Visualizing Geo-Related Data Using Cartograms

Dissertation zur Erlangung des akademischen Grades
eines Dr. rer. nat. an der Universität Konstanz

vorgelegt von

Christian Panse

Dezember 2004

Parts of this thesis were published in [80, 75, 55, 88].

Submitted: December 14, 2004
Defended: February 22, 2005
Published: September 29, 2005

1. Referent: Prof. Dr. Daniel A. Keim, Universität Konstanz, Germany
2. Referent: Prof. Dr. Ulrik Brandes, Universität Konstanz, Germany
3. Referent: Dr. Stephen C. North, AT&T Shannon Laboratory, New Jersey, USA

Acknowledgements

The work described in this thesis was done with the help of many collaborators and friends.

First of all, I would like to thank my supervisor Prof. Dr. Daniel A. Keim, University of Konstanz, for introducing me into the highly interesting world of research and into the exciting field of cartograms. His continuous support and interest during my Ph.D. time in Konstanz and New Jersey and our fruitful discussions led to the success of this work. He always supported me to present my project on international conferences and doing so, I had the opportunity to meet colleagues and to exchange ideas with them.

Dr. Stephen C. North, AT&T Shannon Lab., joined in supervising my work and I'm very grateful to him for his great ideas and his interest in my research. Whenever problems occurred he helped and supported me, no question remained unanswered.

My colleague and valuable friend Dr. Roland Heilmann contributed considerably to the success of this thesis. His critical comments and discussions as well as our InfoVis 2004 publication were very important for this thesis.

I would like to thank my friends and colleagues Svetlana Vinnik, Carmen Sanz Merino, Jörn Schneidewind, Markus Wawryniuk, Alexander Hinneburg, Tobias Schreck, André Seifert, Benjamin Bustos, Hartmut Ziegler, Christian Goldberg, and Florian Mansmann for our discussions. My Ph.D. time in Konstanz is full of pleasant memories thanks to them. Especially, I would like to thank Mike Sips who never left me alone when we had a deadline or a technical problem. He was always ready for experiments and discussions. Thanks!

To my colleague Dr. Simon Byers, I'm grateful for getting me in touch with all the script-languages.

Dr. Eleftherios Koutsofios was very helpful especially in the beginning in organizing data and he kindly introduced me into powerwall-systems.

Furthermore, I'm grateful to Prof. Dr. Ulrik Brandes for his readiness to help and his continuous encouragement and support.

I would like to thank Prof. Dr. Thomas Seidl and Prof. Dr. Thomas M. Stricker for their help and advice in 2001 and 2002. Also, I would like to thank the anonymous reviewers from IEEE CGA, IEEE TVCG, and Palgrave' Information Visualization as well as from the IEEE InfoVis. Their friendly comments were very helpful for me to improve my work. Furthermore, I would like to thank Prof. Dr. Waldo Tobler, who read many of our publications in an early stage, for his very helpful comments concerning our research.

I'm very grateful to Oliver Maruhn for his kind technical help when I was at the other end of this earth and in answering all my technical questions.

Special thanks to Daniel's wife Ilse and their children who gave me a home and a dear family during my stay in New Jersey. I would like to thank my parents for encouraging me doing a Ph.D. and for their support during these years.

The most particular thanks go to my wife Anita whose patience, encouragement, and help during my Ph.D. time made it possible to finish this work.

Abstract

Cartograms are a well-known technique for showing geography-related statistical information, such as demographic and epidemiological data. The idea is to distort a map by resizing its regions according to a statistical parameter, but in a way that keeps the map recognizable.

In this thesis, we formally define a family of cartogram drawing problems. We show that even simple variants are unsolvable in the general case. Furthermore, we propose two methods of cartogram generation. Both algorithms cover a wide range of user requirements which can be directly maintained by the user. The first algorithm strictly retains the topology and shape of the map regions while minimizing the area error. The second algorithm approximates the map regions by rectangles focusing on an exact area approach and minimizing shape, topology and relative position of the map regions. Both algorithms are fast enough to be used in interactive systems which is important to be used as information visualization technique.

Application experiments show that the proposed algorithms can compute high-quality cartograms in few seconds, even for large maps with a high number of polygons. Also, our algorithms were designed for a dynamic visualization, for which we need an algorithm that recalculates a cartogram in a few seconds. None of the algorithms published before provides an adequate performance with an acceptable level of quality for this application. Additional application examples of the two new algorithms introduced in this work help to demonstrate their potential.

Zusammenfassung

Die Wissenschaft und Kunst, Karten zu erstellen ist so alt wie die Menschheit. Erste einfache Landkarten datieren bis in die Steinzeit zurück. Die Kartographie ist von immenser Bedeutung für die Entwicklung der Menschheit gewesen, Karten sind ein grundlegendes Werkzeug für die Entdeckung der Erde und des Weltalls.

Neben diesen traditionellen Karten, die bedeutsam für die Navigation sind, existieren sogenannte Kartogramme. Dies sind abstrakte Karten, die neben der geographischen noch zusätzliche Informationen visualisieren, z.B. statistische Werte. Dies ist von großer Bedeutung: Bei traditionellen Karten entsprechen die Flächen der Regionen der geographischen Fläche. Diese stehen jedoch in keinerlei Zusammenhang mit regionen-verknüpften statistischen Werten, z.B. bei der Darstellung von Bevölkerungszahlen. Sehr große Gebiete können sehr kleine statistische Werte haben und umgekehrt, z.B. niedrige oder hohe Bevölkerungsdaten. Bei einem Kartogramm werden die Flächen entsprechend den statistischen Werten verzerrt, d.h. für das Beispiel eines Bevölkerungskartogramms, daß dünn besiedelte Gebiete sehr klein werden und dicht besiedelte Gebiete sehr groß. Dadurch werden Fehlinterpretationen vermieden, das Verständnis erleichtert.

Um Kartogramme effektiv, d.h. leicht verständlich zu gestalten, ist es essentiell, daß der Mensch die dargestellten Daten leicht verstehen kann und mit den ursprünglichen geographischen Daten in Verbindung bringen kann. Dieses Verständnis ist wiederum abhängig davon, daß die ursprüngliche Form, die Lage der einzelnen Gebiete zueinander sowie der Zusammenhang der Gebiete möglichst gut erhalten wird.

Im allgemeinen Fall ist dieses Problem nicht lösbar, was zu Beginn dieser Arbeit gezeigt wird. Aufgrund der Überlegung, daß das Problem vermutlich nicht in Polynomialzeit lösbar ist, wird es in dieser Arbeit als Optimierungsproblem behandelt.

Der Hauptbeitrag der vorliegenden Dissertation besteht darin, daß zwei neuartige Algorithmen zur Berechnung von Kartogrammen entwickelt wurden. Der erste Algorithmus wurde *CartoDraw* genannt. Der Vorteil dieses Algorithmus liegt darin, daß die Topologie und Form der Ausgangskarte erhalten wird. Dabei wird versucht, den Flächenfehler, d.h. die Abweichung der Flächen des Kartogramms von den statistischen Werten entsprechenden Flächen, zu minimieren. Naturgemäß kann der Flächenfehler nicht restlos beseitigt werden. Diesen Nachteil umgeht der zweite in dieser Arbeit vorgestellte Algorithmus mit der Bezeichnung *RecMap*. Dabei wird jede Kartenregion durch ein Rechteck ersetzt, dadurch wird ein Flächenfehler vollständig vermieden, bei Verlust der ursprünglichen Form. Der Topologiefehler wird minimiert.

Beide Algorithmen wurden implementiert. Die Flächenfehler sind im Vergleich mit bereits vorhandenen Algorithmen ähnlich oder kleiner, die benötigte Rechenzeit ist im Vergleich um Größenordnungen kleiner. In einer visuellen Gegenüberstellung zu existierenden Methoden generieren die vorgestellten Verfahren vergleichbare oder bessere Kartogramme.

Welcher der beiden Algorithmen zu bevorzugen ist, hängt von der Zielsetzung ab. Die Anwendung von *CartoDraw* und *RecMap* wird anhand von zahlreichen Beispielen im Kapitel *Applications* gezeigt.

Die vorliegende Arbeit bietet eine Vielzahl von neuen Möglichkeiten zur Visualisierung geographiebezogener Daten mit Hilfe von Kartogrammen.

Contents

List of Tables

List of Algorithms

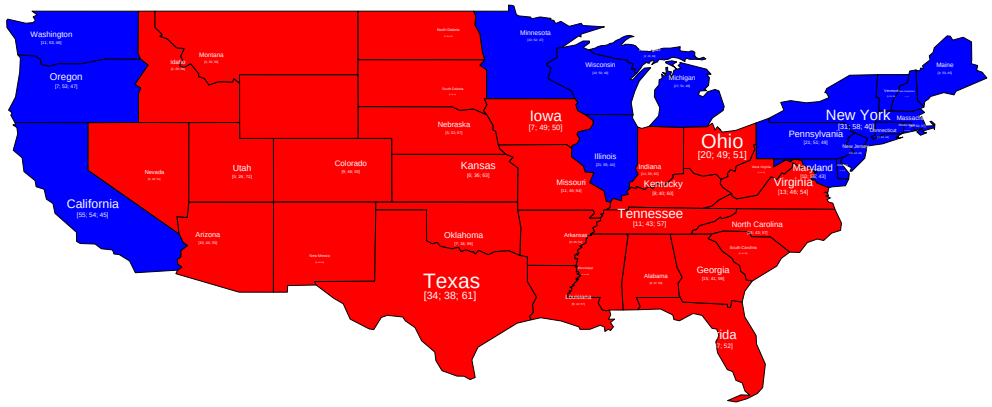
List of Figures

1 Introduction

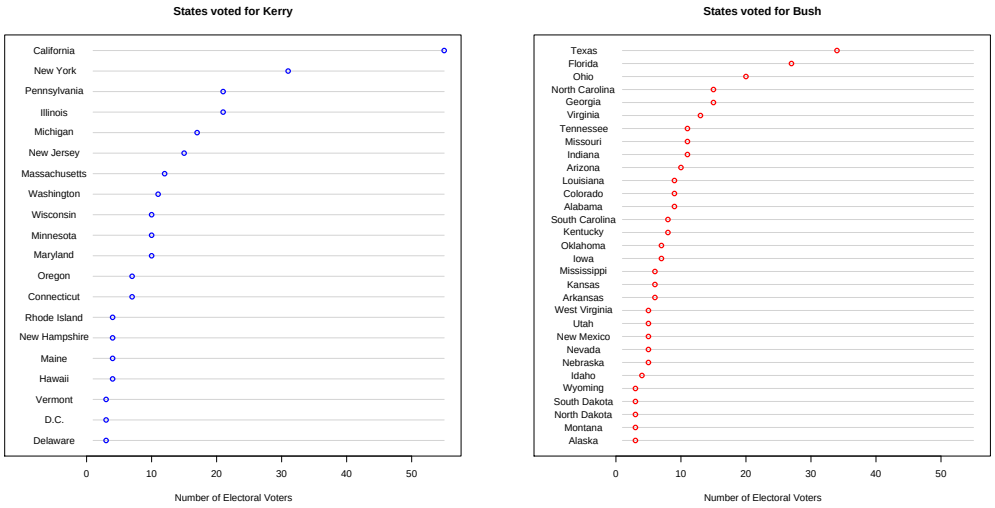
Maps are as old as humans. First maps were probably manufactured using animal bones and used by humans in the Stone Age [61, page 63]. The so far oldest known map of the cosmos, was found on a presumably 3600 years old bronze plate famous today as the *Sky Disc of Nebra* [98]. This *Sky Disc* was used to predict sowing and harvest for several hundred years. During the evolution of mankind the art of drawing maps became more and more sophisticated. Maps were and still are probably among the most important tools for the discovery of the earth and the universe. Maps are abstract descriptions of things which are directly linked to a spatial position, e.g., sky maps, land-covering maps, or constructional maps. Beside traditional maps, which are used for navigation, there exist a large amount of abstract maps which visualize additional, often statistical, information on the map. The Greek and Romans called them cartograms.

