Using R-trees for Interactive Visualization of Large Multidimensional Datasets

Alfredo Giménez, René Rosenbaum, Mario Hlawitschka, and Bernd Hamann

Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, CA 95616-8562

Abstract. Large, multidimensional datasets are difficult to visualize and analyze. Visualization interfaces are constrained in resolution and dimension, so cluttering and problems of projecting many dimensions into the available low dimensions are inherent. Methods of real-time interaction facilitate analysis, but often these are not available due to the computational complexity required to use them. By organizing the dataset into a level-of-detail (LOD) hierarchy, our proposed method solves problems of both inefficient interaction and visual cluttering. We do this by introducing an implementation of R-trees for large multidimensional datasets. We introduce several useful methods for interaction, by queries and refinement, to explain the relevance of interaction and show that it can be done efficiently with R-trees. In order to project many dimensions into lower-dimensional spaces used for visualization, we utilize properties of the R-tree as well as existing methods for multidimensional visualization. We examine the applicability of *hierarchical parallel coordinates* to datasets organized within an R-tree, and build upon previous work in *hierarchical star coordinates* to introduce a novel method for visualizing bounding *hyperboxes* of internal R-tree nodes. Finally, we examine two datasets using our proposed method and present and discuss results.

1 Introduction

As measuring instruments advance technologically, datasets increase in both dimension and quantity. It becomes very difficult to interactively explore and analyze large, multidimensional datasets because of the high computational complexity required. Visualizing these dimensions also becomes a major problem for large dimensionalities and large datasets since standard visualization interfaces are constrained to a small number of dimensions and resolutions.

In fields of algorithms and complexity, efficient methods for data processing are often introduced through the use of hierarchical data structures. We have applied a hierarchical structure to large multidimensional datasets by generating an R-tree that contains the dataset. We utilized the efficiency of R-trees by implementing several interactive operations for analysis. Because large datasets introduce problems of clutter for low-resolution visualization interfaces, we have also used the hierarchical properties of R-trees to visualize the data at increasing levels-of-detail (LODs).

To achieve appropriate low-dimensional visualization of high-dimensional data within a hierarchy, we have implemented existing methods of hierarchical multidimensional visualization and have extended upon one of these methods in order to accommodate it for more beneficial and efficient use within an R-tree structure. Specifically, we have examined *hierarchical parallel coordinates* and built on previous work to develop a new method for *hierarchical star coordinates*.

2 Previous Work

2.1 Multidimensional Visualization

Multidimensional visualization explicitly involves the problem of how to project d dimensions onto the a small number of dimensions available on visualization interfaces. Popular methods to project d dimensions into lower dimensions include using visual cues, multiple visualizations, and alternative coordinate systems.

Chernoff [1] introduced a method using *visual cues* which involved transforming individual features of a face geometrically, and visualizing each multidimensional element as the resulting face. However, he stated that this technique is constrained to a small number of dimensions. This is an inherent problem; *visual cues* must be explicitly defined for each dimension.

Wright [2] introduced the use of multiple visualizations by using scatterplot matrices, where a matrix of two-dimensional scatterplots is displayed such that every dimension is plotted against every other dimension. Several extensions of multiple visualization schemes have been developed as well; however, with all these techniques, either the number of dimensions is constrained by the screen space available for multiple plots, or the visualization cannot display all dimensions at once, which makes interaction and analysis more difficult.

Alternative coordinate systems, in contrast to visual cue-based and multiple view-based multidimensional visualizations, attempt to provide a visualization for an unlimited number of dimensions. We have implemented and built on two of these techniques, specifically *parallel coordinates* [3] and *star coordinates* [4].

2.2 Hierarchical Data Structure Visualization

In order to generate a visualizable hierarchy from a dataset, several proposed methods involve hierarchical clustering algorithms. Fua [5] presented one of these algorithms based on proximity information and Linsen [6] presented another one based on density functions. Though both are effective for generation of a hierarchy, they both involve an added preprocessing step to cluster the data. These approaches not only have high computational complexity for generation, but for interactive operations, since these hierarchies are not guaranteed to be balanced trees.

