view in real time, and different features to facilitate the manipulation of the visualization and interactions with it. It was created as a web application in javascript, using the d3 library to generate and manipulate the visualization, and bootstrap (BOOTSTRAP, 2018) to create the layout and structure of the application. The application consists of a folder with several files: an *index.html* file that contains the page layout, a *javascript* file to generate the visualization and manipulate the applciation's layout dynamically, and other files to load libraries and color schemes.

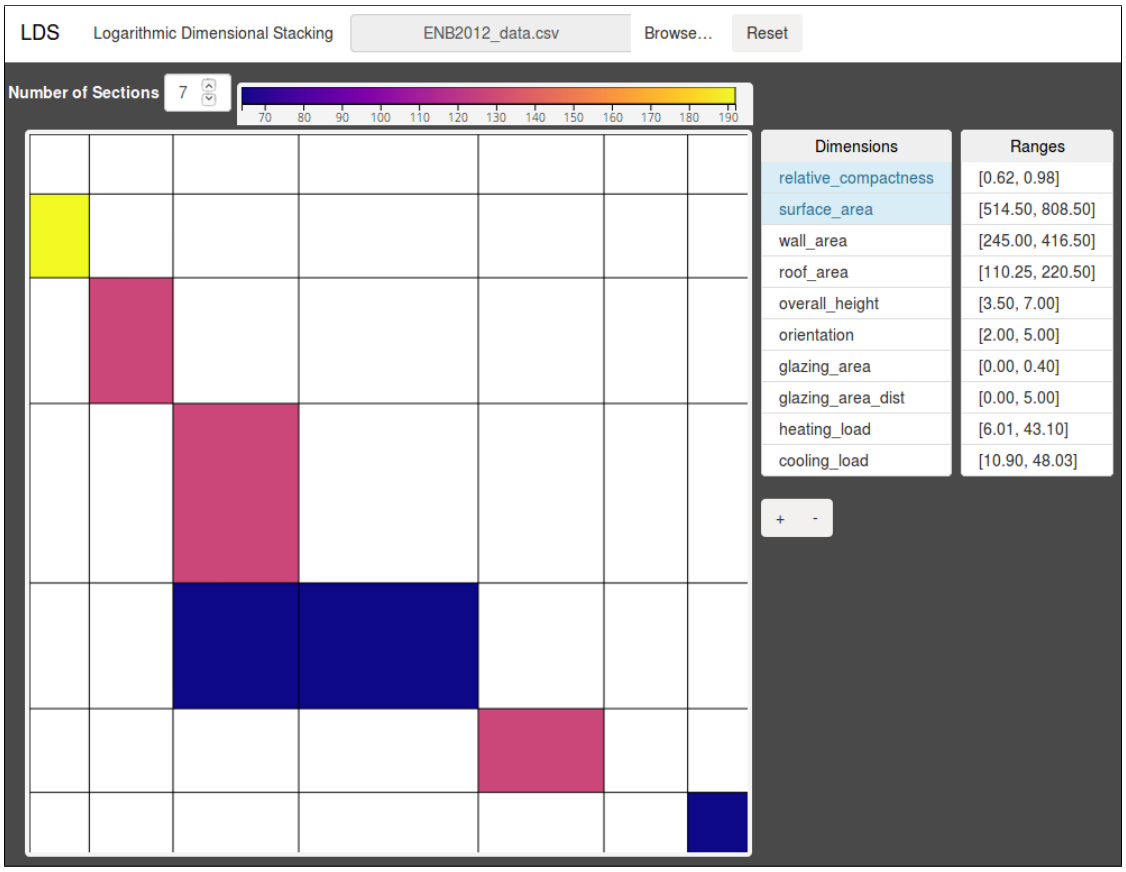
The application's interface has several different components. Figure 3.2 shows an example of the application's interface after the user has loaded a dataset.

1. **Browse:** To select the dataset to be used, the user has to browse it from the computer. The dataset is loaded and the visualization is generated automatically. Later on, the user can browse again for other dataset, and the visualization will update accordingly, generating a new DS view and removing the old dataset.
2. **The view:** The visualization generated by the dimensional stacking, along with the distortions applied by the fisheye lens.
3. **Dimensions:** The list of the dimensions of the dataset, ordered according to the order of the dimensional stacking. The dimensions can be dragged and dropped to change this order for any other order desired. This will draw a new dimensional stacking visualization and update the changes in the screen.
4. **Ranges:** The list of the range of values for each dimension of the dataset.
5. **Reset:** To return the application to its original state, the user can press the Reset button, that changes the focus point back to the center, backtracks the dimension

level to the beginning, resets the smallest lens size, and reorders the dimension list according to the original order of the dataset.

6. **+ (plus):** Recreates the dimensional stacking visualization with two more dimensions, adds a new lens to the view and update the Dimensions list, highlighting the added dimensions.
7. **- (minus):** Backtracks the dimension level by two, removes a lens from the view, and remove the highlight of the removed dimensions on the Dimenson list.
8. **Number of Sections:** Allows the user to configure the number of sections in the dimensional stacking visualization. A new visualization is generated and drawn in the view.
9. **Amount scale:** The color scale of the blocks in the dimensional stacking visualization. This scale is generated when a dataset is loaded. To create it, we define a range based on the two numbers representing the biggest and smallest amount of data instances in blocks, and map it to a color scale. This scale is the same for blocks in all dimensions, and it is updated when the dataset is changed.

Figure 3.2: The application's interface in its initial state, after loading a dataset.



Number of Sections 7

Dimensions	Ranges
relative_compactness	[0.62, 0.98]
surface_area	[514.50, 808.50]
wall_area	[245.00, 416.50]
roof_area	[110.25, 220.50]
overall_height	[3.50, 7.00]
orientation	[2.00, 5.00]
glazing_area	[0.00, 0.40]
glazing_area_dist	[0.00, 5.00]
heating_load	[6.01, 43.10]
cooling_load	[10.90, 48.03]

Source: The author.

4 EVALUATION AND RESULTS

This chapter discusses the results obtained through iterations of development, receiving user feedback, and analyzing the success of different techniques implemented. It is discussed the problems encountered, the techniques and features that failed in their objective, and a more detailed analysis of the multi-lens fisheye by using the application to discover patterns and properties in different datasets.

