The *Gridfit* Algorithm: An Efficient and Effective Approach to Visualizing Large Amounts of Spatial Data

Daniel A. Keim, Annemarie Herrmann
Institute for Computer Science, University of Halle-Wittenberg
Kurt-Mothes-Str. 1, D-06120 Halle (Saale)
{keim, herrmann}@informatik.uni-halle.de

Abstract

In a large number of applications, data is collected and referenced by their spatial location. Visualizing large amounts of spatially referenced data on a limited-size display often results in poor visualizations due to the high degree of overplotting of neighboring data points. In this paper, we introduce a new approach to visualizing large amounts of spatially referenced data. The basic idea is to intelligently use the unoccupied pixels of the display instead of overplotting data points. After formally describing the problem, we present two solutions which are based on (1) placing overlapping data points on the nearest unoccupied pixel and (2) shifting data points along a screen-filling curve (e.g., Hilbert-curve). We then develop a more sophisticated approach called Gridfit, which is based on a hierarchical partitioning of the data space. We evaluate all three approaches with respect to their efficiency and effectiveness, and show the superiority of the Gridfit approach. For measuring the effectiveness, in addition to comparing the resulting visualizations we introduce mathematical effectiveness criteria measuring properties of the generated visualizations such as distance- and position-preservation.

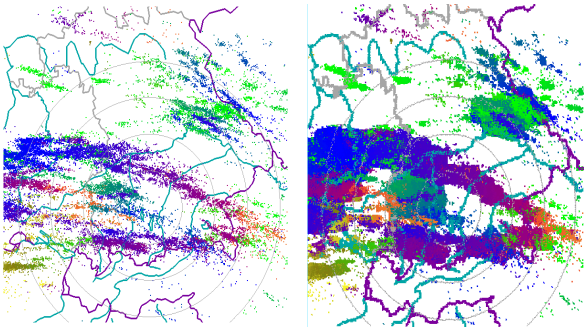
Keywords: *Visualizing Large Data Sets, Visualizing Spatially Referenced Data, Visualizing Geographical Data, Interfaces to Databases*

1. Introduction

There are a large number of applications where spatial data arise. Examples include weather data such as temperature, rainfall, wind-speed, etc. measured at a large number of locations, use of connecting nodes in telephone business, load of a large number of internet nodes at different locations, air pollution of cities, etc. Visualizing this type of information requires representing the data values (e.g., air pollution) and their spatial location. A natural way to visualize the data is, for example, to represent the data values as colored pixels on a screen where the position on the screen directly correlates to the spatial location of the data. Since the spatial locations of the data are not uniformly distributed in a rectangular data space, however, the display will usually be sparse in some regions while in other regions of the display a high degree of overplotting occurs. Consider, for example, the air pollution example from above. Cities (e.g., all cities with more than 10.000 inhabitants) cluster in few places (such as North America, Europe, Asia, etc.) while large portions of the earth are only sparsely populated. In addition, if the data is presented on a world map the large portion of the screen which corresponds to oceans is not used, while only a few data values corresponding to European cities may be displayed. This results in a loss of large amounts of potentially important information.

A simple but intuitive idea to avoid this problem is to present data values which would be overplotted at nearby unoccupied positions. If the data values are presented in an appropriate way, the visualization naturally reflects the spatial location of the data and the loss of information can be avoided. In addition, as many pixels of the display as necessary are used while still reflecting the spatial nature of the data. In Figure1, we show a data set of lightning strikes in southern Germany. On the left, the data set is presented with overlapping data points, whereas on the right, overlapping data points are placed on unoccupied pixels close to their original position, thereby avoiding the information to be lost. The goal of the this paper is to introduce an efficient algorithm for visualizing spatially referenced data in an effective way.

Visualization technology has already been used for the exploration of large amounts of data in the past. Examples for visual data exploration approaches include geometric projection techniques such as prosection matrices [FB94] and parallel coordinates [Ins85, ID90], icon-based techniques (e.g., [PG88, Bed90]), hierarchical techniques (e.g., [LWW 90, RCM91, Shn92]), graph-based techniques (e.g., [EW93, BEW95]), pixel-oriented techniques (e.g.,[Kei94, KK94, KKA95]), and combinations hereof ([Asi85, AS94]). In general, the visualization techniques are used in conjunction with some interaction techniques (e.g.,[BMMS91, AWS92, ADLP95]) and sometimes also some distortion techniques [SB 94, LRP95]. The research also resulted in data exploration and analysis systems which implement some of the mentioned techniques. Examples include statistical data analysis packages such as SPlus/Trellis [BCW88], XGobi [SCB92], and Data Desk [WUT95], visualization oriented systems such as ExVis [GPW89], XmdvTool [War94, MW95], and IBM's Parallel Visual



a. With Overlap b. Without Overlap

Figure1: Lightening Strike Data

Explorer, as well as database oriented systems such as TreeViz [Shn92], the Information Visualization and Exploration Environment (IVEE) [AW95], and the *VisDB* system [KK95].

The techniques presented in this paper can be seen as pixel-oriented techniques. The principle idea of pixel-oriented techniques is to use each pixel of the display to represent one data value (e.g., spiral [KK94], recursive pattern [KKA95], and circle segments techniques [AKK96]). In contrast to previous pixel-oriented techniques, the techniques presented in this paper deal with displaying spatially referenced data, which means that the data have real world 2D coordinates and some additional data values. The goal of the new techniques is to present the data on the display such that the (absolute and relative) position of the data points and their distance is preserved as much as possible. These objectives can be described formally as global optimization goals of the mapping function between original and new position of the data points (cf. subsection 2.1). In subsection 2.2, we introduce a first algorithm which implements this mapping function by placing all data points to their original position unless the position is already occupied by some other data point. In a second step, all remaining data points are placed on the nearest unoccupied pixel. A second algorithm also presented in subsection 2.2 places the data points which could not be placed in the first step by shifting the data points along a screen-filling curve (e.g., Hilbert- or Z-curve). A third and more sophisticated algorithm called *Gridfit* is based on a hierarchical partitioning of the data space into subregions (cf. section3). The partitioning into subregions is determined such that each region in the hierarchical structure (in our case a variant of the quadtree [Sam84]) allows a visualization of the data which belong to the region. An experimental evaluation using a number of different data sets shows the superior efficiency (cf. subsection 4.1) and effectiveness (with respect to the optimization criterions, cf. subsection 4.2) of the *Gridfit* approach. A visual comparison (cf. subsection 4.3) confirms these results. In section5, we finally discuss our three approaches and present an alternative idea which is based on clustering the data for an effective display.