There exist several kinds of cartograms. Some of them are mentioned below. *Timetable cartograms* are known from the London or New York subway [115]. The subway lines are straightened and stations are drawn as points. *Traffic-flow maps* are simplified street maps where the line width corresponds to the number of vehicles passing the streets. *Migration maps* show migration of people during a time period by arrows on the map. *Isochrones* are used to illustrate the travel time from one to several locations on a map. Destinations with the same travel time are connected by lines. *Route maps* are used to describe the path from a starting to a target location an a map. On *Choropleth maps* the regions are colored lighter or darker to express the statistical value. If the areas of the maps regions are proportional to the statistical value, the maps are called *value-by-area cartograms*. However, these maps were simplified because of two reasons. First, simplified maps are easier to draw and second, they are easier to understand and to use by humans. On such visualizations only those things are drawn which are important for the understanding. This dissertation deals with value-by-area cartograms. When mentioning cartograms in this thesis we will always consider value-by-area cartograms. For a cartogram to be effective, a human must be able to understand quickly the displayed data and relate it to the original geographical model. Recognition in cartograms depends on preserving basic properties, such as shape, orientation, and contiguity. This, however, is difficult to achieve in the general case because it is impossible to retain the original map's topology. Because the generation of contiguous cartograms by simultaneous optimization of these objectives is difficult, all currently available algorithms are very time-consuming. Let's consider a potential application example. Supposed we have a map as the U.S. continental map and the results of the 2004 U.S. presidential election should be visualized. The political map on figure 1.1 is an often-used visualization of the election results. In this conventional choropleth map, each state is colored according to the winner of the vote, A drawback of that visualization is that the area is not proportional to the number of electors and it appears that the red party got a big majority. For completeness, this type of visualization needs additional information (e.g. as scatterplot or table) linked to the map regions. Instead, by using cartograms the map can be distorted in a way that the area of each state represents the number of electors and therefore it can be seen that the U.S. election in 2004 was a head-to-head race between both candidates. Figure 1.2 shows such a cartogram. The application of cartograms is not restricted to visualization of election data. Example applications include population demographics [120] and epidemiology [54].

The goal of this work is to display continuously the behavior of an input parameter, in particular, its deviation from an expected value. Our aim is to create dynamic cartograms for on-line network monitoring, such as display of traffic or transaction event levels by country, state, and local regions. This requires a very fast cartogram generation, and to our knowledge there is currently no competing algorithm with adequate speed for that.



(a)



(b)

Figure 1.1: Election 2004 analysis – Political map 1.1(a) and categorized and sorted election results 1.1(b). Numbers show [numbers of electors, percentage of votes for Kerry, percentage of votes for Bush].

Cartographers and geographers have used cartograms long before computers were available to make displays [106, 107, 58]. References date back as far as 1868 (see remarks on Levasseur in [47] on page 355). A short historical overview can be found in [27]. The basic idea of a cartogram is to distort a map by resizing its regions by some geographically-related parameter. Because cartograms are difficult to make manually, the study of algorithms to draw them is of high interest.

The main contribution of this work is the design of two completely new algorithms for computing contiguous cartograms based on an observation of existing methods. The first method, called *CartoDraw*, retains the topology by minimizing the area error while on the second approach, named *RecMap*, each map region is approximated by a rectangle to avoid the area error. Both algorithms were implemented and various application examples show their functionality. Beside that the area error and the computational time is less or similar to the other methods, a visual comparison with existing cartogram construction techniques shows that our algorithms compute comparable if not better cartograms. Additionally, we modified and combined our algorithms with some other techniques, which creates new opportunities for visualizing geo-related data.

The dissertation is organized as follows: In chapter 2, we expose the classification in the information visualization field. We study some geographic phenomena and we shortly introduce a pixel based visu-

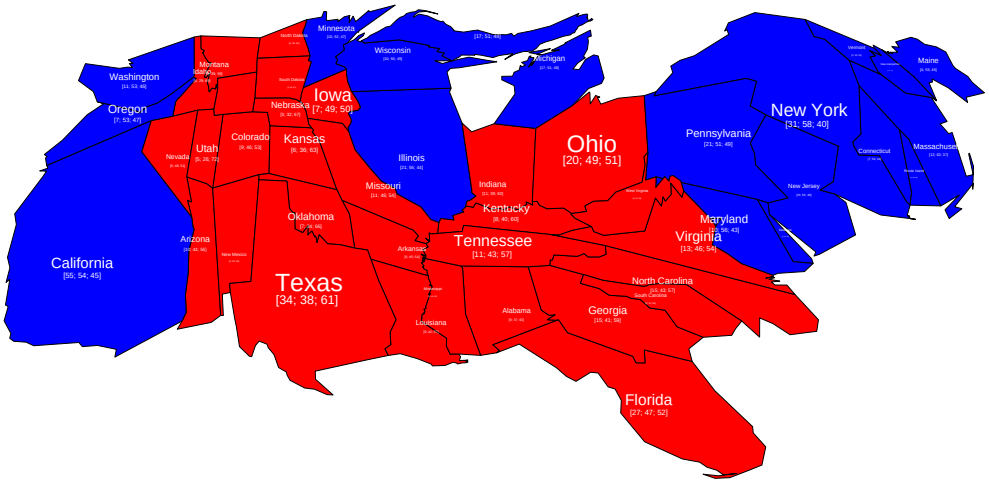


Figure 1.2: Election 2004 analysis cartogram – The area corresponds to the number of electors. The shape of the states is clearly recognizable. Numbers show [numbers of electors, percentage of votes for Kerry, percentage of votes for Bush].

alization for geo-related data called *PixelMap*. The chapter finishes with a description of high resolution display wall design at the University of Konstanz.

Chapter 3 describes what cartogram drawing is all about, explores the problems to be faced, and reviews previous work on cartogram drawing. Then we define several variants of the problem and show that even simple ones are unsolvable in the general case. Because it is not known, if the cartogram problem can be solved in polynomial time, we expect that feasible variants are likely to be $\mathcal{NP}$-hard problems. Therefore, heuristics are necessary to solve the problem. Followed by observations on previous cartogram drawing, we introduce a map simplification technique which is used as pre-processing step of our cartogram algorithms.

Chapters 4, 5, and 6 include the main contribution of this thesis. Based on some important observations in chapter 3, in chapter 4 we develop the *CartoDraw* heuristic which uses scanline-based local repositioning of vertices with an explicit shape error control function to preserve both the global shape and the shape of interior polygons while providing sufficient speed for dynamic cartograms drawing.

In the following chapter a genetic based algorithm is introduced which approximates each map region by a rectangle, called *RecMap*.

In chapter 6 we give a description of a various extensions and combinations of cartogram techniques. Furthermore, we describe the design of the *CartoDraw*-System which is used as a graphical user interface (GUI).

In chapter 7, we present a number of application examples and provide a detailed comparison with previous approaches, showing the effectiveness and efficiency of our proposed algorithms.

Chapter 8 summarizes our approaches and discusses open issues.

All chapters which introduce new methods for computing cartograms, i.e. 4, 5, and 6 will have there own evaluation part. The appendix chapters provide related work to this thesis.

2 Information Visualization: Scope, Techniques and Opportunities for Geovisualization

2.1 Introduction

Geovisualization deals with many disciplines including cartography, scientific visualization, image analysis, information visualization and exploratory data analysis [33, see chapter 1] and [94]. Cartography is the art and science of drawing maps [107, page 293] and this thesis covers most of it. A map is a visualization of points, lines, or areas. The maps are used in many ways e.g., they can be static or dynamic. Dynamic maps are often used in exploratory data analysis. Using maps can be useful for the data exploration because e.g., maps can visualize information at location, they can show distribution of spatial pattern, or it is possible to compare pattern in two or more maps. The information to be visualized are often massive data generated from sensors, e.g., radio telescopes [100] or genome data base [117], transaction processes, e.g., ecommerce data, or even feature vectors of real world objects, and they are often stored as tables in log files or data bases. Beside time this data is often referenced by a geographic location. Both, time and space can not easily be exchanged [9] which make them often difficult to visualize.

This chapter touches several areas of geovisualization. It starts with an overview of information visualization. It explains the phenomena of geo-related visualization, shows one example technique for visualizing point data, and ends with a demonstration of a device for data exploration. Parts of this chapter were published in [88, 87].

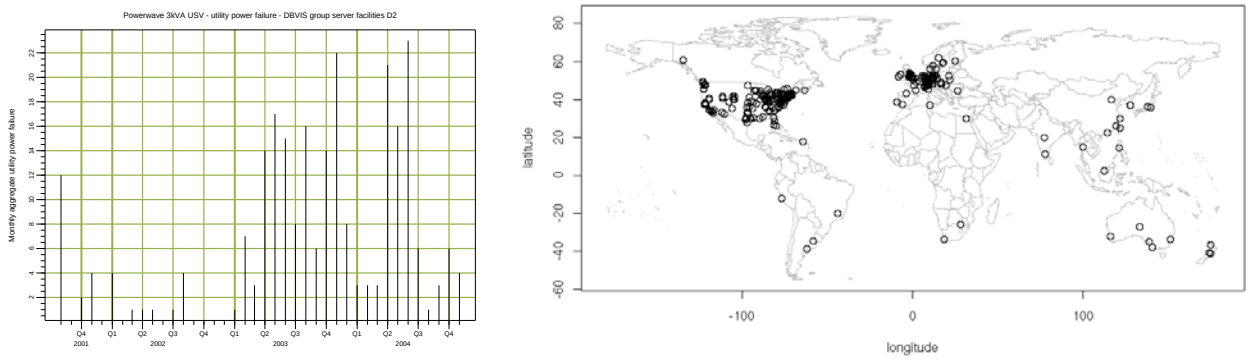
2.2 Visual Exploration Paradigm

Visual data exploration usually follows the *Information Seeking Mantra* [110] which is a three-step process: ***Overview first, zoom and filter, and then details-on-demand***.

First, the user needs to get an overview of the data. In the overview, the user identifies interesting patterns or groups in the data and focuses on one or more of them. For analyzing these patterns, the user needs to drill-down and access details of the data. Visualization technology may be used for all three steps of the data exploration process. Visualization techniques are useful for showing an overview of the data, allowing the user to identify interesting subsets. In this step, it is important to keep the overview visualization while focusing on the subset using another visualization. An alternative is to distort the overview visualization in order to focus on the interesting subsets. This can be performed by dedicating a larger percentage of the display to the interesting subsets while decreasing screen space for uninteresting data. The visualization technology does not only provide visualization techniques for all three steps but also bridges the gaps between them.