4.1 Techniques and features developed

While the single point and the rubber sheet stretching distortion techniques accomplished the goal of visualizing a dimensional stacking technique with more dimensions than it would be possible without it, there were some drawbacks. The single point fisheye distorts the entire screen at once. That means that when increasing the distortion factor in real time with the purpose to zoom in the focus point and see the region in a bigger size, the rest of the screen shrinks to accomodate this, sometimes unnecessarily, causing the colored blocks on the edges of the image to be almost invisible to the naked eye. The local context is preserved, but the global context is impaired. The rubber sheet stretching only distorts the region inside the lens, and the rest of the visualization retains its original shape. To visualize a region in more detail, it is necessary to increase the size of the lens, and refocus it on the desired region, and then shrink it again to navigate through the view. This process has to be repeated each time the user wants to focus on an region. The global context is preserved, but the local context is impaired. To solve this, seamless transition between the two distortions was implemented, but it still was not enough. It was not yet possible to dramatically increase the number of dimensions, and the method seemed to not work well with more than 6 dimensions and 7 buckets per section. The ability to understand the global and local context gets worse as the mentioned parameters increases. Multiple attempts were made to solve these issues. In the single point technique, a rectangular region around the focus point was defined, and inside it, the blocks and lines of 6 dimensions are shown, and outside it, only for the 4 outer dimensions. For the rubber sheet stretching technique, the same idea was implemented, but the rectangular region is the lens. This increased performance significantly, by hiding a very large unnecessary number of lines that were being displayed. It did not solve completely the context problem, because the blocks on the edges of the screen were still being shrank to the point of being less than the size of a single pixel. To solve this, a resize mechanic was developed. When a block reaches a distance between its edge and an edge of the screen less than a threshold, the block "glues" to the edge of the screen. This causes any block close to the edge of the screen to become visibly big, giving a clear indication for the user that there are blocks there. Another feature developed was a minimap that showed the whole screen in a reduced size, as well as the lens or focus point location in it. This helped with the global context, because the user could see where he or she was, in relation to the whole image. These features caused the layout of the application to get increasingly complex and overwhelming. There were too many components acting and moving, and that increased the learning time for the user, while not in fact solving the problems with the distortion techniques themselves. It was also necessary to perform different operations and visually analyze and process multiple regions to use it properly.

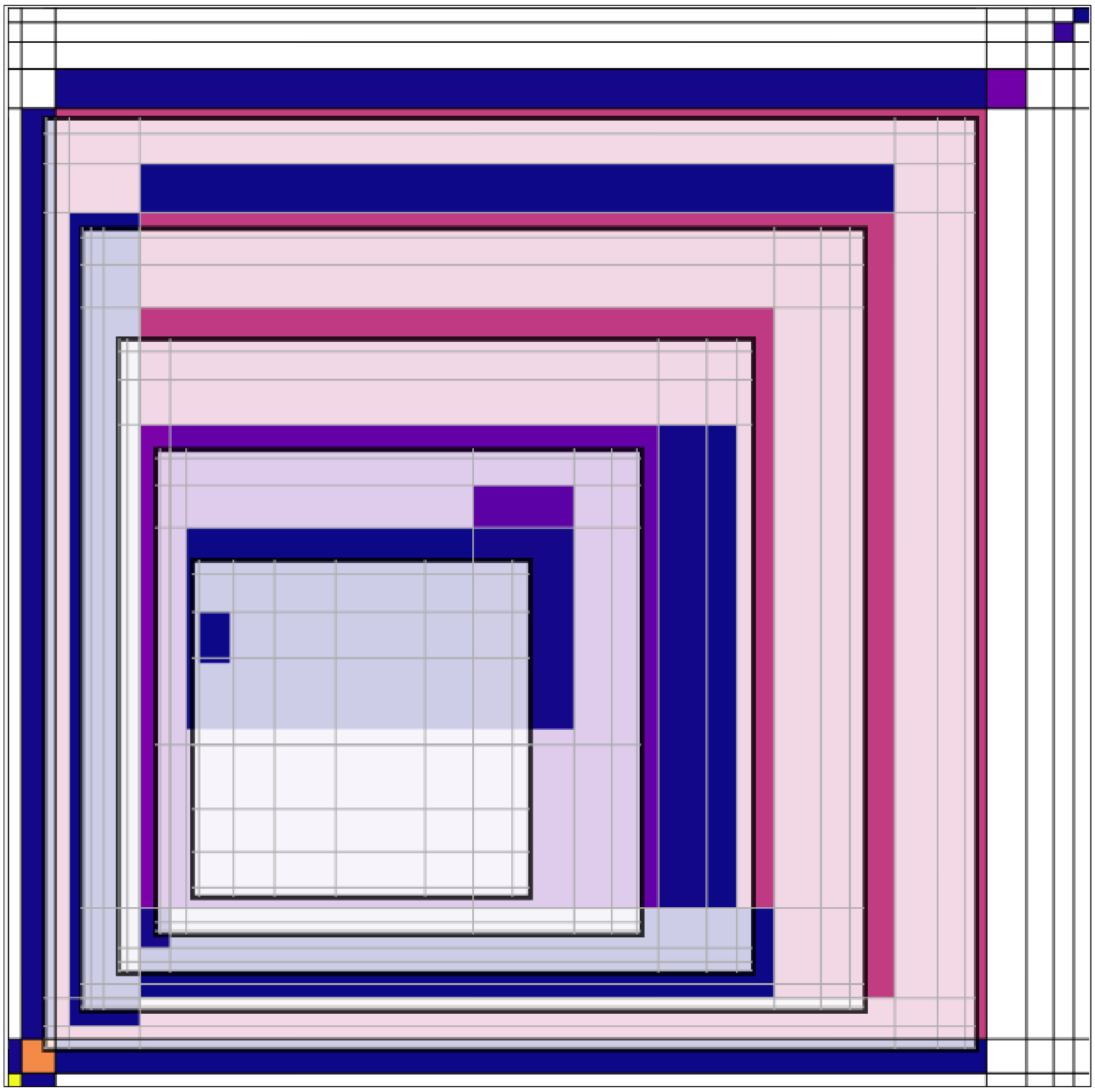
4.2 Exploring Datasets with the Multi-Lens Fisheye

To evaluate whether the application is useful, a dataset has to be loaded through it, and peculiar or useful patterns have to be discovered about it. First, though, there has to be a confirmation of the ability of the multi-lens fisheye to display and navigate in real time a dimensional stacking view of higher dimensionality than what is traditionally possible. In Figure 4.1, it is shown an example of just how many dimensions can be visualized at the same time using the multi-lens fisheye. Even with 12 dimensions, the number of rows and columns in the screen in incredibly low. The size of the buckets of the inner-most dimensions are clearly visible. There is a smooth transition of elements when the lens move through the screen. The elements of the first two dimensions are visible, retaining global context. For each lens, we can see ,without visual clutter, where are the interesting regions, where the view is empty, where there are more instances and where there are fewer. Thus, local context is retained as well. Performance struggles as the number of dimensions and sections increases, but still allows for real time interaction.

The application was developed to handle datasets in common formats, such as *.csv* or *.data*. The requirement for the displacement of the information in the file is very simple as well. The application process it so that each dimension is a column, the first row is the names of the dimensions, and the other rows are data instances. Class attributes must first be mapped to a numeric identifier, and there cannot be any empty entries.

The first dataset used is a set from the UCI Machine Learning Repository(DHEERU;

Figure 4.1: Example of the multi-lens fisheye applied to an image generated with dimensional stacking, with 6 lenses, 12 dimensions and 7 sections.



Source: The author.