2. The Problem and Two Solutions

In this section, we formally describe the problem and present two solutions — the nearest-neighbor algorithm and the curve-based algorithm.

2.1 Formal Description of the Problem

The problem of visualizing spatially referenced data can be described as a mapping between the multiset of original positions and the set of display positions. Let A be the data set of original positions

$$A = \{a_0, \ldots, a_{N-1}\} \text{ with } a_i = (a_i^x, a_i^y)$$

where it is possible that $a_i = a_j$ for an arbitrary i and j. Let the data space (or better screen space) $DS \subseteq Z^2$ be defined as $DS = \{0, \ldots, x_{max} - 1\} \times \{0, \ldots, y_{max} - 1\}$ where x_{max} and y_{max} are the maximal extension of the screen.

The goal of the algorithm is to determine a solution set $S = \{s_0, \ldots, s_{N-1}\}$ of new positions with s_i being the new position of a_i such that

$$i \neq j \Rightarrow s_i \neq s_j \qquad \forall i, j \in \{0 \ldots N-1\}$$

and the resulting visualization should be as similar as possible to the visualization of the original data. The similarity may be defined by the absolute distance of the data points to their original positions or by the relative distance or relative position between data points. This leads to the following optimization goals:

(1) absolute position-preservation

$$\sum_{i=0}^{N-1} d(a_i, s_i) \rightarrow min$$

(2) relative position-preservation

$$\sum_{i=0}^{N-1} \sum_{j=0, j \neq i}^{N-1} d(s_i, s_j) - d(a_i, a_j) \rightarrow min$$

(3) relative distance-preservation

$$\sum_{i=0}^{N-1} \sum_{j=0, j \neq i}^{N-1} \frac{d(s_i, s_j)}{d(a_i, a_j)} \rightarrow min$$

where d is an arbitrary distance metric in 2D such as

$$d(a_i, a_j) = |a_i^x - a_j^x| + |a_i^y - a_j^y|$$

or the Euclidean metric

$$d(a_i, a_j) = \sqrt{(a_i^x - a_j^x)^2 + (a_i^y - a_j^y)^2}.$$

The optimization goals make sure that as little as possible of the spatial information is lost. Which of the three optimization goals is most important and should be fulfilled first depends on the application. The formal description of the problem indicates that finding a mapping which fulfills the above properties is a typical optimization problem. Most typical optimization problems are NP-complete and we assume that our mapping problem also belongs to the class of NP-complete problems. A formal proof, however, has not yet been found.

2.2 Two Solutions

Both algorithms work in two steps. In step one, all data points a_i are placed on the display unless their position is already occupied by some other data point. In the second step, a new position which is as close as possible to the original position is determined for the remaining data points.

More formally, the general idea of the algorithm can be described as follows. The set S is the set of data points with unique positions and the set of points T is a temporary set of data points to be placed in the second step.

Step 1:

- $S_0 = \{a_0\}$, $T_0 = \emptyset$
- $$S_{i+1} = \begin{cases} S_i \cup \{a_{i+1}\} & if\ (a_{i+1} \neq s)\quad \forall s \in S_i \\ S_i & otherwise \end{cases}$$

$$T_{i+1} = \begin{cases} T_i & if\ (a_{i+1} \neq s)\quad \forall s \in S_i \\ T_i \cup \{a_{i+1}\} & otherwise \end{cases}$$

Step 2:

- $S_0' = S_{N-1}$, $T_0' = T_{N-1}$
- $T_{i+1}' = T_i' - \{a\}$ *for an arbitrary* $a \in T_i'$

  $S_{i+1}' = S_i' \cup \{a_{new}\}$

  where a_{new} is determined differently by the two algorithms.

Nearest-Neighbor Algorithm

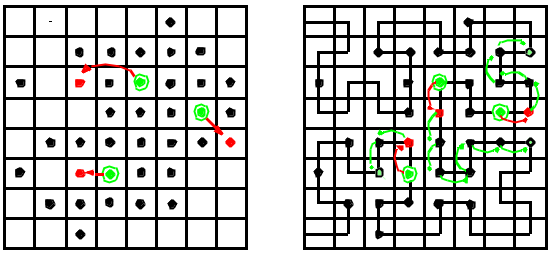
In case of the nearest-neighbor (NN) algorithm, the new position for the data points which have not been placed in the first step is determined by simply placing the data points on the nearest unoccupied positions (cf. Figure 2a). More formally, the new position a_{new} of a data point a is determined as

$$a_{new} = \{s' \in DS \setminus S_i \mid d(a, s') \leq d(a, s)\ \ \forall s \in DS \setminus S_i\}.$$

An advantage is that the new position can usually be determined locally and therefore, in general, the algorithm works quite efficiently. For a very dense display, however, the efficiency and effectiveness suffer from the fact that the new position may be rather far from the original position (cf. section 4).

Curve-based Algorithm

In case of the curve-based algorithm, the new position for the data points which have not been placed in the first step is determined by computing the nearest unoccupied positions on a given screen-filling curve and shifting all data points between the overlapping data point a and the unoccupied position in that direction (cf. Figure2b). Screen-filling curves such as the Hilbert-curve [Pea90, Hil91] or Z-curve [Mor66] provide a bijective mapping between a position on a one-dimensional line and a two-dimensional position. The advantage is that data points which are close in 1D are usually mapped to nearby points in 2D and vice versa. The idea of the curve-based algorithm is to shift the data points along the screen-filling curve (1D) which in general will also correspond to nearby positions in 2D.