3 Main Idea

In this paper, we introduce an efficient method to generate a hierarchical structure of data which reduces computational complexity required for interactive operations as well as methods for visualization of data within this hierarchical structure. In contrast to previous work, our method provides a great degree of efficiency and requires minimal data-specific information, while also adding functionality for analysis.

We propose using R-trees to generate this hierarchy and examine the benefits for doing so in section 4. R-trees provide functionally visualizable aggregate items within an LOD-hierarchy while also increasing efficiency. These properties serve to improve upon the problems encountered in some previous proposals (see subsection 2.2).

We examine methods to visualize aggregate items as well as data items within the R-tree, in section 5. Some of these methods are already defined in previously published papers, and we introduce a new method to visualize multidimensional R-tree aggregate items, based on some existing proposals. To analyze the efficiency and functionality of interaction, we examine two types of interactive operations, queries and refinement, in section 6. Finally, we apply our proposed methods on real datasets in section 7 and present and discuss our results.

4 R-trees: An Effective Data Structure for Interactive Visualization of Large Multidimensional Datasets

In order to provide 1) a scalable hierarchy for large multidimensional datasets, 2) visualizable and accurately representative aggregate items within that hierarchy, and 3) efficient interactive operations on the structure, we propose organizing datasets into R-trees.

4.1 Generation of an LOD-Hierarchy

R-trees generate a hierarchical structure of aggregate and data items in a "bottom-up" fashion. All individual data elements are inserted into the leaves, or the bottom level, and nodes are split into two new ones when their respective number of children exceeds the maximum number of children, m. Whenever the root node is split, a new level of detail in the hierarchy is introduced. Every internal node of the R-tree contains not only a number of children, but also a region which bounds all of its children, and this bounding region adjusts as nodes are split. Node splitting in R-trees is a widely covered research topic, as the optimal solution requires factorial time complexity [7]. Our method uses linear splitting, a method which delivers accurate enough results for our application as well as linear time complexity[1]. Because of the way they are generated, R-trees represent an LOD-hierarchy.

[1] R-tree efficiency may be improved by using more advanced node-splitting algorithms. For a more in-depth analysis, see Guttman's [7], which offers several different node-splitting algorithms with variable complexities and benefits.

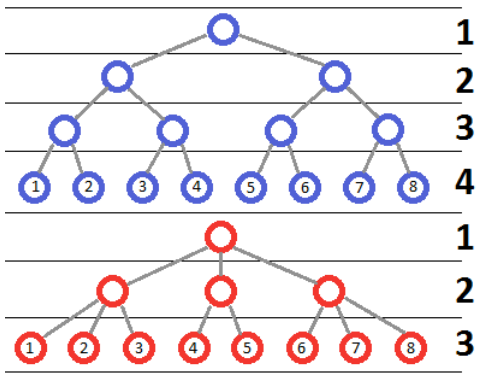

Fig. 1. Two R-trees with the same 8 elements inserted, but with different user-specified values for m (maximum number of children per node). The top has an m value of 2 and 4 levels-of-detail (LODs), while the bottom has an m value of 3 and 3 LODs. The top requires storage of 7 *hyperboxes* (one per internal node), while the bottom requires storage of 4. The ability to define m allows for dynamic LODs and storage space, and in turn, different quantities of refinable *hyperboxes* and regions-of-interest (ROIs) within each level.

R-trees allow for alteration of their internal tree depth, and different tree depths directly affect both user preferences and storage space. We can control the depth of the hierarchy by specifying different values for m. A larger value of m corresponds to fewer splits and, therefore, fewer levels within the hierarchy. With more children per node, there are more available refinable nodes per level and less total aggregate items. This means that there are more detailed specification of ROIs within a level, and less total internal storage space is required. However, having too many children per node contributes to visual clutter and less LODs within the hierarchy. For smaller datasets, a small value of m is useful, because more LODs organize the data more efficiently for interactive operations and introduce more LODs. Lower values do, however, increase the storage space. An illustration of the differences between high and low m values is shown in figure 1.