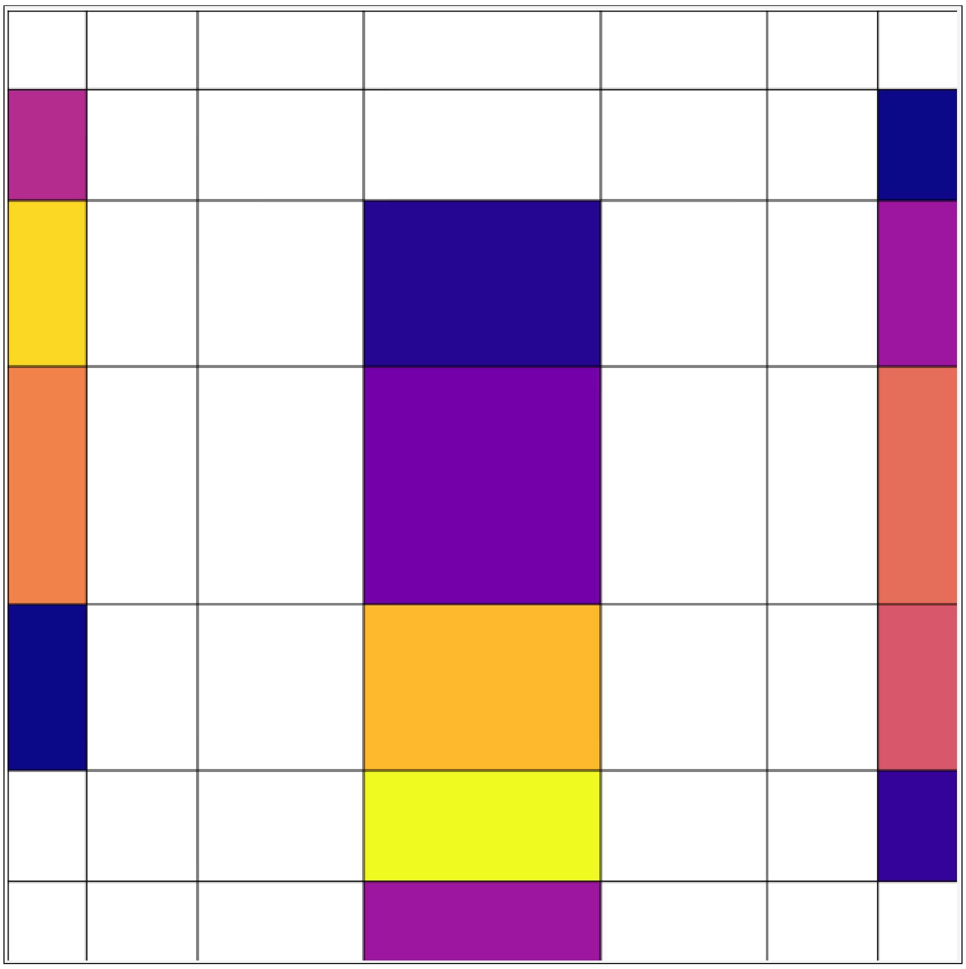
TANISKIDOU, 2017) containing chemical properties of wines grown in a region of italy, but originated from three distinct cultivars. The analysis of these wines produced 13 continuous attributes, corresponding to 13 dimensions, and another dimension for the class identifier. This identifier classifies the three cultivars, and is the main attribute to be used in a multivariate analysis of the dataset. This dataset was selected due to having a well defined purpose, a modestly high number of dimensions and few instances.

This dataset was loaded into the application, and a 2d dimensional stacking was generated with the default configuration, seen at Figure 4.2, along with an amount scale of range $[1, 28]$, and a list of dimensions was displayed, highlighting the first two dimensions, $Class$ and $Alcohol$.

In this initial 2d-dimensional stacking view at Figure 4.2, the $Class$ dimension can clearly be seen. Since the default number of sections per dimension is 7, the application mapped the class identifier of range $[1, 3]$ to the first, the fourth and the seventh bucket, generating three vertical "strips". The middle one is positioned lower than the others, indicating that the highest and lowest ABV of wines of the second class are lower than in other classes. For any class, if we look at the bucket with the color closest to the right-most side of the scale, and look to the other buckets on the $Alcohol$ dimension (the y axis) until we reach the edges of the strip, we can see that the colors of the class are each time more to the left of the scale. We can infer that the dataset, in relation to the $Alcohol$ dimension, follow a normal distribution, within the same class. In the second class (the middle strip), we can see that the most yellow bucket is near the bottom, while in other classes, it is closer to the top. This happens because the average ABV of the wines in the second class is less than in the other classes. Just by glancing at the view and analyzing the colors in relation to the amount scale, we can see that in the second class, there are more wines with lower ABV than in the other classes. The class with the highest median ABV is probably the first, but to reach this conclusion with more certainty, it would be needed further analysis.

Two more dimensions were added, $Color\ intensity$ on the x axis, and every other dimension was tested on the y axis. By moving the lens across the buckets, it was discovered that wines in the second class have on average a lower value of $Color\ intensity$, as almost all buckets for the class identifier have data instances on the left-most side of them. Other classes had buckets of color intensity more distributed across them. Little correlation was found between the color intensity and the other dimensions, but it was discovered that only for the first and second class, there was a correlation between $color\ intensity$

Figure 4.2: The initial state of the view after loading the wine data set.



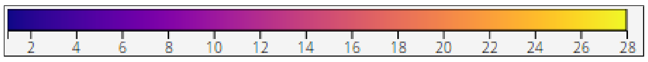
Source: The author.

and ash, as seen in Figures 4.4c and 4.4a, that seemed to follow a normal distribution.

As we increased the number of dimensions, the visualization became less dense, and finding patterns became harder. This happened because the wine dataset have a relatively low number of instances. This phenomenon can be described by what is called the curse of dimensionality (KEOGH; MUEEN, 2011). By adding new dimensions, it is needed exponentially more volume to effectively sample the new information space. The wine dataset seemed to not have enough data instances to properly identify patterns of higher dimensionality, even though there might be.