2.3 Classification

There are a number of well-known techniques for visualizing large data sets, such as x-y plots, line plots, and histograms. These techniques are useful for data exploration but are limited to relatively small and low dimensional data sets. Over the last years, a large number of novel information visualization techniques (see [16, 130, 114]) have been developed, allowing visualizations of multidimensional data sets without inherent two- or three-dimensional semantics. Keim [70] classifies the techniques according to three criteria: the data to be visualized, the visualization technique, and the interaction technique used.



(a) 1D data (b) 2D data

Figure 2.1: Data type to be visualized.

The *data type to be visualized* [110] may be:

- ***One-dimensional data*** - such as temporal (time-series) data
  One-dimensional data usually has one dense dimension. A typical example of one-dimensional data is temporal data. Note that with each point of time, one or multiple data values may be associated. An example are time series of stock prices. Figure 2.1(a) shows the accumulated monthly number of utility power failure of an USV system during four years at the University of Konstanz.
- ***Two-dimensional data*** - such as geographical maps
  Two-dimensional data usually has two dense dimensions. A typical example is geographical data, where the two distinct dimensions are longitude and latitude. Longitude and Latitude describe locations on a 3D surface and some transformation is required to project the relationships between locations specified in this way on a plane. Besides, depending upon the cartography used, various characteristics of the relationships between locations are either preserved or lost. After the projection, the geographical data can be stored as two-dimensional data with X-Y-dimensions. X-Y-plots are a typical method for showing two-dimensional data and maps are a special type of X-Y-plot for showing geographical data. Figure 2.1(b) displays a world map with 600 “world wide web accesses” to the *CartoDraw* web site [76] during 6 months.
- ***Multi-dimensional data*** - such as relational tables
  Many data sets consist of more than three attributes and therefore do not allow a simple visualization as 2-dimensional or 3-dimensional plots. Examples of multidimensional (or multivariate) data are tables from relational databases, which often have tens to hundreds of columns (or attributes). Since there is no simple mapping of the attributes to the two dimensions of the screen, more sophisticated visualization techniques are needed, such as parallel coordinates[59], or e.g. the scatterplot matrix in figure 2.2. The figure shows 3085 items of the continental U.S. election 2004 data where the attributes are `(longitude, latitude, area, vote% (for Kerry), #electors)`.
- ***Text & hypertext*** - such as news articles and web documents
  Not all data types can be described in terms of dimensionality. In the age of the World Wide Web,

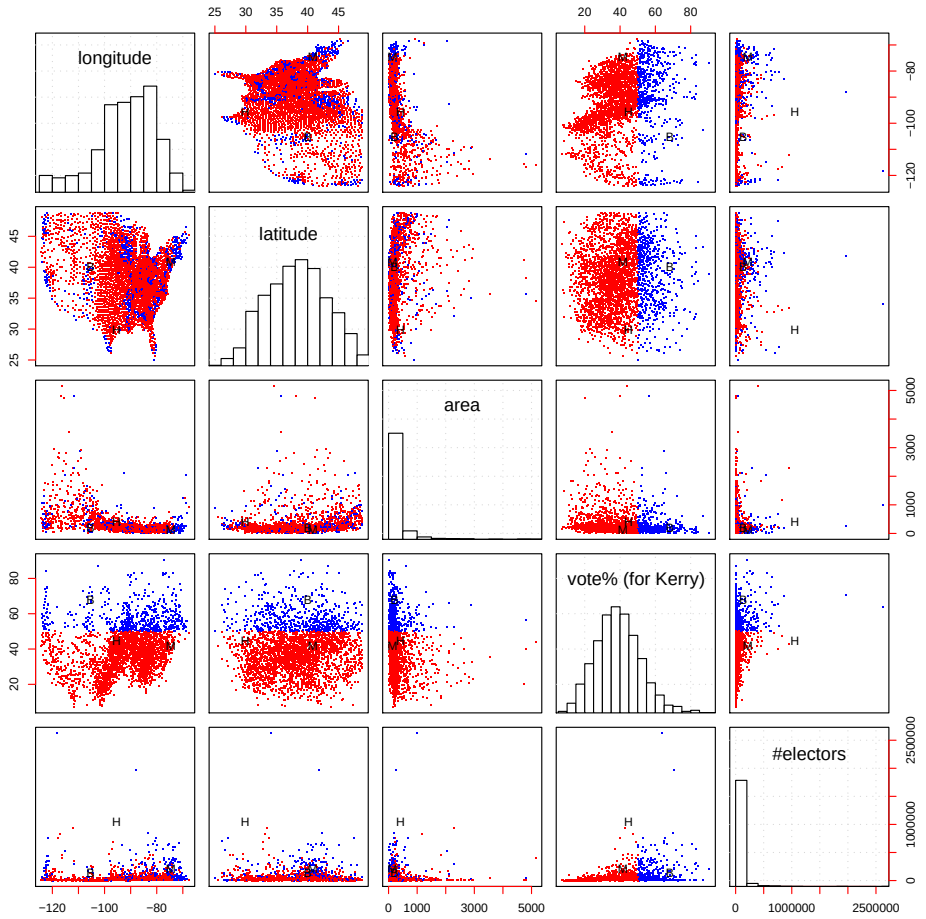


Figure 2.2: Data type to be visualized: multi-dimensional data – color brushing: red - Bush; blue - Kerry – label brushing: B - Boulder, Colorado; H - Harris, Texas; M - Morris, New Jersey

one important data type is text and hypertext, as well as multimedia web page contents. These data types differ in that they cannot be easily described by numbers, and therefore most of the standard visualization techniques cannot be applied. In most cases, a transformation of the data into description vectors is necessary before visualization techniques can be used. An example for a simple transformation is word counting which is often combined with a principal component analysis (PCA) [40, 91] or multidimensional scaling (MDS) [123, 12] to reduce the dimensionality to two or three. Figure 2.3 is a result of a MDS of all keywords in this thesis which have a frequency of more than twenty. A distance matrix was computed using the minimal distance between the text positions of each pair of words. The size of the words is scaled according to their inferred relevance and the distances between each pair of the plot reflect the connectivity of the research keywords. The terrain color map visualizes a 2D kernel density estimation[128] of the word location.

- ***Hierarchies & graphs*** - such as network data
  Data records often have some relationship to other pieces of information. These relationships may be ordered, hierarchical, or arbitrary networks of relations. Graphs are widely used to represent such interdependencies [6]. A graph consists of a set of objects, called nodes, and connections between these objects, called edges or links. Examples are the e-mail interrelationships among people, their shopping behavior, the file structure of the hard disk or the hyper links in the world wide web. The graph in figure 2.4 reflects 1,224,733 IP addresses and 2,093,194 IP links, (immediately adjacent addresses in a traceroute-like path) of skitter data from 21 monitors probing approximately 932,000 destinations spread across over 75,000 (70%) of globally routable network prefixes [14].

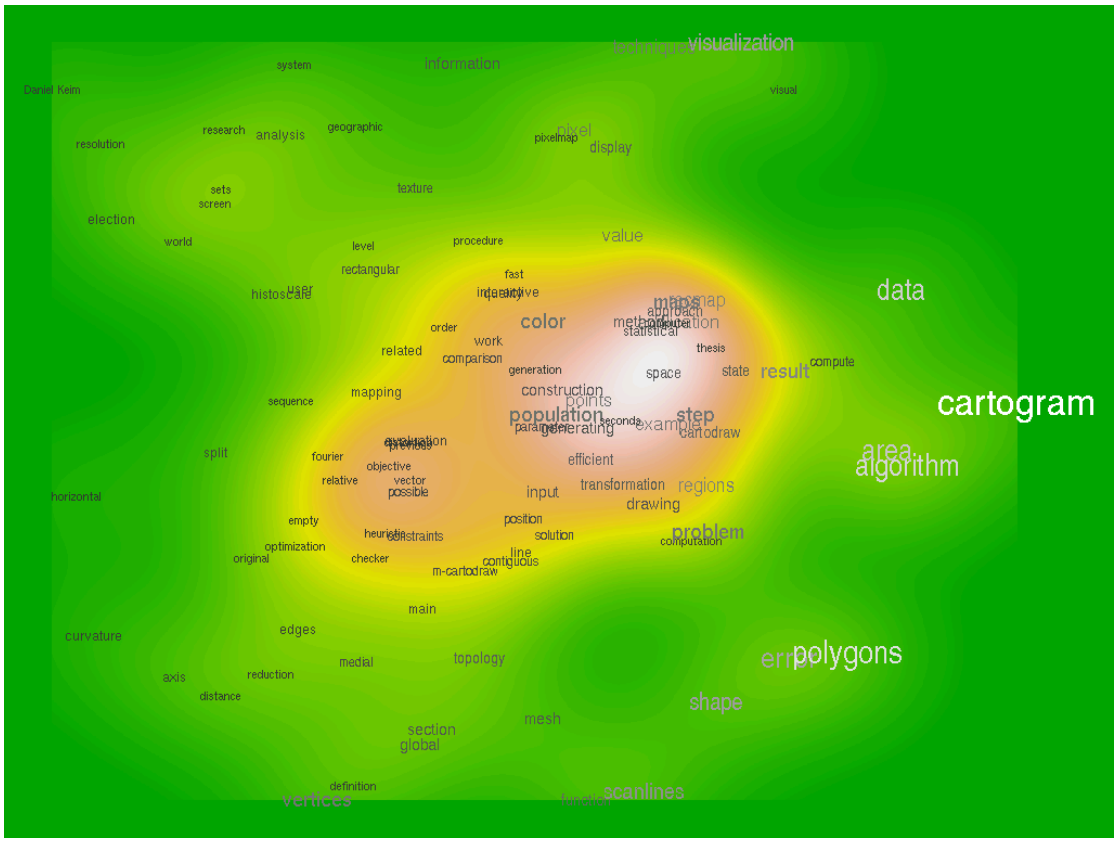


Figure 2.3: Data type to be visualized: text & hypertext

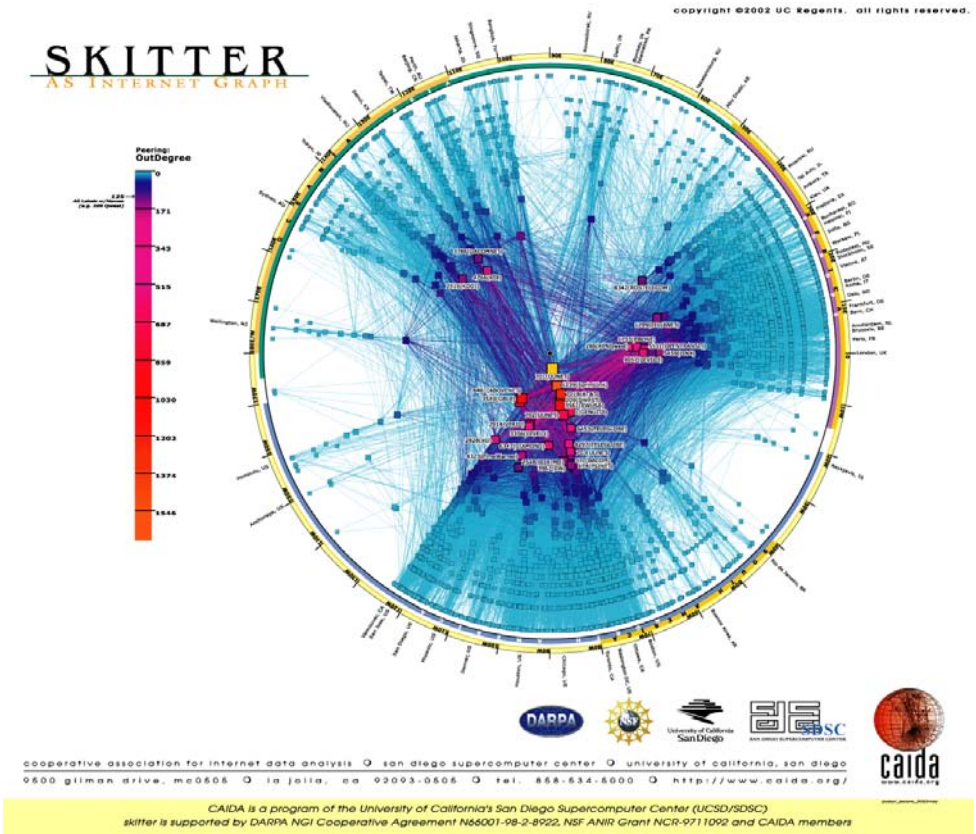


Figure 2.4: Data type to be visualized: Hierarchies & graphs (*Courtesy of ©CAIDA, University of California, 2000*).