4.2 Effective, Visualizable Aggregates

In order to allow for a scalable representation, we require a structure that not only organizes the data into an LOD-hierarchy, but allows for appropriate visualization of levels within it. For this reason, it is crucial that we generate aggregate items that are accurately representative to the actual data, as well as usable in various visualization schemes. An item that is accurately representative of the data is one which does not remove semantic information from the dataset.

The R-tree aggregate items are multidimensional rectangular parallelepipeds, or a range of values for each of the total d dimensions, which we will denote throughout the paper as *hyperboxes*. In one dimension, a *hyperbox* is a range of points, or a finite extension of a single point, which is a line segment. In two dimensions, a *hyperbox* is a range of lines, or a finite extension of a single line segment, which is a rectangle. We continue this process of extending lower-dimensional *hyperboxes* in order to generate *hyperboxes* of unlimited dimensionality as shown in figure 2.

Visualization of *hyperboxes* is a researched topic and there already exist methods that project them into two-dimensional screen space without loss of accurate

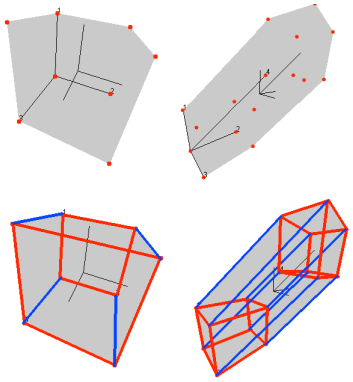

Fig. 2. We show representations of three-dimensional (left) and four-dimensional (right) *hyperboxes* within star coordinates. The red points (top) are the corners of the hyperboxes. We visualize the hyperboxes by filling in the convex hull of the corners with a color, in this case gray. By extending lower-dimensional hyperboxes to higher dimensions (bottom) we can visualize hyperboxes of unlimited possible dimensionality.

representation. The significant characteristic we wish to emphasize at this point is that multidimensional R-trees generate aggregate items which are explicitly defined in d-dimensional space as well as visualizable in two-dimensional space.

These *hyperboxes* are accurately representative aggregate items for visualizing internal levels of a hierarchical data structure. The *hyperboxes* denote where the children of their respective nodes are as well as how sparse or dense the elements within that *hyperbox* are, due to their bounding properties. These characteristics allow the user to draw conclusions about what values within the dimensions of the data are most common as well as how varied the dimensional values are in comparison with each other. Because many *hyperboxes* may be visualized at once, the user can also compare these densities and dimensional correlations with those of other *hyperboxes*, and thus quickly select a region of interest.

4.3 Efficiency for Real-Time Interaction

It is crucial for our application to interactively operate on datasets that are not only large in quantity of elements, but large in dimensionality as well; therefore, generation, queries, and refinement operations must be low in computational complexity. The use of R-trees allows us to execute hierarchical generation and interactive operations very quickly, even with large datasets of many dimensions.

The "bottom-up" generation of R-trees allows them to continuously maintain balance, which in turn provides a great deal of efficiency. Every time a node is split, its children are distributed amongst the new nodes in order to maintain the same depth throughout the R-tree and avoid empty nodes. This property allows for insertions, deletions, and searches to be made in worst-case $O(md \log_m n)$ time for n data elements of d dimensions. Our proposed method to execute queries requires even less time than searches, as we will explain in detail in section 6.2. This method for generation of a hierarchy improves upon Linsen's [6], which does not maintain tree balance and generates many empty nodes. Furthermore, while Linsen's [6] method applies a more accurate automatic generation of clusters, it necessitates specification of a density function and introduction of another preprocessing step to evaluate densities and quanti-

ties of clusters. By taking advantage of R-trees as inherently balanced trees and therefore efficiently alterable data structures, we avoid much of the problems of efficiency encountered in other hierarchical generation schemes.

5 Visualization of Datasets within R-trees

The visualization of massive multidimensional datasets as organized within R-trees requires a transformation from the d dimensions of the R-tree data into the two dimensions available on screen space, as well as effective methods of visualizing both aggregate items and individual data elements.