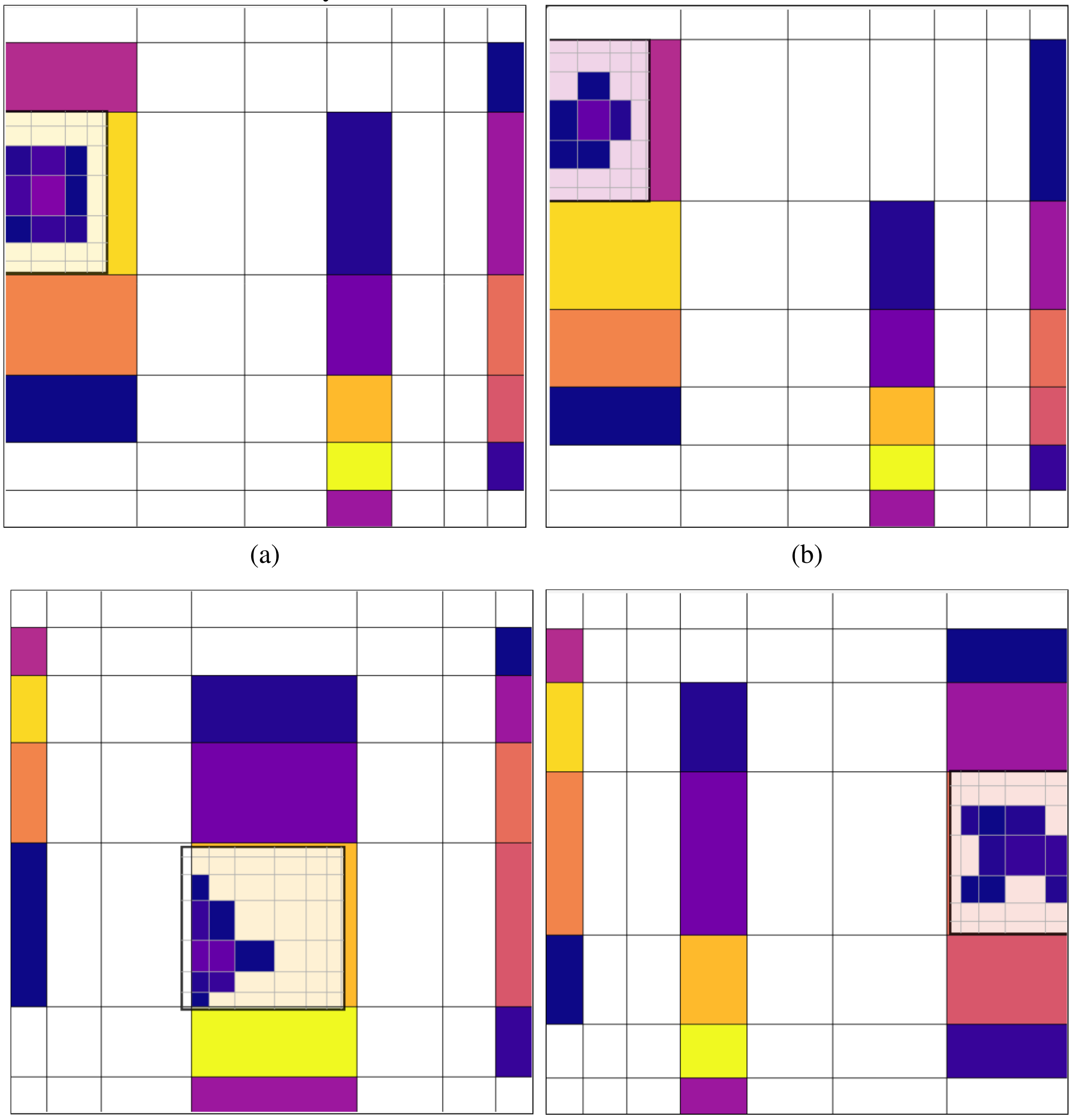
Another dataset selected was the Abalone set (DHEERU; TANISKIDOU, 2017). It has fewer attributes than the wine set, but a much higher number of instances. It consists on predicting the age of abalone by analyzing its physical measurements. It has 4177 instances, and the following attributes (ABALONE. . . , 1995):

Figure 4.3: The color scale generated after loading the wine data set. It represents the number of data instances inside the same bucket.



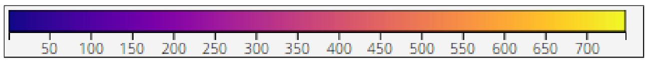
Source: The author.

Figure 4.4: Moving the lens through the view in the wine dataset, to compare the correlation between color intensity and ash, based on different values for class and alcohol.



Source: The author.

Figure 4.5: The color scale generated in the application, after loading the abalone dataset.



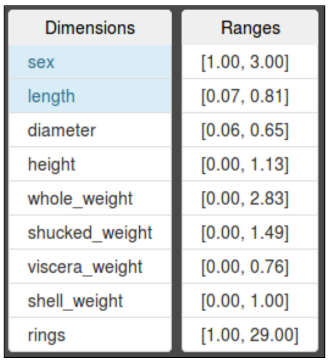
Source: The author.

1. *Sex*: male, female, and infant.
2. *Length*: Longest shell measurement.
3. *Diameter*: perpendicular to length.
4. *Height*: with meat in shell.
5. *Whole weight* whole abalone.
6. *Shucked weight:* weight of meat.
7. *Viscera weight:* gut weight (after bleeding).
8. *Shell weight*: after being dried.
9. *Rings*: +1.5 gives the age in years.

In Figure 4.6 and Figure 4.5, it is shown the dimension list, ranges and color scale generated by the application in its initial state after selecting the abalone dataset. It was chosen the sex as the first dimension, and alternated the other between $whole\ weight$, $shell\ weight$, $rings$ and $length$. The first vertical "strip" in Figure 4.7 represents the male population, the second, female, and the third, infants. It was noted that males and females have equal $whole\ weight$ and $shell\ weight$, while infants have lower weights and $length$. Attributes for infants, however, showed a larger standard deviation than expected. It can be speculated that this is because infants grow at different rates, or, more likely, because the dataset contains infants at different growth stages. Since the $ring$ attribute determines age, in 4.7b, it can be seen that the infant's age range is almost as large as males or females.

Then, the dimension number was increased to 6, the section number set to 7, and it was selected dimensions in the following order: sex, $diameter$, $length$, $height$, $whole\ weight$ and $rings$. The lenses were moved through the view, and it was analyzed the relation between the weight of abalone and its age, given the selected physical measurements. This process of dragging the lenses and using it to navigate through the view is captured in Figure 4.8, 4.9 and 4.10. Inside the inner lens, when moving through it through the view, the relation between $whole\ weight$ and $rings$ is not linear, but actually follows what seems to resemble a normal distribution. This happens for any sex or

Figure 4.6: The list of dimensions and its corresponding ranges after loading the abalone dataset.



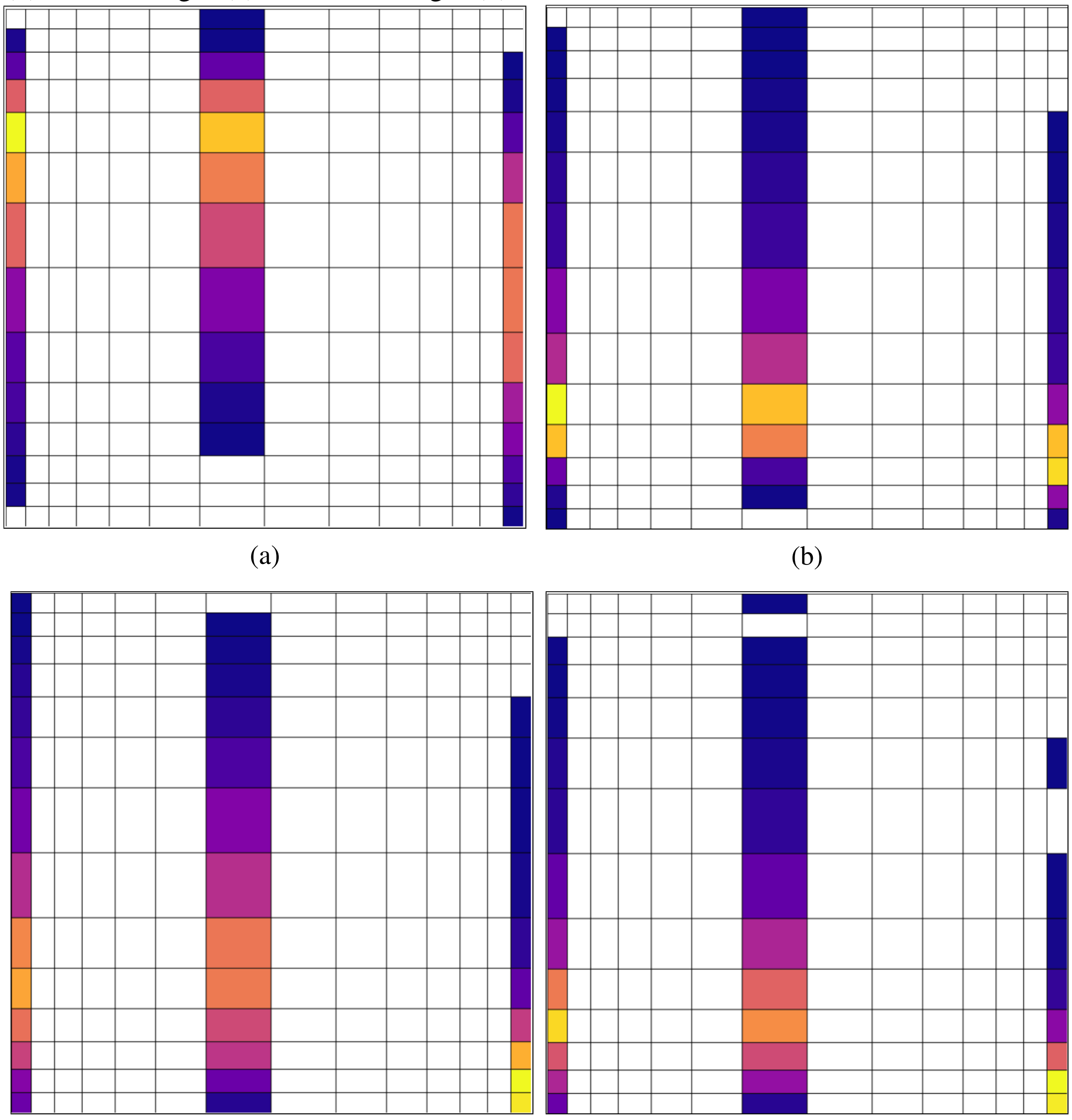
Dimensions	Ranges
sex	[1.00, 3.00]
length	[0.07, 0.81]
diameter	[0.06, 0.65]
height	[0.00, 1.13]
whole_weight	[0.00, 2.83]
shucked_weight	[0.00, 1.49]
viscera_weight	[0.00, 0.76]
shell_weight	[0.00, 1.00]
rings	[1.00, 29.00]