a. NN-Algorithm b. Curve-based Algorithm

Figure2: NN- and Curve-based Algorithms

More formally, let C: $\{0, \ldots, x_{max} \cdot y_{max} - 1\} \rightarrow DS$ be a bijective curve function where $C(i) = a$ gives the 2D position of the i-th position on the curve and

$$C^{-1}: DS \rightarrow \{0, \ldots, x_{max} \cdot y_{max} - 1\}$$

the inverse function. Then, the one-dimensional distance to the nearest unoccupied position on a given screen-filling curve can be computed as

$$\tilde{j} = \begin{cases} j_1 & if\ |j_1| \leq |j_2| \\ j_2 & otherwise \end{cases} \qquad \wedge \oplus = signum(\tilde{j})$$

where $$j_1 = min\left\{j \in N_0 \mid C(C^{-1}(a) + j) \in DS \setminus S_i\right\}$$

$$j_2 = -min\left\{j \in N_0 \mid C(C^{-1}(a) - j) \in DS \setminus S_i\right\}$$

Then, all data points between a and $C(C^{-1}(a) + \tilde{j})$ have to be shifted along the space-filling curve. This can be described as

$$S_{i+1} = S_i - \{b\} \cup \{b' \in DS \mid b' = C(C^{-1}(b) \oplus 1)\}$$
$$\forall b \in \{b \in DS \mid C^{-1}(a) < C^{-1}(b) < C^{-1}(a) + \tilde{j}\}.$$

The new position a_{new} of the data point a is determined as

$$a_{new} = \{s' \in DS \mid s' = C(C^{-1}(a) \oplus 1)\}.$$

The efficiency and effectiveness of the curve-based algorithm are discussed in section 4.

3. The *Gridfit* Algorithm

As a first evaluation showed, neither the nearest-neighbor nor the curve-based algorithm provide visualizations which are sufficiently position- and distance-preserving (cf. subsection 4.2). In addition, for a high degree of non-unique data points the performance degenerates (cf. subsection4.1). We therefore developed a more sophisticated algorithm called *Gridfit* which provides a better efficiency and effectiveness (visually as well as measured) than the other two algorithms.

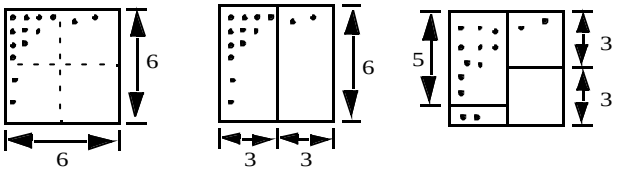


Figure3: Partitioning of a Node

3.1 Basic Idea

The basic idea of the *Gridfit* algorithm is to hierarchically partition the data space. In each step, the data set is partitioned into four subsets containing the data points which belong to four equally-sized subregions. If the number of data points in some of the four subregions exceeds the size of the subregion, we have to determine the new extends of the four subregions (without changing the four subsets of data points) such that the data points of each of the subsets can be visualized in the corresponding subregion. For an efficient implementation of the algorithm, a quadtree-like data structure is used to manage the required information and to support the recursive partitioning process. The partitioning process works as follows. Starting with the root of the quadtree, in each step the data space is partitioned into 4 subregions. The partitioning is made such that the area occupied by each of the subregions (in pixels) is larger than the number of pixels belonging to the corresponding subregion (cf. Figure3).

3.2 Formal Description

The quadtree data structure consists of a root node (B^0), which is the starting point of the tree, and each non-leaf node of the quadtree has four son nodes. Let B^i denote a node on hierarchy level *i*, then

$E(B) = (x_{min}, x_{max}, y_{min}, y_{max})$
denotes the extend of a node *B*,

$A(B) = (B.x_{max} - B.x_{min}) \cdot (B.y_{max} - B.y_{min})$
the area occupied by node *B* (in pixels), and

$P(B)$ the number of data points which belong to the equally-sized subregion corresponding to node *B*.

The root node B^0 of a normalized data space can be defined as $E(B^0) = (0, 1, 0, 1)$, $A(B^0) = 1$, and $P(B^0) = N$. The son nodes $(B_1^i, B_2^i, B_3^i, B_4^i)$ of a node B^{i-1} and the extend of the corresponding subregions can be defined recursively as

$$E(B_1^i) = (B_{x_{min}}^{i-1}, x^i, B_{y_{min}}^{i-1}, y_1^i)$$

$$E(B_2^i) = (x^i, B_{x_{max}}^{i-1}, B_{y_{min}}^{i-1}, y_2^i)$$

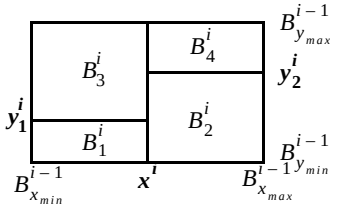


Figure4: Partitioning of a Node B^{i-1}

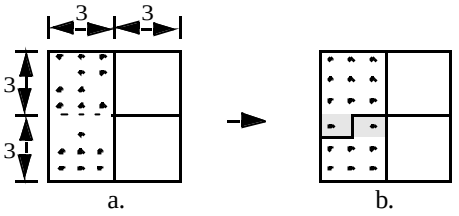


Figure5: Split of the Dividing Line

$$E(B_3^i) = (B_{x_{min}}^{i-1}, x^i, y_1^i, B_{y_{max}}^{i-1})$$

$$E(B_4^i) = (x^i, B_{x_{max}}^{i-1}, y_2^i, B_{y_{max}}^{i-1}).$$

where (x^i, y_1^i, y_2^i) describe the partitioning of the node. The partitioning of a node on the next hierarchy level is done by first partitioning the node vertically (at x^i) and then, both halves are partitioned horizontally (at y_1^i and y_2^i). The partitioning is therefore given by the three values x^i, y_1^i, and y_2^i (cf. Figure4), which are determined as

$$x^i = \frac{B_{x_{min}}^{i-1} + B_{x_{max}}^{i-1}}{2} + \Delta x$$

where $\Delta x = min\{x \in 9 \mid P(B_1) + P(B_3) \le A(B_1) + A(B_3) \wedge P(B_2) + P(B_4) \le A(B_2) + A(B_4)\}$

$$y_1^i = \frac{B_{y_{min}}^{i-1} + B_{y_{max}}^{i-1}}{2} + \Delta y_1 \text{ and } y_2^i = \frac{B_{y_{min}}^{i-1} + B_{y_{max}}^{i-1}}{2} + \Delta y_2$$

where $\Delta y_1 = min\{y \in 9 \mid P(B_1) \le A(B_1) \wedge P(B_3) \le A(B_3)\}$,

$\Delta y_2 = min\{y \in 9 \mid P(B_2) \le A(B_2) \wedge P(B_4) \le A(B_4)\}$.