We examined and implemented two alternative coordinate systems for multidimensional visualization. We show that R-trees are visualizable using an existing method, *hierarchical parallel coordinates*, as well as using a method which builds upon Kandogan's [4], Linsen's [6] and Fua's [5] research, a new technique for *hierarchical star coordinates*. In both cases, we describe how to represent multidimensional data elements as well as bounding *hyperboxes*.

5.1 Hierarchical Parallel Coordinates

Visualization of elements in *hierarchical parallel coordinates* was defined by Inselberg [3], and has been extended to represent multidimensional value ranges, *hyperboxes*, by Fua [5]. In *parallel coordinates*, each dimension is denoted by a single line such that all lines are unique and parallel to each other, and points are represented as polygonal lines with values plotted on each respective dimensional line. To represent *hyperboxes*, we simply plot two data elements in this fashion, the maximum and minimum, and fill the area between both segments, so that we attain a polygon which covers all values within the range of the *hyperbox*.

5.2 Hierarchical Star Coordinates

For single elements within *star coordinates*, the technique is, again, explicitly defined [4], and we propose extending this idea to also represent *hyperboxes*, as Fua [5] did with *parallel coordinates*. The dimensional axes are represented by a set of lines which all emanate from a single point (the *star coordinate* origin). The data elements in *star coordinates* can either be represented as a polygonal line which connects dimensional values, or as a single point which is translated in the direction of each dimensional line by the magnitude of the value. We use the latter. In order to represent the *hyperboxes*, we cannot simply plot the minima and maxima of the range as with *parallel coordinates*, because the area between the minimum point and maximum point no longer accurately represents the range. Instead we introduce a method that plots all possible combinations of the minimum and maximum values in each dimension–the corners of the *hyperbox*– and fills the area between those points. This gives us 2^D points, and we fill the area by calculating the convex hull of these points and constructing its respective convex polygon. This process is illustrated in figure 2.

6 Interactive Operations for Visualization and Analysis

6.1 Refinement Methods for Dynamic Removal of Clutter

The fact that R-trees are an LOD-hierarchy allows for several methods to remove clutter, both programmatically and interactively. Clutter is defined as the ratio of LOD to available screen resolution; thus, LOD corresponds directly to the amount of clutter in the visualization. In an LOD-hierarchy, it is possible to refine down the hierarchy and therefore alter the LOD of the visualization dynamically. Dynamic alteration of LODs, in turn, allows for dynamic removal of clutter.

To be more explicit, refinement means breaking down certain regions within the R-tree into their more detailed components. During traversal the actual process of refining a node within the R-tree involves simply removing its respective *hyperbox* from the visualization and replacing it with the *hyperboxes* or data elements of its children. This provides us with a more accurate visualization, albeit with more elements to visualize on screen space.

Refinement can occur uniformly or non-uniformly as well as programmatically or interactively, with different benefits for each. By giving the user control over which refinement methods are used as well as to what extent they are used, we achieve dynamic alteration of LODs.

Uniform Programmatic We introduce one uniform programmatic method for refinement: a simple breadth-first search (BFS). The first iteration of the BFS algorithm applied to the R-tree returns the root, which is represented by a *hyperbox* that bounds all the lower *hyperboxes* and therefore all the data. The next iteration returns the m nodes that constitute the second level of the hierarchy, the third returns the M^2 nodes of the third level, and so on. In this way, it is possible to alter the LOD uniformly–all elements visualized have the same LOD at all times. This method is beneficial when little is known about the data–the user can draw conclusions about the dataset as a whole and determine which areas are more of interest than others. When the user determines a region within the dataset that is particularly of interest, the ability to refine non-uniformly and interactively becomes crucial.

Non-uniform Interactive To facilitate interactive as well as non-uniform refinement, we introduce methods for region-of-interest (ROI) refinement. The user may construct a query and refine all nodes of the R-tree that are returned by that query. The queries themselves can operate with any number of the available dimensions as specified, which facilitates prioritizing certain dimensions over others as well as certain values within those dimensions over others (explained in detail in the next subsection). In this way, the user can increase the LODs of nodes within an ROI while maintaining lower LODs for nodes that are of less interest. The lower LODs contribute less to clutter, and more screen space is available for the ROI.