Source: The author.

size. In Figure 4.8, when the lens moves from the bottom of the view to the top, the focus region covers abalone from lower to higher $diameter$. From figures 4.8a to 4.8i, the distribution of blocks inside the inner lens is somewhat similar as the lens moves to the top, but the mean increases slightly. Notice how Figure 4.8a shows a distribution closer to the left and bottom of the lens, while in 4.8i, it is located more on the right and top of the lens. This phenomenon happens for females as well. Weight seems to be a true, but weak indicator for age, because the fact the the distribution moves to the right and top side of the lens as its measurements increases means that abalone with higher $diameter$, $length$, $height$ have more $whole\ weight$ and $rings$. One difference, though, is that the distribution in infants seems to be more "linear-like" than in females and males, as seen in 4.10b and 4.10e. This could be an indication that the age of infants correlates strongly with their $whole\ weight$, and as the abalone becomes an adult, other unknown factors may exert a higher influence on its age.

The $whole\ weight$ dimension was then switched for the $shucked\ weight$, generating a DS view with dimensions in the order: sex, $diameter$, $length$, $height$, $shucked\ weight$ and $rings$. After moving the lens through the view (Figure 4.11), it was discovered that, for males and females, the age seems to drop with shucked weight, regardless of its size. In infants with small measurements, seen in Figure 4.11c, this pattern occurs as well. For medium size infants (Figure 4.11d), the shucked weight grows with age, and for bigger infants (Figure 4.11e), the pattern returns to be similar with adults. By switching the $shucked\ weight$ dimension with $shell\ weight$, the distribution resembles a linear distribution, for any sex, age or size. This means that the shell of abalone consistently grows heavier with age, while its weight without the shell, after fully developed, actually

Figure 4.7: The abalone dataset, with sex attribute being tested alongside length (a), rings (b), whole weight (c), and shell weight (d), with 14 sections.



(a)

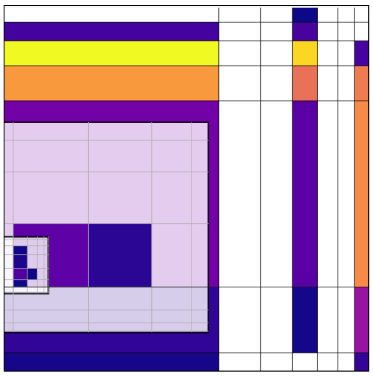
(b)

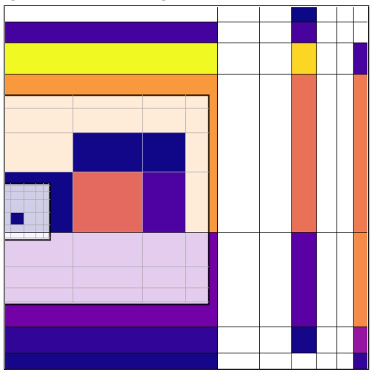
(c)

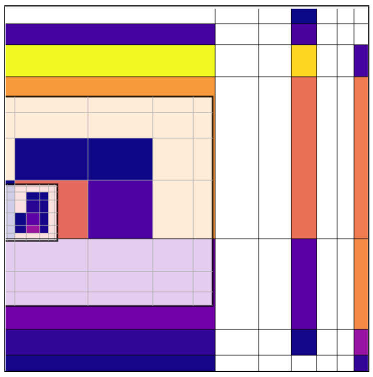
(d)

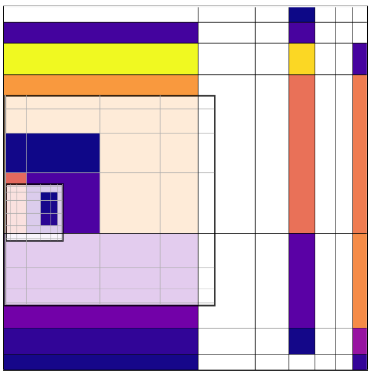
Source: The author.

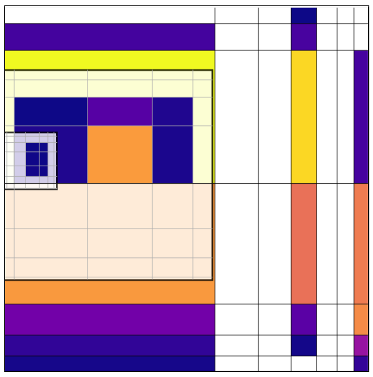
Figure 4.8: Analyzing the difference in distribution between *whole weight* and *rings* across different *diameter*, *length* and *height* of males in the abalone dataset.