3.3 Implementation Details

In some rare cases, the partitioning as described so far runs into problems. Even if the condition $P(B) \le A(B)$ is fulfilled for node *B*, there may not be a straight-line partitioning of *B* into four subregions such that all data points can be visualized in the corresponding subregions. Consider, for example, the data presented in Figure 5a. In this case,

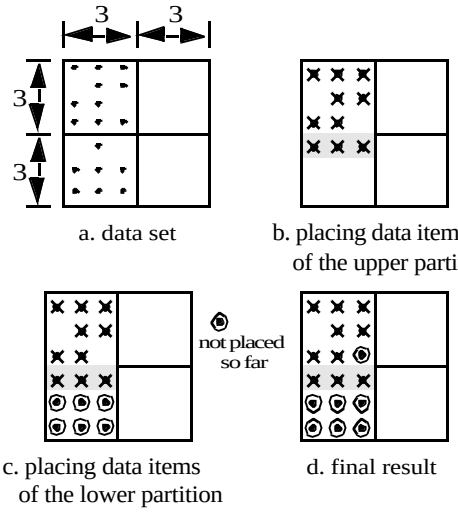


Figure6: Placing Data Points on Dividing Lines

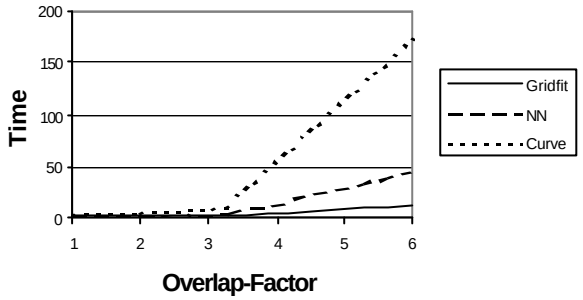
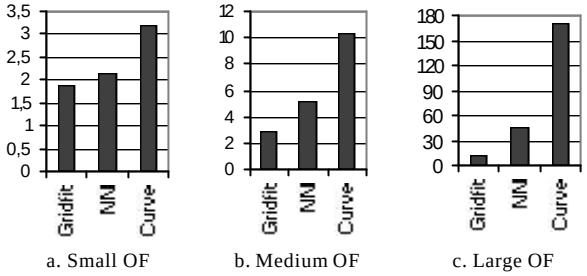


Figure7: Comparison of the Efficiency



a. Small OF b. Medium OF c. Large OF

Figure8: Efficiency for Different Data Sets

the partitioning is only possible if the dividing line is split and used by both, the upper and the lower partition (cf. Figure5b). In our implementation of the *Gridfit* algorithm, we solve this problem by making the dividing line part of both partitions. In placing pixels, however, we have to be careful if the dividing line is used by both partitions. Let us consider the situation in Figure6a. If the data points of the upper partition are placed the dividing line is already completely filled (cf. Figure6b). This however leads to the problem that one data point in the lower partition can not be placed any more (cf. Figure6c). Our implementation solves this rarely occurring problem by storing the data points which can not be placed in the first run and placing them to the nearest unoccupied position in a second run (cf. Figure6d).

4. Comparison and Evaluation

All algorithms introduced in sections 2 and 3 have been implemented as part of the ***VisualPoints*** system. The system is implemented in C++ and running under HP-UX and LINUX. Public-domain versions of the system are available via *http://www.informatik.uni-halle.de/~keim*.

We used the ***VisualPoints*** system to evaluate and compare the different algorithms. We evaluated not only the efficiency but also their mathematically defined absolute and relative position- and distance-preservation (cf. subsection 2.1) and their visual effectiveness. For the experiments, we used a number of different data sets including two different versions of a world map containing simulated data, a real data set of lightning strikes, and a synthetic data set which has been designed to demonstrate effects and artifacts of the algorithms.

4.1 Efficiency

The theoretical time and space complexities of the three algorithms are all similar. In all three cases, the space complexity is $O(N)$ and the time complexity is between $O(N)$ in the best case and $O(N^2)$ in the worst case.

An experimental evaluation based on different realistic data sets, however, clearly shows the advantage of the *Gridfit* algorithm. Since the number of data points, which on the average would be positioned at the same position plays an important role in all three algorithms, we used data sets with a different overlap-factor (OF). In Figure7, we provide the performance curves for a varying overlap-factor. It is clear that for all algorithms the time increases significantly with an increasing overlap-factor. In Figure8, we present the time performance of the three algorithms for a small, medium, and large overlap-factor. For all three data sets, the *Gridfit* algorithm is significantly faster than the other two algorithms, and the nearest-neighbor algorithm is faster than the curve-based algorithm. The speed-up of the *Gridfit* algorithm increases with the overlap-factor and reaches a factor of about 13 over the curve-based algorithm and a factor of about 4 over the nearest-neighbor algorithm.

4.2 Effectiveness

More important than the efficiency, however, is the effectiveness of the generated visualizations. The effectiveness can be determined by visually comparing the generated visualizations but it can also (at least partially) be determined mathematically according to our optimizations goals. Before presenting the visual comparison in subsection 4.3, in this subsection we briefly present the measured effectiveness, i.e. the absolute and relative position and distance preservation (cf. the definitions in subsection 2.1) for an L^1 distance function.