6.2 Queries

The user may construct and execute two types of queries on the R-tree: 1) bounded and 2) overlap. Both query methods iterate over nodes of the R-tree and execute comparisons between the constructed query and each node processed. Both also require the same input: a set of 3-tuples, which each specify 1) a dimensional index, from 1 to d inclusively, 2) a value within that dimension, and 3) a margin value. The two types of queries differ in the implementation of these comparisons and the provided semantic benefit.

Bounded Queries *Bounded queries* find nodes whose *hyperboxes* completely encompass the query. The implemented comparison tests whether the node *hyperbox* completely bounds the values and their respective margins within the dimensions specified by the user.

This type of query facilitates interactive searching for programmatically generated clusters of data. Because they test for nodes that completely encompass the query region, this is an effective way for the user to find specific bounded clusters. If the query returns true for a node within the R-tree, that query is accurately represented by a *hyperbox* in the R-tree. In this way the user can find which bounded clusters of data were generated by the R-tree processing and subsequent node-splitting operations. In other words, it is a way for the user to query based on the programmatic clustering of data that occurs during the R-tree generation.

Overlap Queries *Overlap queries* find all nodes which encompass any part, rather than all parts, of the query. The comparison tests whether any of the specified dimensional value/margin pairs overlap in the same dimension as the value/margin pairs of the *hyperbox* of the processed node.

Whereas *bounded queries* allow searching for programmatically generated clusters, *overlap queries* allow searching for any data within the specified range. This technique is useful for drawing conclusions about the data regardless of the internal R-tree structure, and therefore is based on the data elements rather than the data structure.

7 Visual and Interactive Data Exploration and Analysis

7.1 Isolating Outliers

Commonly, real datasets contain outliers. Whether they are errors of the measuring instruments used or exceptional cases within the data, it is essential in visual data analysis to isolate and either extract more detail from or eliminate these outliers. With our system, it is possible to discover outliers quickly and either decrease their significance or refine them in order to examine them more closely. We explain the process by example: discovering outliers in a commonly analyzed dataset of regional wines [8], which has 13 dimensions and 178 elements.

First, we choose a value for m. As 178 elements is a fairly small number of data, we choose a small value for m, 2, in order to increase LODs available. Next, we execute BFS refinements to draw initial conclusions about where outliers may lie. From this step, we can see distributions of values in each dimension. Some are densely packed around certain values, like dimension 10 and dimension 5. The outliers in each dimension are those values which lie outside of the densely packed regions; there are fewer generated aggregate items for them, which indicates that there is less data within them. In order to show the efficacy of removal as well as examination, we remove the outliers in dimension 5 and examine in detail the outliers of dimension 10.

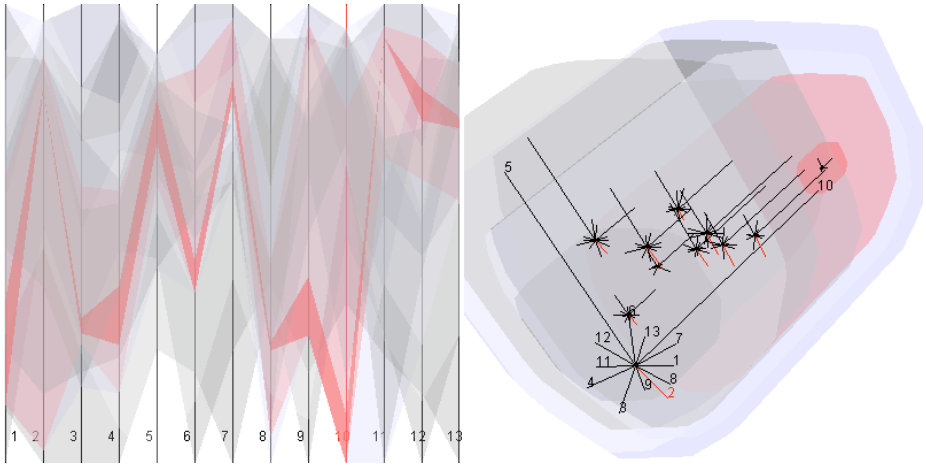

Fig. 3. We color visualized hyperboxes in parallel coordinates (left) and star coordinates (right). By increasing the LOD of a large hyperbox containing sparsely distributed data, we obtained detail of a small hyperbox (at the end of the axis numbered 10) within that region containing densely distributed data (the small red hyperbox).