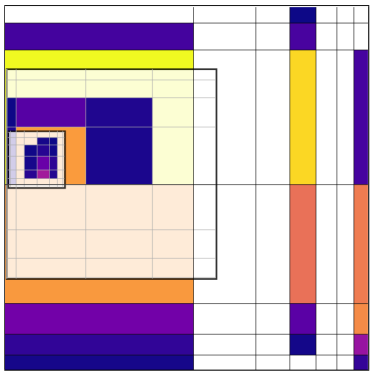

(a)

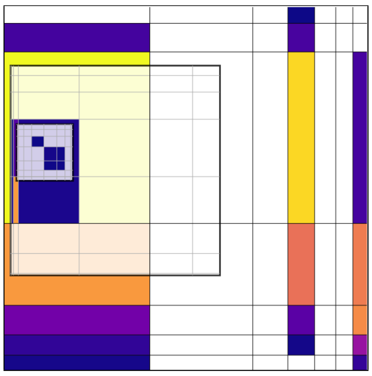

(b)

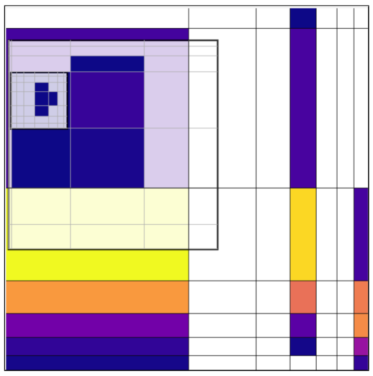

(c)

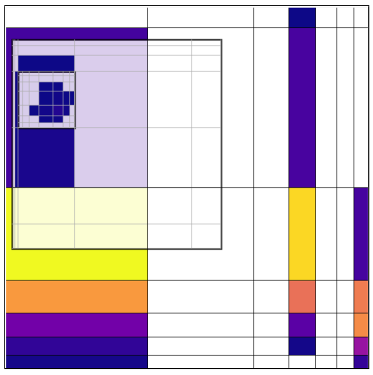

(d)


(e)


(f)

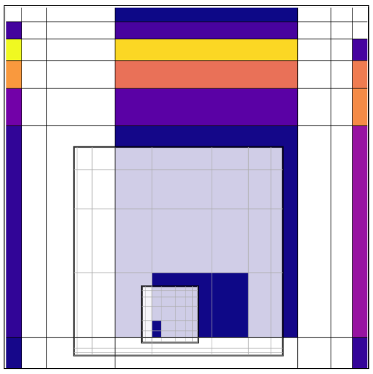

(g)

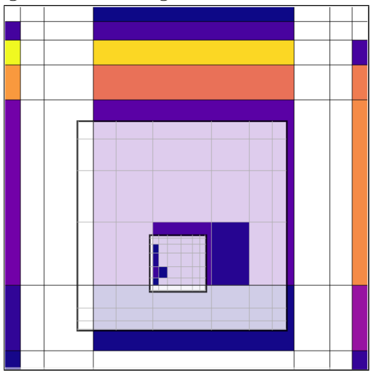

(h)

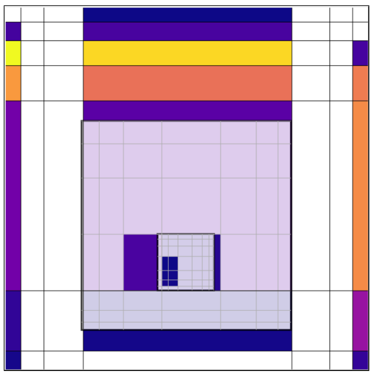

(i)

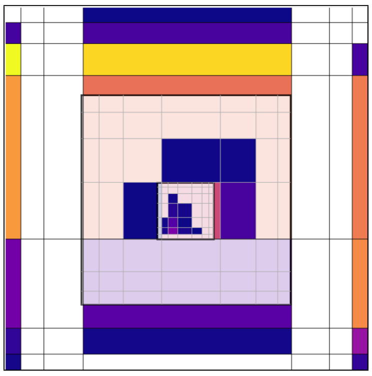
Source: The author.

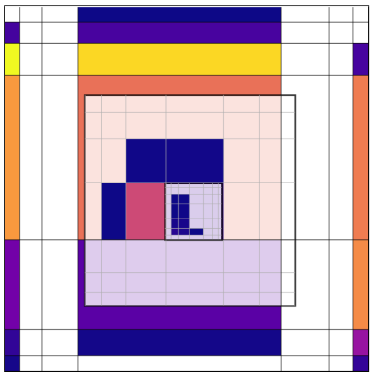
Figure 4.9: Analyzing the difference in distribution between $whole\ weight$ and $rings$ across different $diameter$, $length$ and $height$ of females in the abalone dataset.

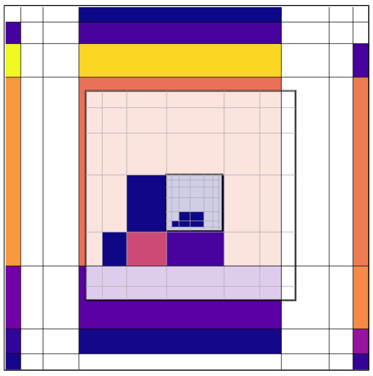

(a)

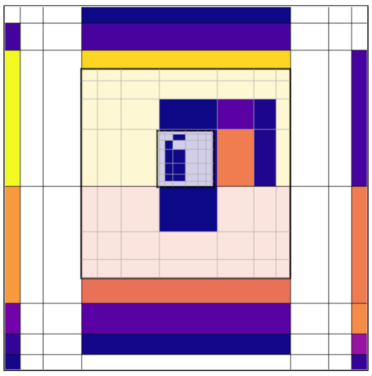

(b)

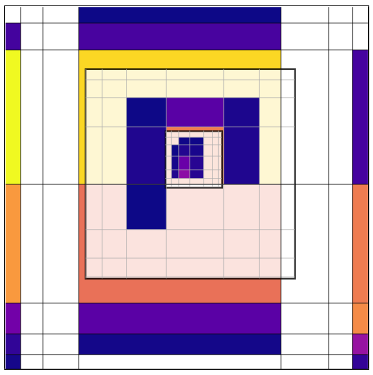

(c)

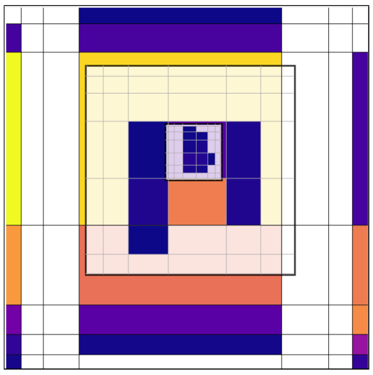

(d)

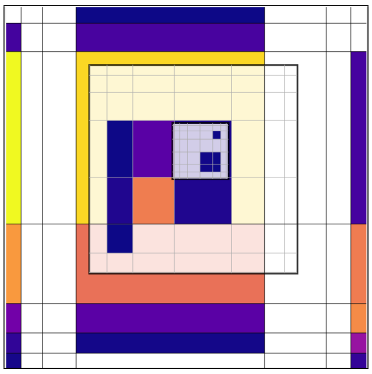

(e)

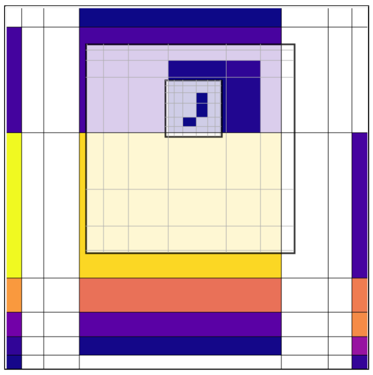

(f)