In Figure9, we present the absolute position measure of the three algorithms depending on the overlap-factor. Figure9 clearly shows that the *Gridfit* algorithm provides a smaller average deviation from the original position than

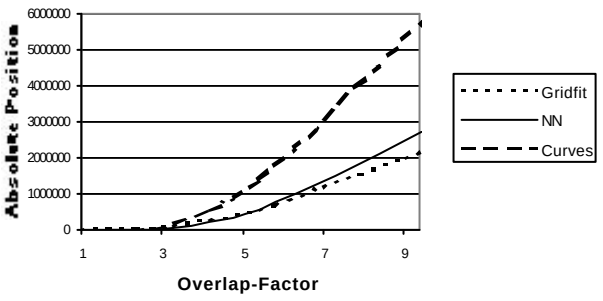


Figure9: Absolute Position

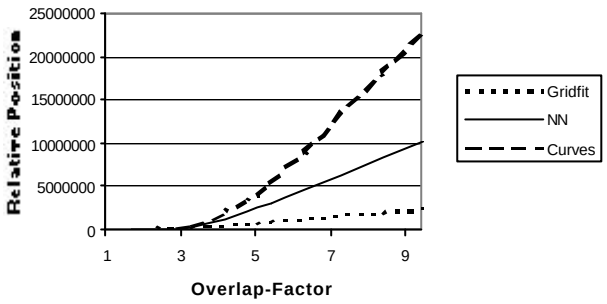


Figure10: Relative Position

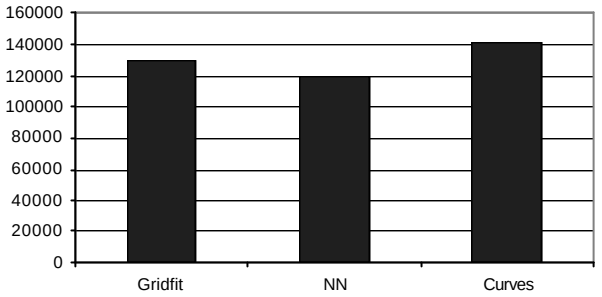


Figure11: Relative Distance

the nearest-neighbor and curve-based algorithms, especially for higher overlap factors. This can also be confirmed by our visual comparison which is presented in subsection 4.3. Figure10 presents the development of the relative position measure. The relative position measure determines how good the relative position of the data points is preserved in the visualization (a smaller value means a better relative position preservation). Here, the advantage of the *Gridfit* algorithm over the nearest-neighbor and curve-based algorithms becomes even more impressive. The improvement is up to 390% over the nearest-neighbor algorithm and up to 870% over the curve-based algorithm. Figure11 shows the relative distance measure of the three algorithms for a high overlap-factor. In this case, all three algorithms provide approximately the same performance, and the value of the *Gridfit* algorithm is between the values of the nearest-neighbor and curve-based algorithms.

4.3 Visual Comparison

All formal effectiveness measures as defined by the absolute and relative position and distance are of limited value if they do not correspond to improvements in the generated visualizations. In this section, we therefore provide a visual comparison of the three techniques, which confirms the measured effectiveness criteria presented in the previous subsection.

Our first comparison uses a greyscale world data map with simulated points distributed over the surface of dry land. Figure 13 shows the result of visualizing the data using our nearest-neighbor, curve-based, and *Gridfit* algorithms on two different resolutions. The visualizations clearly show the advantages and disadvantages of the three approaches. The nearest-neighbor algorithm provides nice results, at least for the portion of data which can be placed at their original positions in the first step of the algorithm. The contours of the continents are clearly visible in their original size. All data points which can not be placed in the first step, however, do not show any structure (cf. right portion of Figure13a). In case of the curve-based algorithm, the continents are rather distorted and their contours are barely visible. In the lower resolution picture (cf. right portion of a Figure13b), there even seems to be no similarity to the original image. This result corresponds to the result of our theoretical effectiveness comparison (cf. subsection 4.2) which showed that the curved-based algorithm is worse than the other two approaches. The visualization generated by the *Gridfit* algorithm (cf. right portion of a Figure13c) also confirms the results of the theoretical effectiveness comparison, namely that the *Gridfit* algorithm provides significantly better results than the other two approaches. In contrast to the nearest-neighbor algorithm, the *Gridfit* algorithm enlarges and distorts the contours such that all data points can be placed close to their original positions. As a result, the visualization retains the spatial locality of the data points as much as possible, which results not only in a better (absolute and relative) position-preservation but also in a better visual representation of the data.

To analyze the properties of the three algorithms in more detail, for the second test we designed a synthetic data set consisting of a number of objects with different properties (cf. Figure14a). Besides a few simple objects such as circles, straight and curved lines, and vertical and horizontal bars, we also use text as well as rectangular and circular patterns. The visualizations generated by our three algorithms allow some interesting insights into the properties of the algorithms. Again, the curve-based algorithm provides the poorest results for all types of objects (cf. Figure14c). The nearest-neighbor algorithm provides rather good results, especially for the text (cf. Figure 14b). The main problem of the algorithm, however, are the rectangular and circular patterns which show the original data in the center but no structure for the overlapping data points positioned in the second step of the algorithm. Since patterns are very important in data exploration, this turns out to be a major drawback of the nearest-neighbor algorithm. In contrast, the *Gridfit* algorithm performs nicely for all types of objects (cf. Figure14d). Note that the rectangular and circular patterns are enlarged which is a desired effect in preserving the spatial locality.

```
void ClusterAlgorithm(D)
{     C = DetermineClusters(D); // C is array of clusters
      for (int i=1; i<=|C|; i++)
            {if (Size(C[i]) <= |C[i]|)
                  {     ES = {C[j] | Neighbor(C[i], C[j])
                                    && Size(C[j]) > |C[j]|}
                        Sort ES according to (Size(C[j]) - |C[j]|)
                        for (int k=1; Size(C[i]) <= |C[i]|; k++)
                        {     ReduceSize(ES[k]);
                              ExpandSize(C[i])
                        }
                  }
            }
}
```

Figure12: Cluster-based Algorithm

The third test shows a series of four *Gridfit* visualizations of a colored world map at different resolutions — from a very high resolution visualization without overlapping data points in Figure15a to a low resolution version in Figure15d. It is interesting to consider the development of the distortion of the continents. In the first generated visualization (cf. Figure 15b), the continents are more compact but their shape is still pretty much unchanged. In the next version (cf. Figure15c), the continents start to get distorted and the partitioning lines of the quadtree data structure become visible (e.g., in the blue region corresponding to the African continent). In the final visualization (cf. Figure15d), basically all pixels of the display are used and the continents are distorted to fill all the screen space.