There are a large number of visualization techniques that can be used for visualizing data. In addition to standard 2D/3D-techniques such as X-Y (X-Y-Z) plots, bar charts, line graphs, and simple maps, there are a number of more sophisticated classes of visualization techniques. The classes correspond to basic visualization principles that may be combined in order to implement a specific visualization system. The *visualization technique* are:

- ***Geometrically-Transformed Displays*** - aim at finding "interesting" transformations of multidimensional data sets. This class of geometric display methods includes techniques from exploratory statistics such as scatterplot matrices [23] and techniques that can be subsumed under the term "projection pursuit" [56].

- ***Iconic Displays*** - The idea is to map the attribute values of a multi-dimensional data item to the features of an icon. The most famous techniques are Chernoff faces [19]

- ***Dense Pixel Displays*** - The basic idea of dense pixel techniques is to map each dimension value to a colored pixel and group the pixels belonging to each dimension into adjacent areas.

- ***Ordering of Dimensions*** - The problem is often how to order the dimensions of multidimensional data in the visualization technique [69].

- ***Stacked Displays*** - Stacked display techniques are tailored to present data partitioned in a hierarchical fashion. In the case of multi-dimensional data, the data dimensions to be used for partitioning the data and building the hierarchy have to be selected appropriately. An example of a stacked display technique is *Dimensional Stacking* [92].

In addition to the visualization technique, for an effective data exploration it is necessary to use one or more interaction techniques. *Interaction techniques* allow the data analyst to directly interact with the visualizations and dynamically change the visualizations according to the exploration objectives. In addition, they also make it possible to relate and combine multiple independent visualizations. The *interaction techniques* used are:

- ***Dynamic Projection*** - Dynamic projection is an automated navigation operation. The basic idea is to dynamically change the projections in order to explore a multi-dimensional data set. A classic example is the GrandTour system [4].

- ***Interactive Filtering*** - Interactive filtering is a combination of selection and view enhancement. In exploring large data sets, it is important to partition the data set interactively into segments and focus on interesting subsets.

- ***Zooming*** - Zooming is a well-known view modification technique that is widely used in a number of applications. In dealing with large amounts of data, it is important to present the data in a highly compressed form to provide an overview of the data, but at the same time, allowing a variable display of the data at different resolutions. Zooming does not only mean displaying the data objects larger, but also that the data representation may automatically change to present more details on higher zoom levels.

- ***Distortion*** - Distortion is a view modification technique that supports the data exploration process by preserving an overview of the data during drill-down operations. The basic idea is to show portions of the data with a high level of detail while others are shown with a lower level of detail. Popular distortion techniques are hyperbolic and spherical distortions [93]. For an example for a combination of zooming, distortion, and filtering techniques see [65].

- ***Brushing and Linking*** - *Brushing* is an interactive selection process that is often, but not always, combined with *linking*, a process to communicate the selected data to other views of the data set. There are many possibilities to visualize multi-dimensional data, each with their own strengths and weaknesses. The idea of linking and brushing is to combine different visualization methods to overcome the shortcomings of individual techniques. Scatterplots of different projections, for example, may be combined by coloring and linking subsets of points in all projections. In a similar fashion, linking and brushing can be applied to visualizations generated by all visualization techniques described above. As a result, the brushed points are highlighted in all visualizations, making it possible to detect dependencies and correlations. Interactive changes made in one visualization are also automatically reflected in the other visualizations. Note that connecting multiple visualizations through interactive linking and brushing provides more information than considering the component visualizations independently. Typical examples of visualization techniques that have been combined by linking and brushing are multiple scatterplots (see figure 2.2), bar charts, parallel coordinates, pixel displays, and maps.

A more detailed discussion of the classification is given in [87, 88].

2.4 Phenomena of Geo-Related Visualization

Geo-related data is different from other kinds of data in the way that geo-related data describes objects or phenomena with a specific location in the real world. Large spatial data sets can be seen as a result of accumulating samples or readings of phenomena in the real world while moving along two dimensions in space. In general, spatial data sets are discrete samples of a continuous phenomenon. Nowadays, there exist a large number of applications, in which it is important to analyze relationships that involve geographic locations. Examples include global climate modeling (measurements such as temperature, rainfall, and wind-speed), environmental records, customer analysis, telephone calls, credit card payments, and crime data. Because of this special characteristic, the visualization strategy for spatial data is straightforward. We map the spatial attributes directly to the two physical screen dimensions. The resulting visualization depends on the spatial dimension or extent of the described phenomena and objects. Spatial phenomena may be distinguished according to their spatial dimension or extent:

- ***point phenomena*** - have no spatial extent, can be termed zero-dimensional and can be specified by longitude and latitude coordinate pairs with a statistical value z. Examples are census demographics, oil wells, and crime data. (see figure 2.1(b))

- ***line phenomena*** - have length, but essentially no width, can be termed one-dimensional and can be specified by unclosed series of longitude and latitude coordinate pairs for each phenomenon. Examples are large telecommunication networks, internet, and boundaries between countries. (see figure 2.4)

- ***area phenomena*** - have both length and width, can be termed two-dimensional and can be specified by series of longitude and latitude coordinate pairs that completely enclose a region and a statistical value z for each phenomenon. Examples are lakes, and political units such as states or counties. (see figure 1.2 and the figures in chapter and 7)

For each of the phenomena, several visualization approaches have been developed over the last years. More details about spatial visualization and cartography can be found in [107, 95, 27, 112, 88, 33].

2.5 *PixelMap* – A Pixel Based Visualization Technique for Large Geo-Related Data

High resolution displays are of value for exploring extremely large data set. However, there exists data where more sophisticated visualization techniques are required.

Varying degree of pixel overlap depending on screen resolution - even with a screen resolution of 1600×1200, the degree of overlap is about 0.3; 30% of our sample of data points (about 12000 points) from the U.S. Year 2000 Census Household Income database cannot be directly placed without overwriting already-occupied pixels

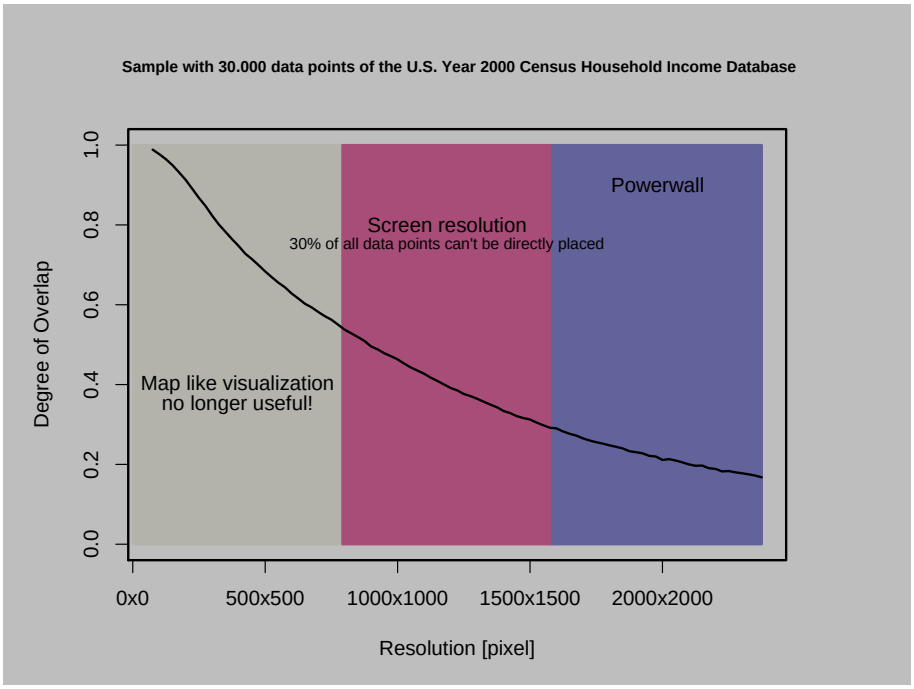


Figure 2.5: Pixel overlap on varying screen resolution using the U.S. Year 2000 Census Household Income database

Pixel visualizations are technique where each data item is represented by exactly one pixel on the computer screen. *PixelMap* is one of these techniques. *PixelMap* solves the problem of displaying dense point sets on maps, by combining clustering and visualization techniques [82].

First, the Fast-PixelMap algorithm [81, 85, 82, 111] approximates a two-dimensional *kernel density estimation* (KDE) in the two geographical dimensions performing a recursive partitioning of the dataset and the 2D screen space by using split operations according to the geographical parameters of the data points and the extensions of the 2D screen space. The goal is

1. to find areas with density in the two geographical dimensions and
2. to allocate enough pixels on the screen to place all data points of dense regions at unique positions close to each other.

The top-down partitioning of the dataset and 2D screen space results in distortion of certain map regions. That means, however, virtually empty areas will be shrinking and dense areas will be expanding to achieve pixel coherence.

For an efficient partitioning of the dataset and the 2D screen space and an efficient scaling to new boundaries, a new data structure called Fast-PixelMap is used. The Fast-PixelMap data structure is a combination of a gridfile and a quadtree [42] which realizes the split operations in the data and the 2D

screen space. The Fast-PixelMap data structure enables an efficient determination of the old (boundaries of the gridfile partition in the dataset) and the new boundaries (boundaries of the quadtree partition in the 2D screen space) of each partition. The old and the new boundaries determine the local rescaling of certain map regions. More precisely, all data points within the old boundaries will be relocated to the new positions within the new boundaries. The rescaling reduces the size of virtually empty regions and unleashes unused pixels for dense regions.