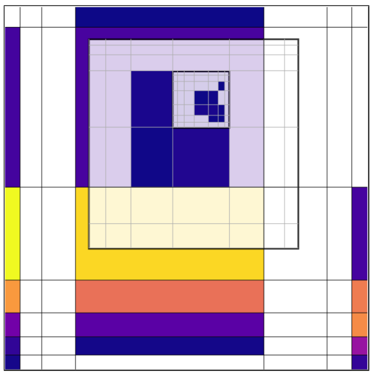

(g)


(h)


(i)

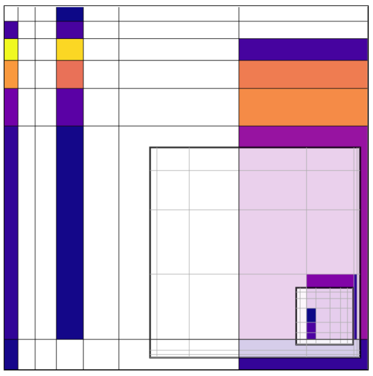

(j)

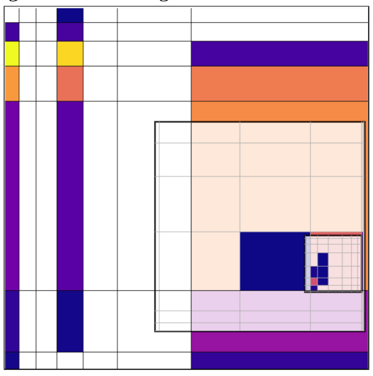

(k)

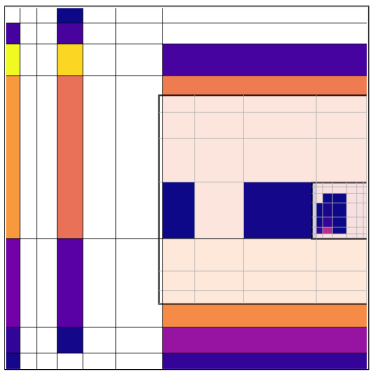

(l)

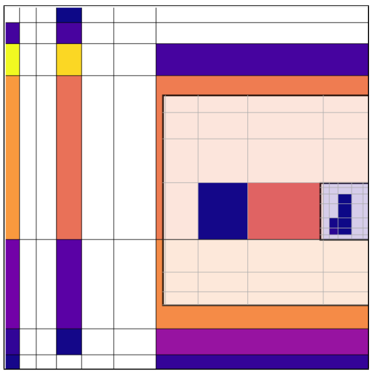
Source: The author.

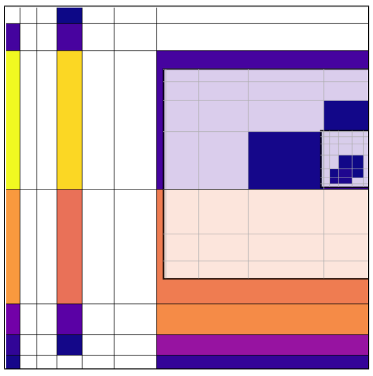
Figure 4.10: Analyzing the difference in distribution between $whole\ weight$ and $rings$ across different $diameter$, $length$ and $height$ of infants in the abalone dataset.

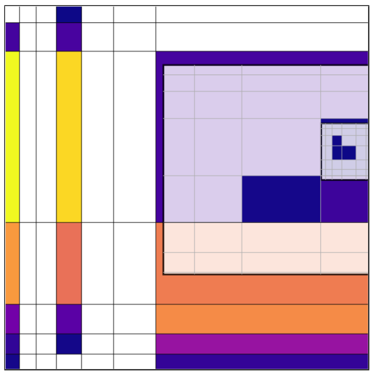

(a)


(b)


(c)

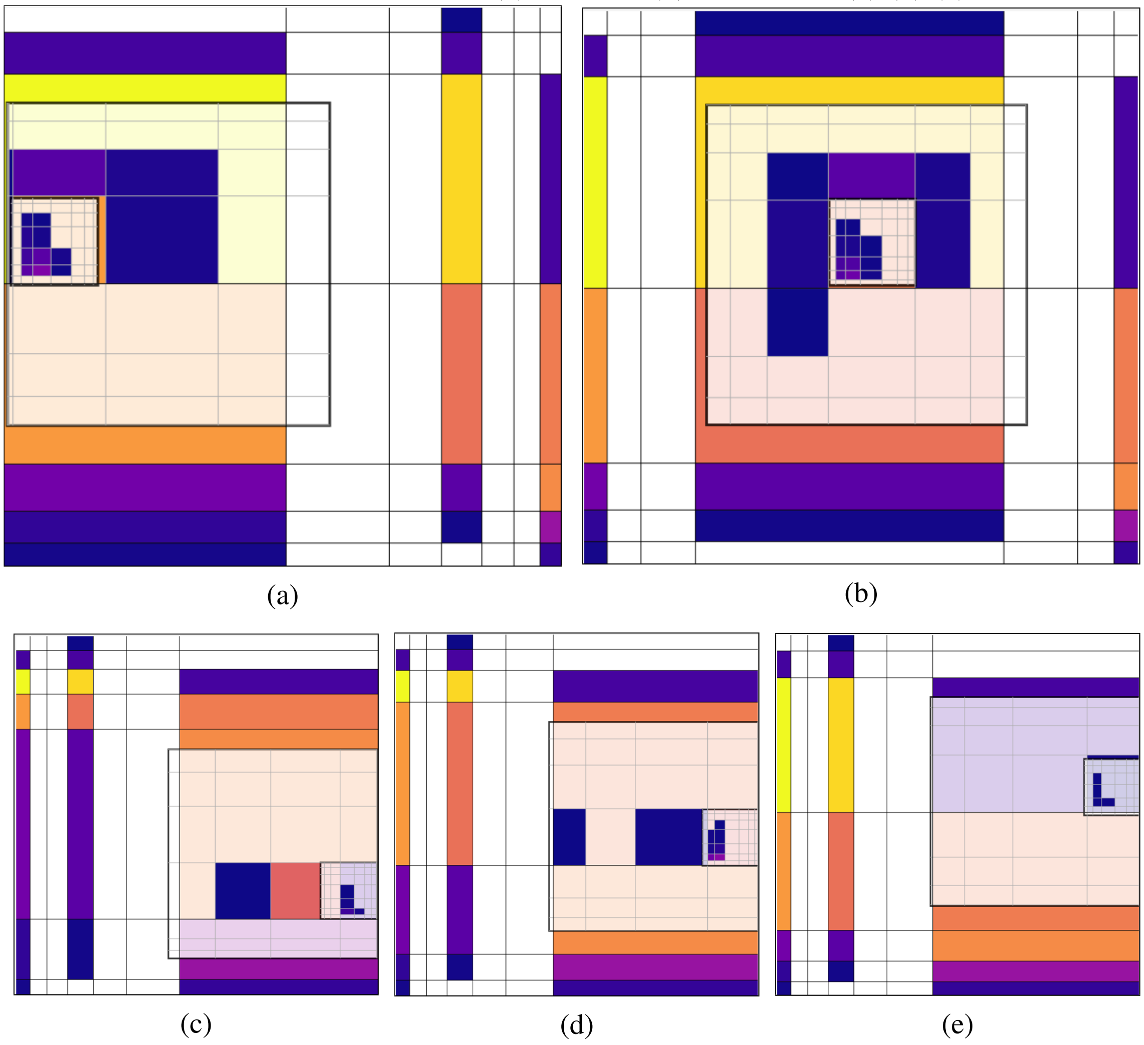

(d)


(e)


(f)

Source: The author.