5. Conclusions

In this paper, we presented three algorithms for visualizing large amounts of spatially referenced data. The algorithms avoid the problem of losing information by overplotting data points. Instead, they map each data point to one pixel of the display and try to preserve the spatial locality (position and distances) as much as possible. We used a number of test data sets to evaluate and compare the three algorithms. It turns out that the *Gridfit* algorithm significantly outperforms the other two algorithms with respect to their efficiency and their (mathematical and visual) effectiveness. In our future work, we will investigate other approaches for visualizing spatially referenced data. One idea is to use a cluster-based approach which first tries to identify the clusters in the data and then transforms the clusters to retain their spatial properties as much as possible. We implemented a first version of this algorithm (cf. Figure12). The generated visualizations, however, turned out to be worse than the visualizations generated by the nearest-neighbor algorithm, and the efficiency is worse by orders of magnitude.

Acknowledgements

We are thankful to R. Gansel for implementing the ***VisualPoints*** system and for doing most of the experiments.

References

[ADLP95] Anupam V., Dar S., Leibfried T., Petajan E.: *'DataSpace: 3-D Visualization of Large Databases'*, Proc. Int. Symp. on Information Visualization, Atlanta, GA, 1995, pp. 82-88.

[AKK96] M. Ankerst, D. A. Keim, H.-P. Kriegel: *'Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Sets'*, VISUALIZATION '96, HOT TOPIC SESSION, San Francisco, CA, 1996.

[AS94] Ahlberg C., Shneiderman B.: *'Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays'*, Proc. ACM CHI Int. Conf. on Human Factors in Computing, Boston, MA, 1994, pp. 313-317.

[Asi85] Asimov D.: *'The Grand Tour: A Tool For Viewing Multidimensional Data'*, SIAM Journal of Science & Stat. Comp., Vol. 6, 1985, pp.128-143.

[AWS92] Ahlberg C., Williamson C., Shneiderman B.: *'Dynamic Queries for Information Exploration: An Implementation and Evaluation'*, Proc. ACM CHI Int. Conf. on Human Factors in Computing, Monterey, CA, 1992, pp.619-626.

[AW95] Ahlberg C., Wistrand E.: *'IVEE: An Information Visualization and Exploration Environment'*, Proc. Int. Symposium on Information Visualization, Atlanta, GA, 1995.

[BCW88] Becker R., Chambers J. M., Wilks A. R.: *'The New S Language'*, Wadsworth & Brooks/Cole Advanced Books and Software, Pacific Grove, CA, 1988.

[Bed90] Beddow J.: *'Shape Coding of Multidimensional Data on a Mircocomputer Display'*, Proc. Visualization'90, San Francisco, CA, 1990, pp.238-246.

[BEW95] Becker R. A., Eick S.G., Wills G.J.: *'Visualizing Network Data'*, IEEE Transactions on Visualizations and Graphics, Vol.1, No.1, 1995, pp.16-28.

[BMMS91] Buja A., McDonald J.A., Michalak J., Stuetzle W.: *'Interactive Data Visualization Using Focusing and Linking'*, Proc. Visualization'91, San Diego, CA, 1991, pp.156-163.

[EW93] Eick S., Wills G. J.: *'Navigating Large Networks with Hierarchies'*, Proc. Visualization'93, San Jose, CA, 1993, pp.204-210.

[FB94] Furnas G. W., Buja A.: *'Prosections Views: Dimensional Inference through Sections and Projections'*, Journal of Computational and Graphical Statistics, Vol.3, No.4, 1994, pp.323-353.

[GPW89] Grinstein G, Pickett R., Williams M. G.: *'EXVIS: An Exploratory Visualization Environment'*, Proc. Graphics Interface '89, London, Ontario, Canada, 1989.

[Hil91] Hilbert D.: *'Über stetige Abbildung einer Linie auf ein Flächenstück'*, Math. Annalen, Vol.38, 1891, pp.459-460.

[ID90] Inselberg A., Dimsdale B.: *'Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry'*, Proc. Visualization '90, San Francisco, CA, 1990, pp.361-370.

[Ins85] Inselberg A.: *'The Plane with Parallel Coordinates, Special Issue on Computational Geometry'*, The Visual Computer, Vol. 1, 1985, pp.69-97.

[Kei94] Keim D. A.: *'Visual Support for Query Specification and Data Mining'*, Ph.D. -thesis, University of Munich, July 1994, Shaker-Publishing Company, Aachen, Germany, 1995, ISBN3-8265-0594-8.

[KK94] Keim D. A., Kriegel H.-P.: *'VisDB: Database Exploration using Multidimensional Visualization'*, Computer Graphics & Applications, Sept. 1994, pp.40-49.

[KK95] Keim D. A., Kriegel H.-P.: *'VisDB: A System for Visualizing Large Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, 1995, p.482.

[KKA95] Keim D. A., Kriegel H.-P., Ankerst M.: *'Recursive Pattern: A Technique for Visualizing Very Large Amounts of Data'*, Proc. Visualization '95, Atlanta, GA, 1995, pp. 279-286.

[LWW90] LeBlanc J., Ward M. O., Wittels N.: *'Exploring N-Dimensional Databases'*, *Proc.* Visualization '90, San Francisco, CA, 1990, pp.230-237.

[LRP95] Lamping J., Rao R., Pirolli P.: *'A Focus + Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies'*, Proc. ACM CHI Conf. on Human Factors in Computing (CHI95), 1995, pp. 401-408.

[Mor66] Morton G. M.: *'A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing'*, IBMOttawa, Canada, 1966.

[MW95] Motro A. R., Ward M. O.: *'High Dimensional Brushing for*

Interactive Exploration of Multivariate Data', Proc. Visualization '95, Altanta, GA, 1995, pp.271-278.

[Pea90] Peano G.: *'Sur une courbe qui remplit toute une aire plaine'*, Math. Annalen, Vol.36, 1890, pp.157-160.

[PG88] Pickett R. M., Grinstein G. G.: *'Iconographic Displays for Visualizing Multidimensional Data'*, Proc. IEEE Conf. on Systems, Man and Cybernetics, IEEE Press, Piscataway, NJ, 1988, pp.514-519.

[RCM91] Robertson G., Card S., Mackinlay J.: *'Cone Trees: Animated 3D Visualizations of Hierarchical Information'*, Proc. ACM CHI Int. Conf. on Human Factors in Computing, 1991, pp. 189-194.

[Sam 84] Samet H.: *'The Quadtree and Related Hierarchical Data Structures'*, ACM Computing Surveys, Vol.16, No.2, 1984, pp.187-260.