In order to remove an outlier, we simply use the background color as our query color, in this case white, and construct a query which contains the region of outliers. As explained in section 6, when we are looking for specific elements, like outliers, *overlap queries* are more effective. After running the *overlap query*, the outliers in dimension 5 barely contribute to the visualizations.

We operate similarly to examine the outliers in dimension 10. Instead of the background color, we choose an appropriate color that will "stand out and execute an *overlap query* followed by a number of overlap refinement operations until we obtain the LOD required. After just a few overlap refines, we achieve a very specific outlying region visualized in both the *parallel coordinates* and *star coordinates*, while avoiding clutter due to the region-specific refinement operations. We can then begin correlational analysis, as explained in the next section.

7.2 Examining Correlations

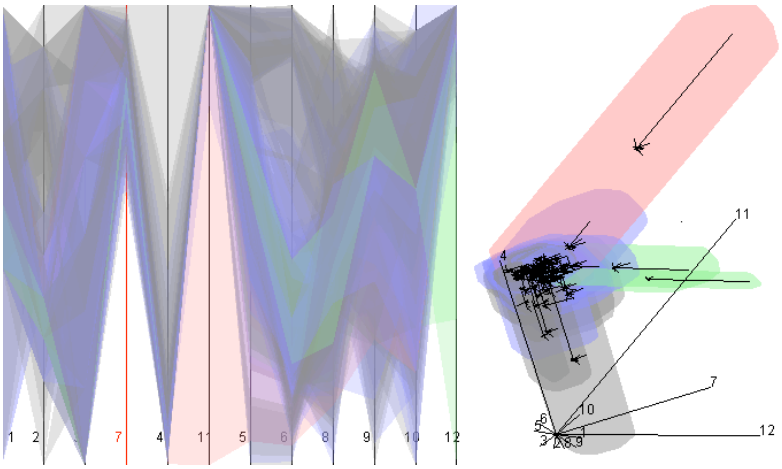

Fig. 4. We show hyperboxes as colored regions in parallel coordinates (left) and star coordinates (right) for a dataset of forest fires. Each numbered line in the star coordinate visualization corresponds to a dimension. These numbered lines can be manipulated in direction and length. If we extrude the line numbered 11, the red hyperbox is extruded more than any other hyperboxes. Therefore, the red hyperbox contains data with a high range of values in dimension 11.

We can examine correlations between dimensions and between individual clusters and elements by performing refinement operations until we achieve the desired LOD in a ROI, and arranging the visualization to show correlations. Again, we explain the process by example, but this time with a dataset of documented forest fires within the northeast region of Portugal [9].

Again, we first choose a number for m, in this case 3 suffices, and execute a number of BFS refinements to obtain initial overviews of the data. As we determine a good ROI, we may begin rearranging the visualization methods in order to analyze correlations. After 2 BFS refinements, we rearrange the *parallel coordinate* axes to observe dimensional correlations: high values in dimension 4 correlate with low values of dimensions 11 and 12. We can more appropriately observe correlations between aggregates or elements in the *star coordinate* visualization. We increase the magnitude and vary the direction of certain dimensions, in this case 4, 7, 11, and 12, shown in figure 7.2. Because we can manipulate these axes interactively, we can also observe to what extent the shape of the aggregates is affected. The blue aggregates are fairly unaffected by manipulation of the axis that represents dimension 11, which means these aggregates have fairly low values in dimension 11. In contrast, the blue aggregates are strongly affected by manipulation of the axis that represents dimension 4, so the data has high values in this dimension. By analyzing the hyperbox shapes, we can draw

further analysis of distribution of values. Very sparse distributions of data can be observed in red, where there is only one aggregate item that is large in visual area, whereas very dense distributions can be observed in blue, where there are many small aggregates and data elements. We conclude that a large quantity of the data has fairly high values in dimension 4 and extremely low values in dimension 11.