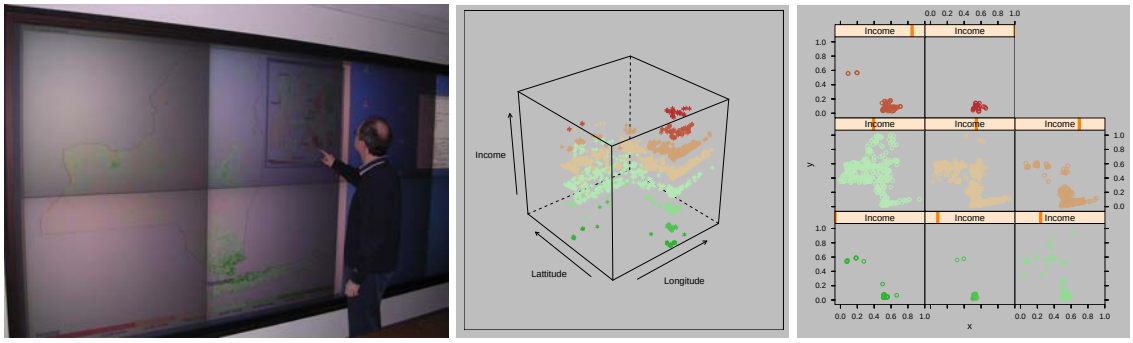
In a second step, the Fast-PixelMap algorithm approximates a three-dimensional *kernel density estimation*-based clustering in the three dimensions performing an array based clustering for each dataset partition. After rescaling of all data points to the new boundaries, the iterative positioning of data points (pixel placement step) is done, starting with the densest regions and within the dense regions the smallest cluster is chosen first. To determine the placement sequence, we sort all final gridfile partitions (leaves of the Fast-PixelMap data structure) according to the number of data points, they contain. The clustering is a crucial pre-processing step to make important information visible and to achieve pixel coherence[1] with respect to the selected statistical parameter.

The final step of the pixel placement is a sophisticated algorithm which places all data points of a gridfile partition to pixels on the output map in order to provide visualizations which are as position-, distance-, and cluster-preserving as possible.

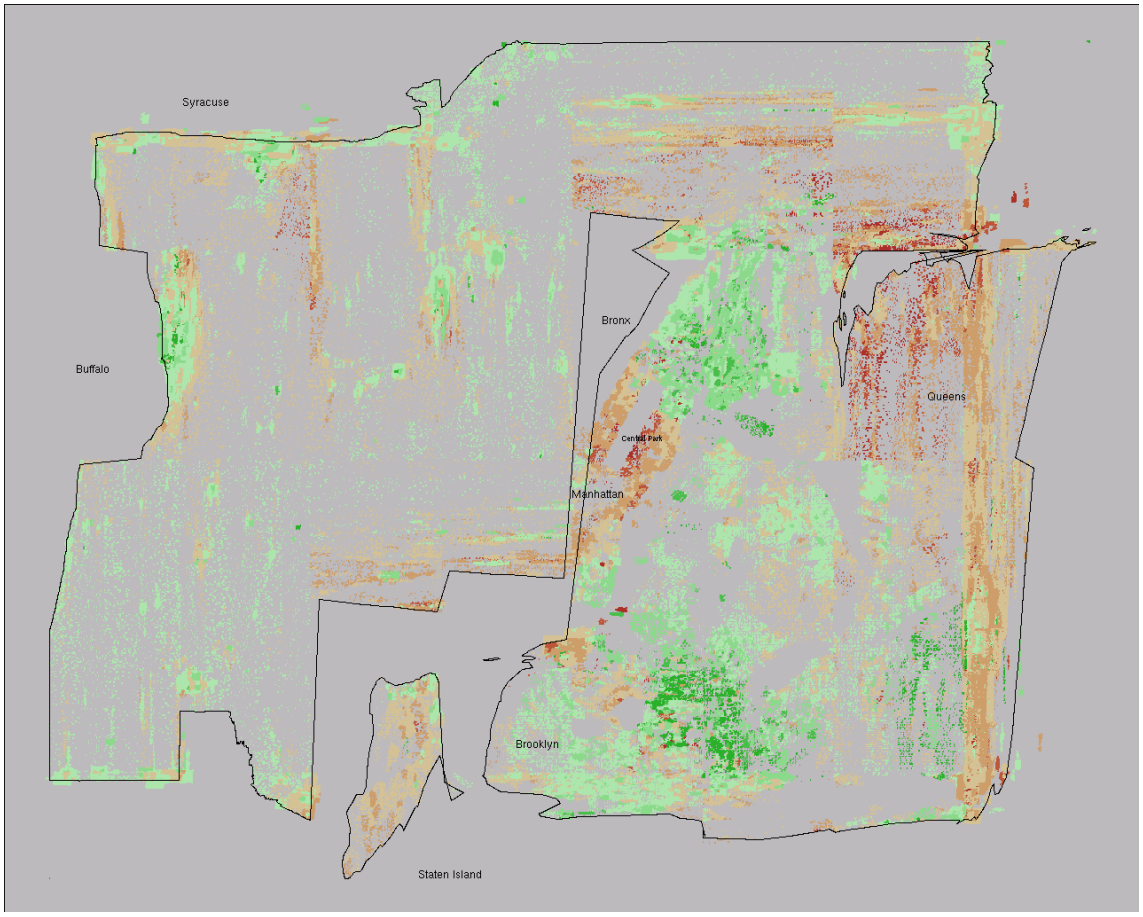
An example based on the U.S. Census income data set is displayed in Figure 2.6. Figure 2.6(a) shows a traditional map. Even on a high-resolution display overplotting obscures data points. The next approach is a 3-D–point clouds visualization of the longitude,latitude, and the statistical value. For that picture we used a random sampling of 1% of the whole data set.

PixelMap shows 100% of the data without overplotting. An example is illustrated in figure 2.6.

[1] Pixel coherence means similarity of adjacent pixels, which makes small pixel clusters perceivable.



(a) Conventional Map (b) 3-D–point cloud (c) X–Y–level plot



(d) *PixelMap*

Figure 2.6: Comparison of Traditional Map versus *PixelMap* - New York State - Year 1999 Median Household Income. This map displays cluster regions e.g. on the East side of Central Park in Manhattan, where inhabitants with high income live, or on the right side of Brooklyn, where inhabitants with low income live.

2.6 High Resolution Display Walls

There are several ways for visualizing all this massive data mentioned in the beginning of this chapter. We can reduce the data by sampling, aggregating the data, averaging them, or some other ways of data pre-processing. Alternatively, we can increase the resolution of the display device using *high resolution displays*[2].

High resolution displays are a new area in research [57, 132, 104]. Because the resolution of existing LCD is restricted or the size of the pixel is too small to make use of them, researchers began to build up so-called *powerwalls*. These *powerwalls* consist of more than one single display to arrange one large virtual screen. The virtual screen is realized by a backward projection using projectors. The projectors are ordered in a matrix layout on the other side of the projection wall to avoid shade redraws of the users. Existing powerwall layouts can be seen in figure 2.7.

(a) AT&T Global Network Operation Center (b) 4 x 2 projectors AT&T's Info Wall

Figure 2.7: High resolution walls (©*IEEE, Courtesy of Stephen C. North, AT&T Shannon labs*)

To achieve the highest visualization quality as possible the main features of the output device should be a *homogeneous screen* which means:

- there is an electronic and optic *edge blending* between the overlapping projections,
- all projectors take the intersection of all projectors color spaces, and
- the brightness of all linked projectors is monitored and adjusted to the lowest value.

To achieve a *homogeneous screen* it is necessary that the projectors are able to communicate their operating parameters to each other. Additionally, it is important that the projectors

- have a high contrast ratio ($\geq 1:1000$),
- geometry correction, and
- stereo visualization facilities.

Beside the projection properties, the system should serve for many different research projects and researchers. Therefore the system must feature to run existing software application without modification and no new compilation of the source code on a wide range of operation systems.

[2]Note that large size printers and plotters are also alternatives devices and they are not to underestimate. Because of their nature they do not allow an interactive exploration of the data and we need a lot of time and other resources to obtain a result.

Based on the experience with the AT&T Info Wall and on related research during the last four years, at the beginning of 2005 the computer science department at the University of Konstanz started to install a high resolution display wall system.

The "wall" consists of eight Barco single chip DLP-projectors [5] with a matrix layout of 4×2 projectors having a resolution of 1280×1024 pixels and an overall netto resolution of $4640 \times 1920 \approx 9 \cdot 10^6$ pixels. A $13600 \times 6000 mm^2$ large room is separated by a $5000 \times 2150 \times 6 mm^3$ ($w \times h \times d$) large projection wall into one $6000 \times 3600 mm^2$ large projector and technique room and a demonstration and presentation room. The pixel size results by

$$\frac{4640 \cdot 1920}{5000 \cdot 2150} \left[\frac{\text{pixel}}{mm^2} \right] \approx 0.8 \frac{\text{pixel}}{mm^2} \tag{2.1}$$

The system was primary designed for visualizing a wide variety of applications known from the infor-

Figure 2.8: High resolution 4×2 Barco DLP based projector ($\approx 9 \cdot 10^6$ pixels) i-wall at the University of Konstanz

mation visualization and data mining community which are

- *pixel based visualization* as *VisDB*, *PixelBarChart*, *PixelMap*
- large graphs, maps and networks
- visualization system as *Splus* [29], *Xmdv* [129], *WALDO* [83], *MineSet*, *HD-eyes*, ESRI ArcGIS-Labkit [37, 38]

Therefore the hardware should support interactions with the visualization software.

Secondary, the system should also serve as high-resolution 3D graphic device which devours magnitudes of more computing performance.

However these are two different application domains and therefore two different hardware configurations are needed. For the primary application domain as graphic system we experimented with an xentera GT 8 single slot 8 port PCI graphic board using ATI Mobility Radeon 9000 chipsets[118] and eight Nvidia 6800 Ultra graphic boards. Both graphic boards run a virtual screen on Windows XP and XFree86 using the XINERAMA extension [119]. As hardware serves an Intel Dual–XEON 3.3GHz clocked system with 4GB RAM.

Beside the single PC solution as base system, for our second application domain we plan to build a OpenGL Chromium based Cluster-System[104] consisting of eight cluster nodes and a cluster super node.

2.7 Conclusion

Visualization of massive data is a challenge. Fortunately, there exist a large number of visualization techniques for different data to be visualized. On the example of geo-related data, we demonstrated that the visualization of this kind of data using traditional maps has several drawbacks if we use them from the information visualization point-of-view. We considered an example of dense point clouds and introduce the *PixelMap* algorithm which re-organizes the screen space to avoid overlapping of the data items by retaining the relative geo-position of the data item as much as possible and place the data item with similar values as close as possible together (clustering). Because the visualization goals are contrary constraints, either all pixels are visualized but the recognition of the map is poor or recognition of the map is good, but not all data are displayed. High resolution walls seem promising as extension for the visualization process. Both, high resolution output devices and visualization techniques can improve the quality of large data exploration. High resolution display walls are relative new technologies and almost always they are unique configurations and since those do not exist as standardized solutions they need a lot of technical effort.

3 Cartogram Drawing

In this section, we introduce a few basic concepts that underlie cartogram drawing. First, we formally define several variants of the problem. Then, we discuss the complexity and theoretical limitations of potential solutions and review the solutions which have been proposed in the literature. Finally, we outline some key observations that are the basis for a new, effective and efficient solution.