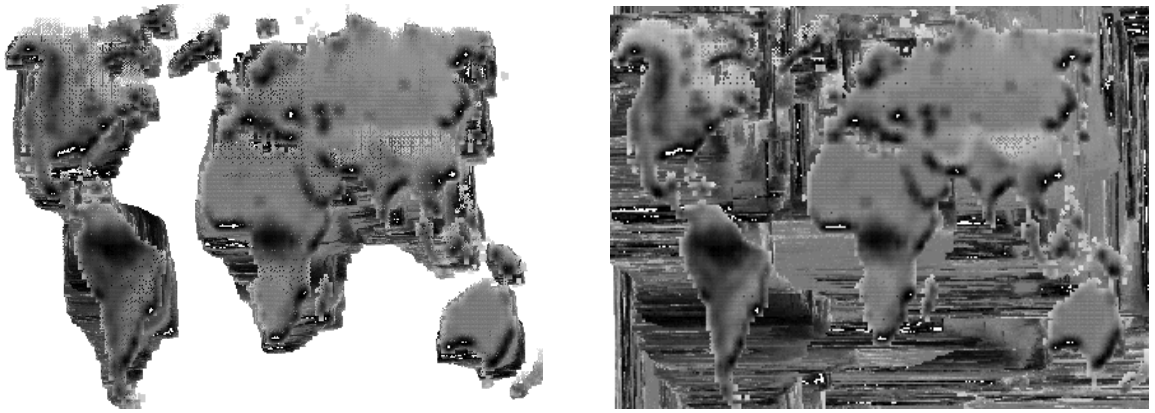
[SB94] Sarkar M., Brown M.: *'Graphical Fisheye Views'*, Communications of the ACM, Vol. 37, No. 12, 1994, pp.73-84.

[SCB92] Swayne D.F., Cook D., Buja A.:*'User's Manual for XGobi: A Dynamic Graphics Program for Data Analysis'*, Bellcore Technical Memorandum, 1992.

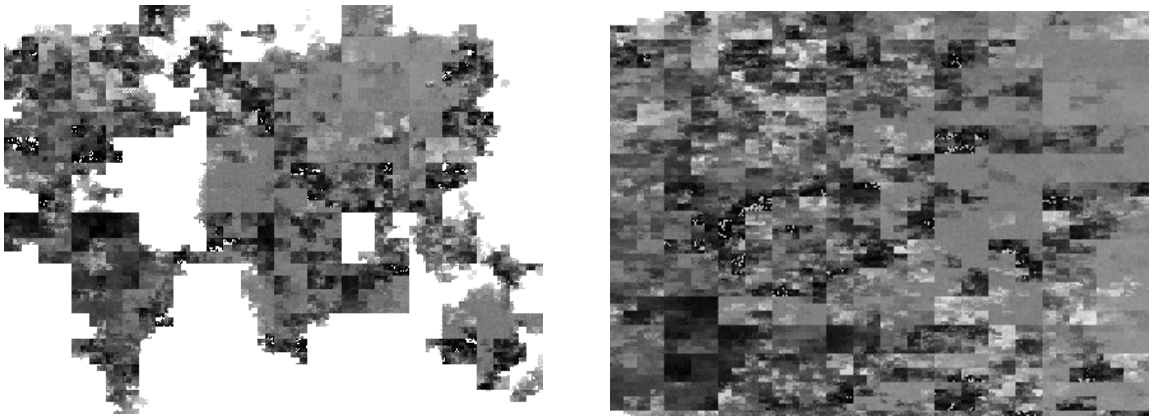
[Shn92] Shneiderman B.: *'Tree Visualization with Treemaps: A 2D Space-Filling Approach'*, ACM Transactions on Graphics, Vol. 11, No.1, 1992, pp.92-99.

[War94] Ward M. O.: *'XmdvTool: Integrating Multiple Methods for Visualizing Multivariate Data'*, Proc. Visualization '94, Washington, DC, 1994, pp.326-336.

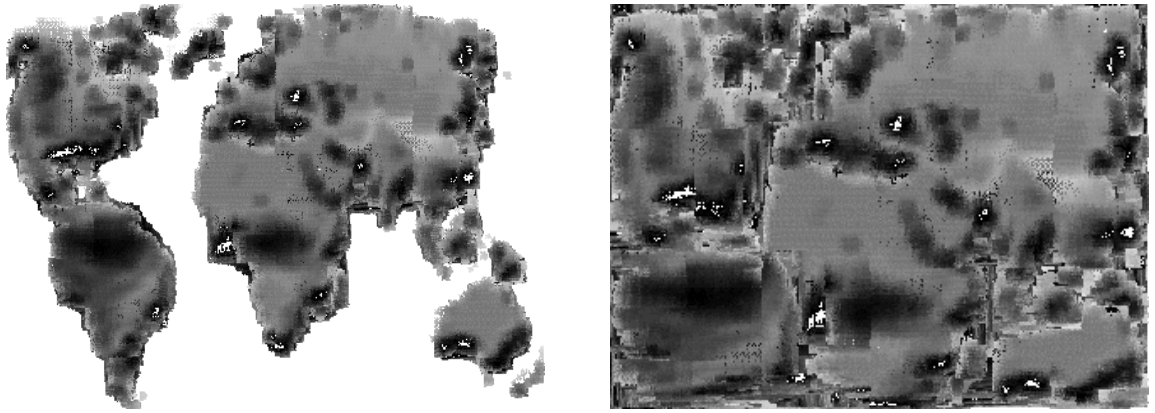
[WUT95] Wilhelm A., Unwin A. R., Theus M.: *'Software for Interactive Statistical Graphics - A Review'*, Proc. Int. Softstat '95 Conf., Heidelberg, Germany, 1995.