Figure 4.11: Analyzing the relation between *shucked weight* and *rings* across abalone with different measurements in males (a), females (b) and infants (c) (d) (e).



(a) (b) (c) (d) (e)

Source: The author.

decreases over time. One conjecture that could be made is that this shift in weight accurately describes the developmental process of abalones. This could also have happened because there is the possibility that some gathered samples were already in state of decay, causing changes in their weight and size.

5 CONCLUSION

Dimensional Stacking is a visualization technique that maps a multidimensional dataset into a 2D space. As the number of mapped dimensions increases, however, it generates images with exponentially larger resolutions.

In this work, it was introduced a method to solve this problem by using the concept of the fisheye lens to distort the image in order to maintain a low resolution, but still keeping the ability to visualize it clearly, in real time, with integrated context. It was developed an application that implements this method and allows loading and interacting with any dataset.

The results showed that images generated with dimensional stacking, when subject to the multi-lens fisheye, produce a minuscule number of elements in the screen compared to its original state. The resolution also increases in a much lower rate, resulting in the possibility of using the dimensional stacking for visualizing datasets with an order of magnitude greater than otherwise would be possible.

It was observed that the interactions between the user and the view is crucial for the resulting quality of the multi-lens fisheye, and the application's shortcomings hindered visualization of images with more than around six dimensions. Many directions can be taken to further enhance this method's capability. First, adding legends in the view may facilitate the user's understanding of which axis represents which dimension, and which dimension is represented by which lens. Captions inside blocks can also be useful, by providing the exact value in the color map and the block's coordinate. By adding the ability to move each lens independently, regions of interest could be fixed, and inner lenses could be used to explore them. Each lens currently is set to a determined size, but allowing it to be set in real time could help the user to understand context and better explore regions of different sizes.

There is potential for much better implementations of the multi-lens fisheye, that could achieve performance of orders of magnitude higher. Optimizations could be done by calculating in real time only the positions of elements visible to the user, or even generating them in real-time, as needed.

The techniques proposed in this work seem promising high dimensional data visualization, but further research could be done to compare it with state of the art techniques. Thorough analysis and comparison of performance, easiness of use and effectiveness could be done to better verify its usefulness.

REFERENCES

ABALONE data. 1995. Available from Internet: <https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.names>.

APPERLEY, M. D.; SPENCE, R. Bifocal display. In: SOEGAARD, M.; DAM, R. F. (Ed.). **The Encyclopedia of Human-Computer Interaction, 2nd Ed**. [S.l.]: The Interaction Design Foundation, 2013. chp. 7.

BECKER, R. A.; CLEVELAND, W. S. Brushing scatterplots. **Technometrics**, Taylor & Francis, v. 29, n. 2, p. 127–142, 1987.

BOOTSTRAP. 2018. Available from Internet: <https://getbootstrap.com/>.

CARPENDALE, M. S. T.; MONTAGNESE, C. A framework for unifying presentation space. In: ACM. **Proceedings of the 14th annual ACM symposium on User interface software and technology**. [S.l.], 2001. p. 61–70.

DHEERU, D.; TANISKIDOU, E. K. **UCI Machine Learning Repository**. 2017. Available from Internet: <http://archive.ics.uci.edu/ml>.

FISHEYE Distortion. 2018. Available from Internet: <https://github.com/d3/d3-plugins/tree/master/fisheye>.

FURNAS, G. W. **The FISHEYE view: A new look at structured files**. [S.l.], 1981.

FURNAS, G. W. **Generalized fisheye views**. [S.l.]: ACM, 1986.

GRINSTEIN, G.; TRUTSCHL, M.; CVEK, U. High-dimensional visualizations. In: CITESEER. **Proceedings of the Visual Data Mining Workshop, KDD**. [S.l.], 2001. v. 2, p. 120.

HANNA, A. R.; RAO, C.; ATHANASIOU, T. Graphs in statistical analysis. In: **Key Topics in Surgical Research and Methodology**. [S.l.]: Springer, 2010. p. 441–475.

INSELBERG, D. Parallel coordinates: a tool for visualizing multi-dimensional geometry. **3D Digital Imaging and Modeling, International Conference on**, IEEE Computer Society, Los Alamitos, CA, USA, p. 361,362,363,364,365,366,367,368,369,370,371,372,373,374,375,376,377,378, 1990.

JACOBS, A. The pathologies of big data. **Queue**, ACM, v. 7, n. 6, p. 10, 2009.

JIANG, R. et al. Scenario discovery workflow for robust petroleum reservoir development under uncertainty. v. 6, 11 2016.

JR, M. S. W.; GRIFFIN, P. Piecewise linear rubber-sheet map transformation. **The American Cartographer**, Taylor & Francis, v. 12, n. 2, p. 123–131, 1985.

KEIM, D. A. Information visualization and visual data mining. **IEEE Transactions on Visualization & Computer Graphics**, IEEE, n. 1, p. 1–8, 2002.

KEOGH, E.; MUEEN, A. Curse of dimensionality. In: **Encyclopedia of machine learning**. [S.l.]: Springer, 2011. p. 257–258.

LANGTON, J. T.; PRINZ, A. A.; HICKEY, T. J. Combining pixelization and dimensional stacking. In: SPRINGER. **International Symposium on Visual Computing**. [S.l.], 2006. p. 617–626.

LEBLANC, J.; WARD, M. O.; WITTELS, N. Exploring n-dimensional databases. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of the 1st conference on Visualization'90**. [S.l.], 1990. p. 230–237.

LEUNG, Y. K.; APPERLEY, M. D. A review and taxonomy of distortion-oriented presentation techniques. **ACM Transactions on Computer-Human Interaction (TOCHI)**, ACM, v. 1, n. 2, p. 126–160, 1994.

MACKINLAY, J. D.; ROBERTSON, G. G.; CARD, S. K. The perspective wall: Detail and context smoothly integrated. In: ACM. **Proceedings of the SIGCHI conference on Human factors in computing systems**. [S.l.], 1991. p. 173–176.

MIHALISIN GAWLINSKI, T. S. Visualizing a scalar field on an n-dimensional lattice. **3D Digital Imaging and Modeling, International Conference on**, IEEE Computer Society, Los Alamitos, CA, USA, p. 255–262, 479–480, 1990.

SARKAR, M.; BROWN, M. H. Graphical fisheye views of graphs. In: ACM. **Proceedings of the SIGCHI conference on Human factors in computing systems**. [S.l.], 1992. p. 83–91.

SARKAR, M. et al. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In: ACM. **Proceedings of the 6th annual ACM symposium on User interface software and technology**. [S.l.], 1993. p. 81–91.

WARD, M. O. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of the Conference on Visualization'94**. [S.l.], 1994. p. 326–333.

WONG, P. C.; BERGERON, R. D. 30 years of multidimensional multivariate visualization. **Scientific Visualization**, v. 2, p. 3–33, 1994.