3.1 Problem Definition

We assume that the input is a map defined by a set of connected simple polygons (a polygonal mesh) $\mathcal{P}$, and a parameter vector $\mathcal{X}$ that gives the desired values for the area of each polygon. Our goal is to generate contiguous cartograms and therefore, the desired output is a set of connected simple polygons $\overline{\mathcal{P}}$ as well. Let $|p|$ denote the number of vertices, $A(p)$ the area, and $S(p)$ the shape of a polygon p, and $T(\mathcal{P})$ the topology of a set of polygons. Then, the ideal solution of the contiguous cartogram drawing problem can be defined as:

Definition 1 (Contiguous Cartograms - Ideal Solution). *A contiguous cartogram of a set of connected polygons $\mathcal{P} = \{p_1, \ldots, p_k\}$ with respect to the parameter vector $\mathcal{X} = \{x_1, \ldots, x_k\}$, $(\forall j \; x_j > 0)$, is a visualization of the transformed set of polygons $\overline{\mathcal{P}}$, where*

$$T(\overline{\mathcal{P}}) = T(\mathcal{P}) \tag{3.1}$$
$$S(\overline{p_j}) = S(p_j), \forall j = 1, \ldots, k \tag{3.2}$$
$$A(\overline{p_j}) = \tilde{x_j}, \forall j = 1, \ldots, k. \tag{3.3}$$

The desired area $\tilde{x_j}$ of a polygon p_j is defined as

$$\tilde{x_j} = x_j \cdot \frac{\sum_{i=1}^{k} A(p_j)}{\sum_{j=1}^{k} x_j}. \tag{3.4}$$

To simplify the description, the assumption is made that we have only one set of connected polygons (such as the continental United States) and not multiple unconnected sets (such as a world map[1]) Let v_j^i denote the i-th vertex of polygon p_j, α_j^i the angle at the i-th vertex, e_j^i the i-th edge, $|e_j^i|$ the length of edge e_j^i, and $CE(v)$ the cyclic order of edges at vertex v (see figure 3.1).

If we assume that the transformed polygons have the same number of vertices (i.e., $|\overline{p_i}| = |p_i|$), then one way of formalizing the topology and shape preservation constraints is the following:

Definition 2 (Topology Preservation - Preservation of Connecting Vertices).
The topology preservation $T(\overline{\mathcal{P}}) = T(\mathcal{P})$ means that for each vertex $v \in \overline{\mathcal{P}}$ the cyclic order of edges remains the same as in $\mathcal{P}$. More formally,

$$\forall v_j^i \in \mathcal{P}, j = 1, \ldots, k; i = 1, \ldots, |p_j| : \exists \overline{v_j^i} \in \overline{\mathcal{P}}, j = 1, \ldots, k; i = 1, \ldots, |\overline{p_j}| : CE(v_i^j) = CE(\overline{v_i^j}) \tag{3.5}$$

[1] These definitions may be easily extended to multiple polygonal meshes.

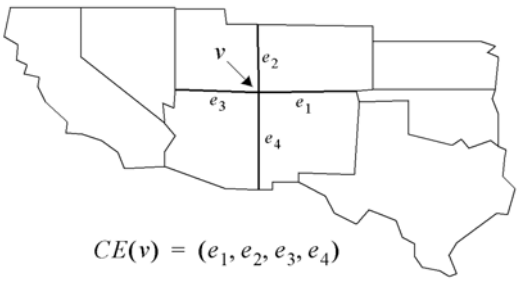
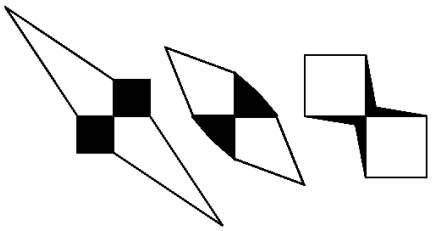


Figure 3.1: Cyclic order of edges

If the cartogram construction algorithm does not provide a mapping to the original polygon set, topology preservation is difficult to test, because as a first step, the isomorphism problem between the two corresponding graphs must be solved. Graph isomorphism is a difficult problem and not known to be polynomial if the graph is not planar. Therefore efficient solutions have to maintain the topology of the original polygon mesh or provide a mapping to the original polygon mesh.



(a) 2×2 checkerboard (b) Relaxed Topology (c) Relaxed Shape

(d) 3×3 (e) Relaxed Topology (f) Relaxed Topology and Shape (g) Relaxed Shape

Figure 3.2: Checker board example

Definition 3 (Shape Preservation - Preservation of Edge Length Ratios and Angles). *Shape preservation $S(\overline{p_i}) = S(p_i)$ means that the edge length ratios of the polygons and the angles are preserved*

$$\forall j = 1,\ldots,k \; \exists c_j \in \mathbb{R} : \; |\overline{e_j^i}| = c_j |e_j^i|, i = 1,\ldots,|p_j|, e_j^i \in \mathcal{P}, \overline{e_j^i} \in \overline{\mathcal{P}} \tag{3.6}$$

$$\forall j = 1,\ldots,k, \forall i = 1,\ldots,|p_j| : \overline{\alpha_j^i} = \alpha_j^i. \tag{3.7}$$

Now let us consider a simple example. Assume that we have a map with the topology of a checker board (see Figure 3.2) and that we want to resize the map according to the color of the fields, scaling white fields by a factor of 2 and black fields by a factor of 0.5. This rescaling is impossible without changing the topology or shapes. So, in general it is impossible to achieve the ideal solution. We state this observation in the following lemma.

Lemma 1 (Impossibility of the Ideal Solution). *The cartogram drawing problem of Definition 1 is unsolvable in the general case, i.e. there exist sets of polygons and parameter vectors such that it is impossible to obtain an ideal solution.*

Proof. Figure 3.2 provides examples of sets of polygons, which do not have ideal cartogram solutions according to Definition 1. □

Constaints	single-polygon	all-polygon	minimum
Topology	$\forall j : d_T(p_j, \overline{p_j}) = 0$	$\sum d_T(p_j, \overline{p_j}) = 0$	-
Area	$\forall j : d_A(\tilde{x_j}, A(\overline{p_j})) \leq \varepsilon$	$\sum d_A(\tilde{x_j}, A(\overline{p_j})) \leq \varepsilon$	$\sum d_A(\tilde{x_j}, A(\overline{p_j})) \xrightarrow{!} \min$
Shape	$\forall j : d_S(p_j, \overline{p_j}) \leq \varepsilon$	$\sum d_S(p_j, \overline{p_j}) \leq \varepsilon$	$\sum d_S(p_j, \overline{p_j}) \xrightarrow{!} \min$

Table 3.1: Possible constraints for cartogram drawing

To derive feasible variants of the problem, we need to relax some of the feature preservation conditions. The *topology error* is measured by the topology distance function d_T

$$d_T : (. \times .) \rightarrow \mathbb{N} \tag{3.8}$$

If topology is the most important property to maintain, the only other conditions left to relax are the shape and area constraints. But there are many ways to go about this. We can explore that in terms of two distance functions - an area distance function d_A

$$d_A : (. \times .) \rightarrow \mathbb{R} \tag{3.9}$$

which measures the distance of the area of a polygon from the desired size, typically, difference in area in the Euclidean plane and a shape distance function d_S

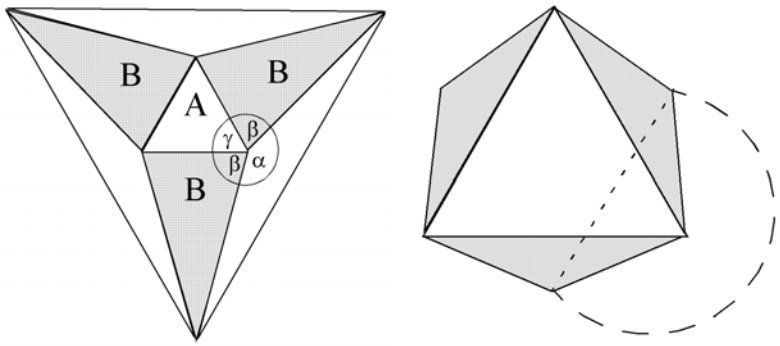
$$d_S : (. \times .) \rightarrow \mathbb{R} \tag{3.10}$$

which measures the similarity of two shapes. Table 3.1 is an enumeration of possible constraints. The first column lists constraints that require a maximum distance for each polygon, the second column lists constraints that require a maximum distance for the sum of the distances of all polygons, and the third column lists minimum constraints for the sum of distances. By combining the different area and shape constraints in table 3.1 we can construct variants of the cartogram drawing problem. A useful combination would be, for example, a restriction of the solution space to solutions where the shape of each polygon has at least a certain similarity to its original shape and the sum of all area differences is minimal. In the following, we discuss the different variants of the problem and their complexity.

3.2 Solvability and Complexity of the Problem

As shown by Lemma 1, in general it is impossible to find an ideal solution of the cartogram drawing problem. If we now consider the variants that may be constructed by a combination of the constraints in table 3.1, it turns out that a large number of these are also unsolvable in the general case.

Lemma 2 (Impossibility of the Solution of Problem Variants). *Any variant of the cartogram drawing problem that involves the single-polygon area constraint or the all-polygon area constraint is unsolvable in the general case, i.e. there exist sets of polygons $\mathcal{P}$ and parameter vectors $\mathcal{X}$, such that for any ε the problem variants do not have a valid, topology-preserving solution.*



(a) (b)

Figure 3.3: Impossible cartogram drawing problem

Proof. In figure 3.3a, we show an example of a symmetric cartogram consisting of 7 polygons. If the parameter vector for scaling the polygons requires the white polygons to become larger and the black ones to become smaller, we can easily construct an impossible case. Due to the symmetric construction of the polygons, without loss of generality we can assume that one angle $\gamma \leq \frac{\pi}{3}$. Thus,

$$\alpha = 2\pi - 2\beta - \gamma \geq 2\pi - 2\beta - \frac{\pi}{3} \tag{3.11}$$

For the above mentioned resize requirements (triangle *A* very large and triangles *B* very small), $\beta \to 0$ and therefore

$$\alpha \geq 2\pi - \frac{\pi}{3} = \frac{5}{3}\pi \quad \Longrightarrow \quad \alpha > \pi \tag{3.12}$$

and thus the topology can not be preserved as shown in figure 3.3b. □

This means that only variants of the problem that use the minimum-area condition are solvable and this is true for any combination with a shape constraint. The solvability is trivial to see since there is at least the identity solution which has a perfect shape preservation but a rather bad value for the area difference. As the following lemma shows, the determination of the actual solution with the minimum area difference, however, is a computationally hard problem.

It is likely that the cartogram problem with the minimum-area condition represents a $\mathcal{NP}$–hard optimization problem.

In using this variant of the problem one easily observes that there is little freedom to improve the second important parameter, namely the shape. In most cases, the minimum area condition will provide some solution which is best optimized according to the area condition but does not take the shape similarity into account. There might be, for example, a solution which much better preserves the shape but is a little bit worse concerning the area condition. To allow the shape constraint to have an impact on the solution, we have to adapt our constraints. In principle, there are two possibilities. The first is to determine the minimum area difference which is possible and then, allow a certain maximum deviation from this minimum difference for finding the best shape. More formally, this may be defined as follows.