a. Nearest-Neighbor Algorithm

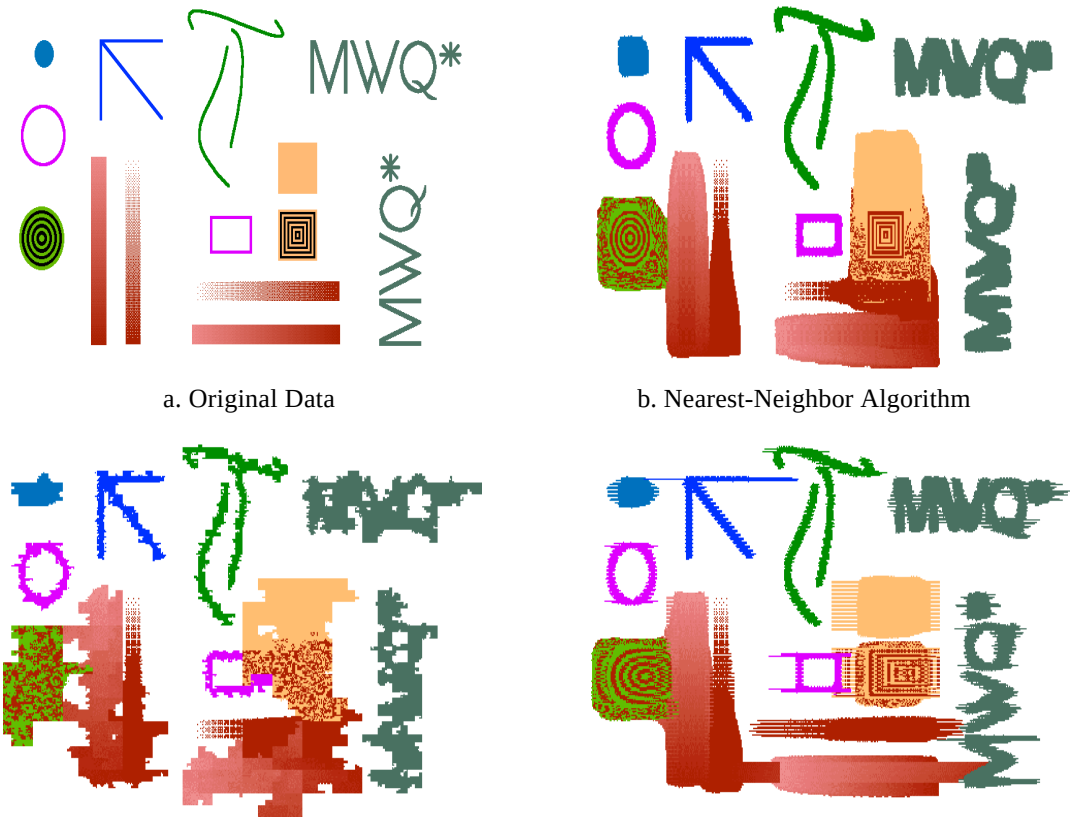


b. Curve-based Algorithm



c. *Gridfit* Algorithm

Figure13: Visualizations of Greyscale World Map Data Using Two Different Resolutions



a. Original Data

b. Nearest-Neighbor Algorithm

c. Curve-Based Algorithm

d. *Gridfit* Algorithm

Figure14: Visualizations of Synthetic Test Data Using the Three Algorithms

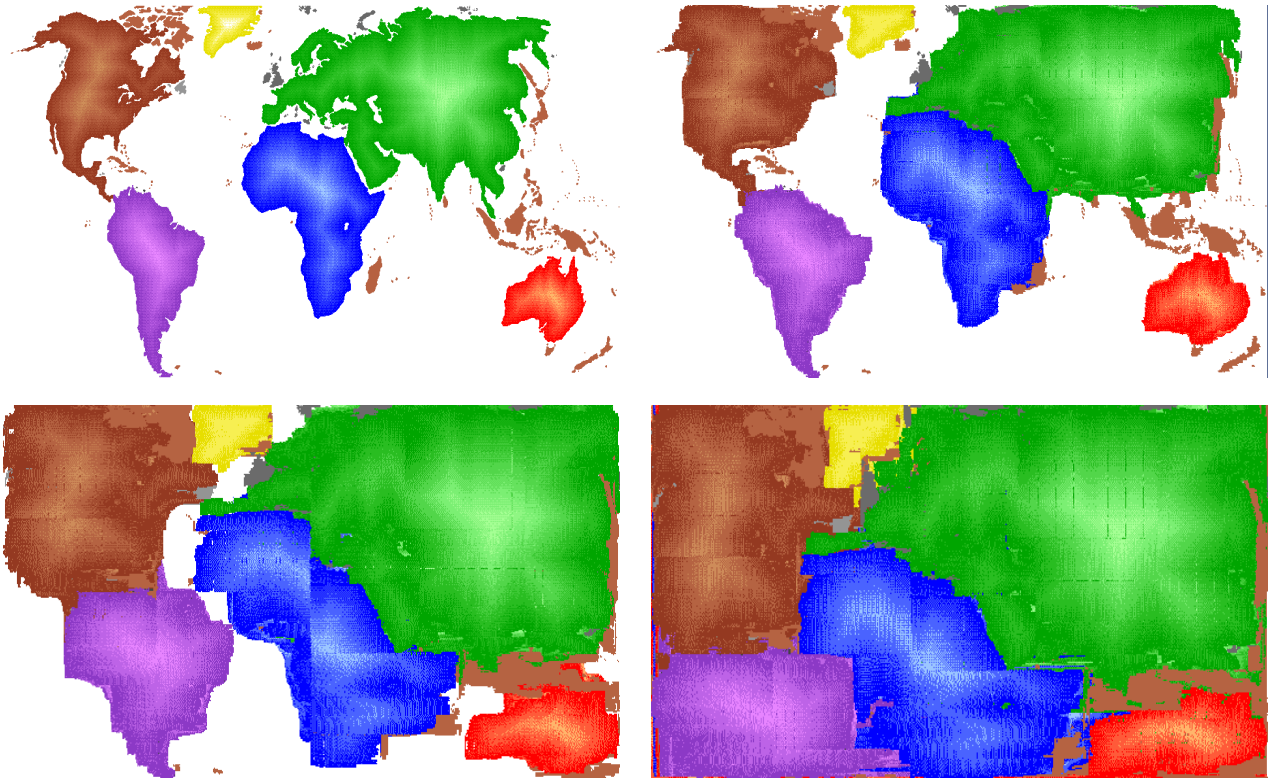


Figure 15: Visualization of World Continent Data at Different Resolutions