8 Drawbacks

One principal drawback to using R-trees is that the hierarchical clusters are generated solely based on proximity, and different clusters are generated when the data is inserted in different orders. The hierarchical clusters are not as accurate as those created with Linsen's [6] or Fua's [5] implementations.

A major drawback of our *hyperbox* visualization method is that it requires $O(2^d)$ complexity to calculate the *hyperbox* corners. The effects are rather detrimental for visualization of 20 or more dimensions, so improved methods would be necessary to provide real-time visualizations at this level of dimensionality. Note that this complexity applies to the visualization, rather than the interactive operations.

We discuss possible improvements to these drawbacks in the next section.

9 Conclusions and Possible Future Research

We have implemented and built upon several existing methods for multidimensional visualization and visualizable hierarchical structuring of multidimensional datasets. We have introduced a novel method to generate an efficient LOD-hierarchy for large, multidimensional datasets using R-trees, we have examined methods to visualize *hyperboxes* and elements within that LOD-hierarchy, and we have examined the use of interactive operations on the data to facilitate analysis. We have used existing visualization schemes, *parallel* and *star coordinates*, in order to introduce a new method for visualizing *hyperboxes*, while retaining the ability to use existing visualization methods as well. Our method for LOD-hierarchy generation provides a great deal of efficiency and functionality in contrast to previous ones, and in combination with the introduced visualization schemes and interactive operations, added benefits for analysis and exploration of data.

A possible improvement to the drawback of complexity mentioned in section 8 could be to apply Linsen's [6] splat-based ray-tracing method to these *hyperboxes*, in which case the complexity would be constrained by screen resolution, rather than the data dimensionality. Another possible improvement, for more accurate hierarchical cluster generation, could be to develop new node-splitting algorithms based on factors other than proximity.

Future implementations of our method could significantly influence areas which use progressive refinement, such as Rosenbaum's [10] technique for device

adaptation or Painter's [11] technique for anti-aliased ray-tracing. As progressive refinement methods require generation of LOD-hierarchies for many types and sizes of multidimensional data, our method provides much of the necessary functionality.

10 Acknowledgements

René Rosenbaum was supported by the German Research Foundation Deutsche Forschungsgesellschaft (DFG), and Mario Hlawitschka was supported in part by NSF grant CCF-0702817. We thank our colleagues from the Institute of Data Analysis and Visualization (IDAV) at UC Davis.

References

1. Chernoff, H.: The use of faces to represent points in k-dimensional space graphically. Journal of the American Statistical Association **68** (1973) 361–368
2. Wright, D.B.: Scatterplot matrices. Encyclopedia of Statistics in Behavioral Science **4** (2005) 1794–1795
3. Inselberg, A.: The plane with parallel coordinates. The Visual Computer **1** (1985) 69–91
4. Kandogan, E.: Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '01, New York, NY, USA, ACM (2001) 107–116
5. Fua, Y.H., Ward, M.O., Rundensteiner, E.A.: Hierarchical parallel coordinates for exploration of large datasets. In: Proceedings of the conference on Visualization '99: celebrating ten years. VIS '99, Los Alamitos, CA, USA, IEEE Computer Society Press (1999) 43–50
6. Linsen, L., Long, T.V., Rosenthal, P., Rosswog, S.: Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. IEEE Transactions on Visualization and Computer Graphics **14** (2008) 1483–1490
7. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, ACM (1984) 47–57
8. Forina, M.: An extendible package for data exploration, classification and correlation (2010)
9. Cortez, P., Morais, A.: A data mining approach to predict forest fires using meteorological data. In: Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence. (2007)
10. Rosenbaum, R., Hamann, B.: Progressive presentation of large hierarchies using treemaps. In: ISVC '09: Proceedings of the 5th International Symposium on Advances in Visual Computing, Berlin, Heidelberg, Springer-Verlag (2009) 71–80
11. Painter, J., Sloan, K.: Antialiased ray tracing by adaptive progressive refinement. In: Proceedings of the 16th annual conference on Computer graphics and interactive techniques. SIGGRAPH '89, New York, NY, USA, ACM (1989) 281–288