Definition 4 (Variant 1 of the Contiguous Cartogram Problem).
Given a set of polygons $\mathcal{P}$, a parameter vector $\mathcal{X}$, and an error value ε, the Contiguous Cartogram problem may be defined as a transformed set of polygons $\overline{\mathcal{P}}$ for which the following two conditions hold:

$$\sum_{j=1}^{k} d_A(\tilde{x_j}, A(\overline{p_j})) \leq \underset{\overline{\mathcal{P}}}{MIN} \left(d_A(\tilde{x_j}, A(\overline{p_j})) + \varepsilon\right) \tag{3.13}$$

$$\sum_{j=1}^{k} d_S(S(p_j), S(\overline{p_j})) \stackrel{!}{\to} \min \tag{3.14}$$

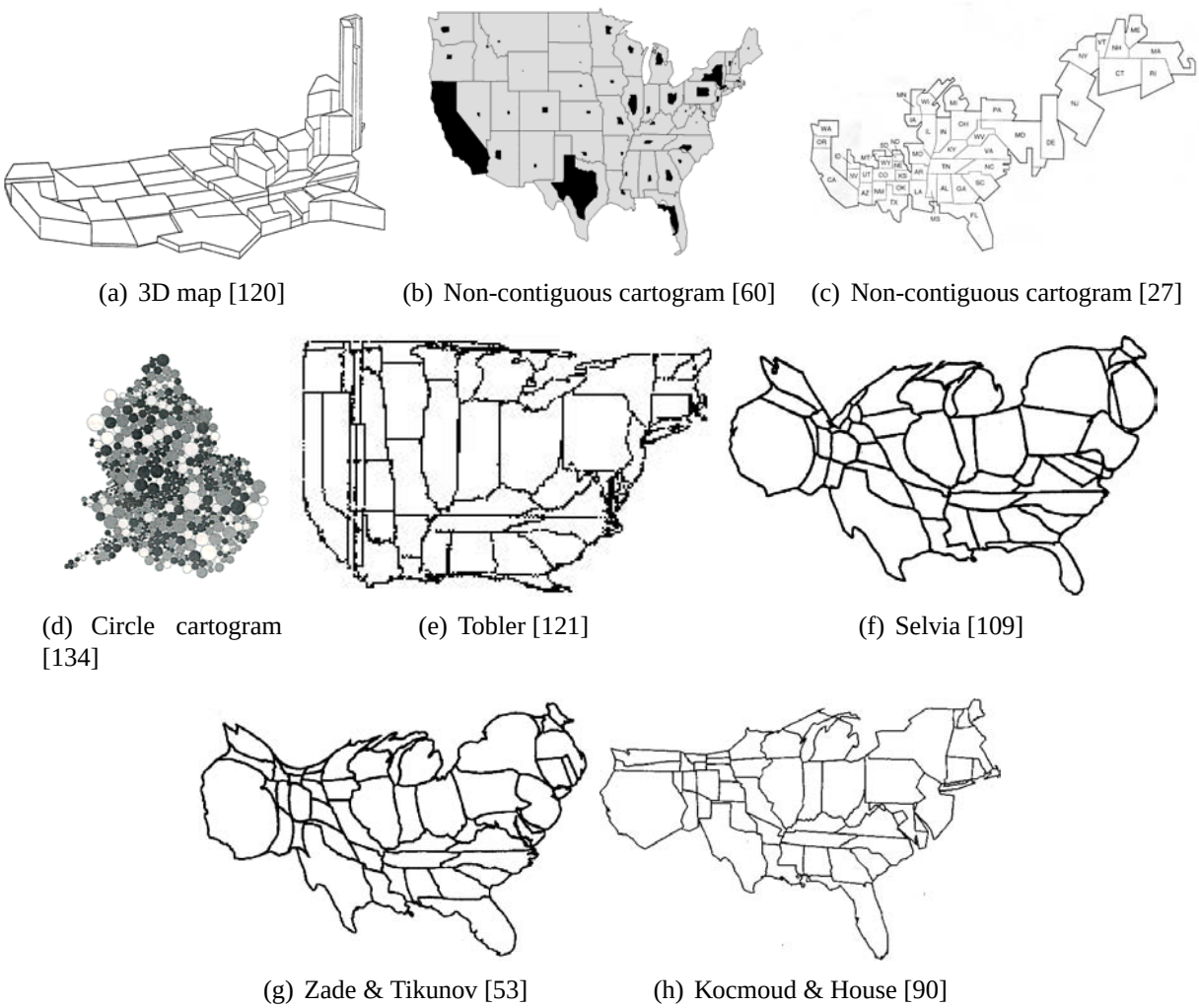
A second possibility is to normalize the area and shape distances and to use a weighted mean of the normalized distances as a combined optimization criterion.

Definition 5 (Variant 2 of the Contiguous Cartogram Problem).
Given a set of polygons $\mathcal{P}$, a parameter vector $\mathcal{X}$, and importance factors for the area and shape distances ($w_a, w_s \geq 0$), the contiguous Cartogram problem may be defined as the transformed set of polygons $\overline{\mathcal{P}}$ for which

$$w_a \cdot \sum_{j=1}^{k} d_A(\tilde{x_j}, A(\overline{p_j})) \tag{3.15}$$

$$+ \quad w_s \cdot \sum_{j=1}^{n} d_S(p_j, \overline{p_j}) \stackrel{!}{\rightarrow} \min_{w_a, w_s \geq 0}. \tag{3.16}$$

There are other meaningful and solvable variants of the problem which, for example, also include the single-polygon constraints (see table 3.1). Most currently available algorithms try to solve the problem according to definition 4 or definition 5. This seems sufficient for some applications but there are others where additional constraints seem necessary. In the following, we discuss some important observations which are the basis for our final definition and also the key to an efficient solution of the problem.



(a) 3D map [120] (b) Non-contiguous cartogram [60] (c) Non-contiguous cartogram [27]

(d) Circle cartogram [134] (e) Tobler [121] (f) Selvia [109]

(g) Zade & Tikunov [53] (h) Kocmoud & House [90]

Figure 3.4: Cartogram drawing methods

3.3 Related Work

Cartographers and geographers have used cartograms long before computers were available to make displays [106, 107, 58]. References date back as far as 1868 (see remarks on Levasseur in [47] on page 355). A short historical overview can be found in [27, 122].

The basic idea of a cartogram is to distort a map by resizing its regions by some geographically-related parameter. Example applications include population demographics [120], election results [90], and epidemiology [54].

Because cartograms are difficult to make manually, the study of programs to draw them is of interest.

Cartograms can made by contiguous or non-contiguous distortions. The non-contiguous case is much easier, since the input map topology does not have to be preserved. As seen in figure 3.4, hand-made non-contiguous cartograms have been drawn with overlapping or touching circles, by eliminating some of the original map's adjacencies, or even by drawing disconnected shapes over the original regions [60, 134].

Most previous attacks on automated drawing of contiguous cartograms do not yield results comparable to good hand–made drawings. One reason, first pointed out by Dent [25, 26], is that straight lines, right angles and other features considered important in human recognition of cartograms are obliterated. Methods that are radial in nature such as the conformal maps proposed by Tobler [120], the radial expansion method of Selvin et. al. [109] and the line integral method of Guseyn-Zade and Tikunov [53] do not provide acceptable results, since the shapes of the polygons are heavily deformed (see figure 3.4). Likewise, the pseudo-cartograms of Tobler expand the lines of longitude and latitude to achieve a least root mean square *area error* [121]. Very similar drawings are made by approaching the problem as distortion viewing by nonlinear magnification [64, 99, 17, 66]. Jackel [60] applied radial forces to change the size of polygons, moving the sides of each polygon relative to its centroid, but the solver runs very slowly (taking 90 minutes to perform 8 iterations on a map of 6 New England states of the U.S.) and seems to have problems with non-convex input polygons and with self-intersections in the output, which is consistent with our early experiments with a similar approach.

Another family of approaches operates on a a grid or mesh imposed on the input map. The „piezopleth" method of Cauvin, Schneider and Cherrier transforms the grid by a physical pressure load model [18]. Dorling's cellular automaton approach trades grid cells until each region achieves the desired number of cells [30]. The combinatorial approach of Edelsbrunner and Waupotitsch [35] computes a sequence of piecewise linear homeomorphisms of the mesh that preserve its topology. While the first method is good at preserving the shape of the polygons, the second method allows a very good fit for area but only poor shape preservation.

A synthesis of both approaches was recently described by Kocmoud and House, who propose a force-based model and alternately optimize the shape and the *area error* [90]. Although the results are better than most other methods, the complex optimization algorithm has a prohibitively high execution time. Kocmoud and House report a time of 18 hours for a modest-sized map with 744 vertices. An other cartogram approach based on diffusion equation currently appeared in [50]. Since the distortion depends of a spacial point set as input (similar to Tobler's Pseudo cartograms [121]) the shape of the polygon can not be taken into account and therefore the approach is different to our one. In figure 3.4, we present population cartograms generated by several of the methods we have mentioned.

3.4 Important Observations

The current solutions have two major problems: first, the high time complexity of the algorithms restricts their use to static applications with a small number of polygons and vertices. Second, they have very limited shape preservation. Although the recent work by Kocmoud and House provides nice results, some effectiveness problems remain. One problem is the significant deformation of the global shape. In evaluating the different heuristic solutions which have been proposed so far, we found that the insufficient

preservation of the global shape is one of their major problems. According to our experience, however, the global shape is one of the most important factors for cartograms to be effective, and it is certainly at least as important as the preservation of interior polygon shapes. In our definition of cartogram drawing, besides the shape and area constraints of table 3.1 we therefore explicitly include a global shape constraint which may be again either a single-polygon, all-polygon, or minimum constraint for the global shape. There may be multiple global shapes as they occur, for example, on a world map. So there is one global polygon for each connected component of the map. If that is the case we denote each component of a map $\mathcal{P}$ with an indices. If $GP_i = GP(\mathcal{P}_i)$ denotes the set of global polygons which may be derived from the set of polygons $\mathcal{P}_i$. $GP(\mathcal{P})$ is defined as follow:

$$GP(\mathcal{P}) = \{v \in \mathcal{P} : |edges(v)| > |polygons(v)|\}. \tag{3.17}$$

The global shape constraints may formally be described as given in table 3.2. Our final definition of the cartogram drawing problem uses a weighted minimum of area, shape, and global shape constraints.

Definition 6 (Variant 3 of the Contiguous Cartogram Problem). *Given a set of polygons $\mathcal{P}$, a parameter vector $\mathcal{X}$, and importance factors for the area, shape, and global shape constraints w_a,w_s, and w_{gs}, the contiguous Cartogram problem may be defined as a transformed set of polygons $\overline{\mathcal{P}}$ for which*

$$w_a \cdot \sum_{j=1}^{k} d_A(\tilde{x_j}, A(\overline{p_j})) \tag{3.18}$$

$$+ \quad w_s \cdot \sum_{j=1}^{k} d_S(p_j, \overline{p_j}) \tag{3.19}$$

$$+ \quad w_{gs} \cdot \sum_{r} d_S(GP(\mathcal{P}), GP(\overline{\mathcal{P}})) \stackrel{!}{\rightarrow} \min_{w_a, w_s, w_{gs} \geq 0} \tag{3.20}$$

single-polygon	$\forall r : d_S(GP(\mathcal{P}_r), GP(\overline{\mathcal{P}}_r)) \leq \varepsilon$
all-polygon	$\sum_r d_S(GP(\mathcal{P}_r), GP(\overline{\mathcal{P}}_r)) \leq \varepsilon$
minimum	$\sum_r d_S(GP(\mathcal{P}_r), GP(\overline{\mathcal{P}}_r)) \stackrel{!}{\rightarrow} \min$

Table 3.2: Global polygon constraints for cartogram drawing

Let us now focus on some important observations which are crucial for an efficient solution of the problem. One important observation is that in practice only very few vertices are actually important for defining the shapes of the polygons. Considering the US map, as an example, we found that in addition to a restricted number of outer vertices, only a limited number of interior vertices are actually relevant. Note also that the importance of polygons and their vertices largely depends on their size (which is directly related to the parameter vector) and on the length of the edges and the angles between them. In our new algorithm, we give special consideration to these facts and determine the importance of vertices based on these observations. A second observation is that – in order to obtain good results – the *shape error* has to be controlled explicitly, which is not done sufficiently in previous approaches. A last observation is that the high time complexity of most algorithms proposed previously is due to a complex and time-consuming optimization. In most cases, however, it is possible to locally reposition vertices and improve the *area error* while retaining the shape. To obtain good solutions, our algorithm iteratively repositions vertices based on scanline-defined locality measures with an explicit *shape error* control function.

3.5 Map Simplification

3.5.1 Introduction

Map simplification is an important task for the cartogram generation. Beside our observations in the last chapter, maps consists often of thousands of points [31, 7, 36]. But only some hundred are needed for visualization on a regular display. This high concentration of points is negative for a fast computation. Furthermore, large points can induce varieties in special cases which have to be studied. In this section we introduce our approach for reducing inner and outer nodes of a map.

The Table 3.3 gives an overview of the size of segments, nodes, and polygons for some maps used in this thesis.

Map	Segments	Node	Polygons
U.S. state-level	1286	808	48
U.S. county-level	20901	7609	3085
Texas county-level	1581	599	254
Germany state-level	16984	8478	15
Germany county-level	50748	25879	434
Germany block-level	125402	60139	NA
World state-level	33182	25624	NA
Afrika	4411	2590	46

Table 3.3: Number of segments, nodes, and polygons for some maps used in this thesis.

The studying of polygon line simplification algorithms goes back to the seventies. The most famous procedure is the Douglas-Peucker algorithm [31]. Douglas-Peucker works as follows. As input a polygon line and a tolerance value is given. The algorithm starts with the two end points and iterates over the point set of the polygon line. If the point with the maximum distance between the two end points is greater than the given tolerance value, the point belongs to the reduced polygon line. Furthermore, the polygon line is split into pieces, and the algorithm continues recursiv. If all points are closer than the tolerance value, the points are not important and they can be reduced. Because our motivation is based on the study of [25, 26] (see section 3.3 of this chapter) we used a more straightforward approach as an alternative.

3.5.2 Reduction of Global Polygon

As mentioned in section 3.4, preserving the global shape is very important in making recognizable cartograms. This is taken into account by the decimation algorithm by simplifying the global and inner polygons differently.

A key observation is that the importance of the vertices of a polygon can greatly vary. Vertices on angles close to 180 degrees and those with short edges make almost no noticeable difference in the shape of a polygon, while others with sharp angles or long edges have a significant effect. The basic idea of the global polygon reduction algorithm is to rate the importance of each vertex according to these criteria. Then, iteratively, the least important vertices are removed. To maintain the topology, only vertices that do not belong to multiple polygons are removed. To formalize the global reduction algorithm, we first define the notion of a vertex' importance as

$$I(v,\sigma) \quad = \quad Sig(\alpha^v,\sigma)\cdot|e_1^v|\cdot|e_2^v| \tag{3.21}$$

where e_1^v and e_2^v are the two edges of vertex v and $Sig(\alpha^v,\sigma)$ is a function denoting the significance of the angle α^v at vertex v. The significance function $Sig(\alpha,\sigma)$ is important because different angles have a

specific impact on the shape of the polygons. Sharp angles and angles close to 90° are more important than obtuse angles (c.f. [25, 26]) and the significance function therefore assigns higher values to sharp angles and lower values for obtuse angles. For our algorithm, we use

$$Sig(\alpha,\sigma) \quad = \sum_{\mu\in\{0,\frac{\pi}{2},\frac{3\pi}{2},2\pi\}} \exp^{-\frac{(\alpha-\mu)^2}{2\sigma^2}} \tag{3.22}$$

as the significance function. This function has peaks for $\alpha = 0, \frac{\pi}{2}, \frac{3\pi}{2}, 2\pi$ and is close to zero for $\alpha = \pi$. The function is defined for $\alpha \in]0, 2\pi[$ and σ is chosen to be $0.2 \cdot \pi$. Figure 3.5 shows a plot of this function.

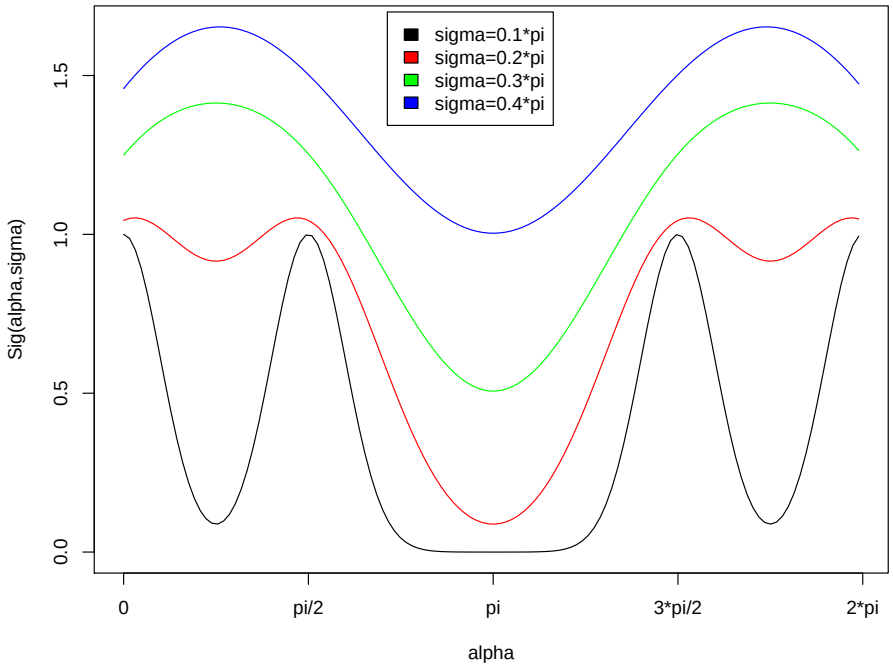


Figure 3.5: Mesh reduction significance function – The plot illustrates the described significance function. The red curve shows $Sig(\alpha^v, \sigma)$ if we use $\sigma = 0.2 \cdot \pi$.

To formalize the global reduction algorithm, we first define the global polygon as a subset of the vertices of $\mathcal{P}$. For each polygon $(p_j)_{j=1\ldots k}$, the portion GP_j of the global polygon $\mathcal{GP}$ were defined in chapter 3 in equation 3.17

The global polygon is defined as $\mathcal{GP} = \cup_{j=1}^{k} gp_j$. The algorithm for the reduction of the global polygon is shown in figure 1. Note that vertices are only considered for removal if they do not belong to multiple polygons (see initialization of V in figure 1) and they are only removed if the induced area difference is smaller than a given constant $MaxAreaDiff$. Note also that the area $A_s(p)$ of a polygon p is determined as if the polygon is perfectly scaled according to the parameter vector $\mathcal{X}$.

3.5.3 Reduction of Inner Polygons

To position interior vertices, we can use again an iterative vertex removal process. A more efficient alternative is based on the observation that for most maps only the connecting interior vertices are important. Instead of iteratively removing unimportant vertices, we therefore take a more direct approach and remove all vertices not common to more than two polygons (non-connecting vertices). In some cases, the shape deformation and area error introduced by this reduction is unacceptably high. We therefore re-introduce a few additional vertices. See figure 2 for the complete algorithm.

ReduceGlobalVertices($\mathcal{P}$**,** $\mathcal{GP}$**,** $MaxAreaDiff$**)** {
  /* Only consider vertices if they are not part of multiple polygons */
  $V = \{v \in gp_j \mid v \notin gp_m \text{ for } m \neq j\}$
  do {
    /* Determine the least important vertex */
    $\overline{v} = \{v \in V \mid \overset{MIN}{v \in V} I(v)\}$
    /* Determine the polygon containing the least important vertex */
    $j = \{j \in \{1 \ldots k\} \mid \overline{v} \in p_j\}$
    if $(|A_s(p_j \setminus \{\overline{v}\}) - A_s(p_j)| \leq MaxAreaDiff)$
      $p_j = p_j \setminus \{\overline{v}\}$;
    $V = V \setminus \{\overline{v}\}$
  } **while** $V \neq \{\}$
}

Algorithm 1: Reduction of global vertices

Figure 3.6 shows an example polygon (see figure 3.6a), a polygon reduced of its interior vertices common to more than two polygons (see figure 3.6b), and the final polygon after re-introducing a few additional vertices (see figure 3.6c). In practice only few polygons need the additional vertices, so the likelihood of re-introducing vertices that were removed is low (see figure 3.6).

3.6 Conclusion

The theoretic work and the study on that chapter is the base for the cartogram algorithm on the following pages.

The algorithms for map simplification proposed in this chapter work very well for our input data as it can be seen in the application part of the thesis. The reduction algorithm are also efficient as it is demonstrated in the next chapter on page 41. Nevertheless, further improvements can be made. First, to amplify the quality we can make use of the shape similarity function (see appendix A) for an extra test which has to be passed during the reduction process. The reduction of a global or local node is only made if the shape error between both polygon are less than a given threshold.

Second, since on cartograms we want to stress polygons with a high statistical value, the circumference is larger as well and therefore the shape thresholds for this polygons should be set more sensitive. Both reduction procedures in that chapter can be easily adapted when we weight the desired value of an area with the significance of a map node.

ReduceInnerVertices($\mathcal{P}$**,** $\mathcal{GP}$**,** *MinAreaImpr*$)$ $\{$

/* all interior vertices */

$\mathcal{IV} = \overset{\cup}{}_{j=1\ldots k}\{v \mid v \in p_j\} \setminus \overset{\cup}{}_{j=1\ldots k}\{v \mid v \in gp_j\}$

/* connecting interior vertices */

$\mathcal{CV} = \{v \in \mathcal{IV} : |edges(v)| > 2\}$

/* remove all non-connecting vertices */

forall $(p_j \in \mathcal{P})$

$p_j = \{c \in p_j \mid c \notin (\mathcal{IV} \setminus \mathcal{CV})\}$;

/* reintroduce important non-connecting vertices */

forall $(p_j \in \mathcal{P})$ $\{$

/* Assume that the adjacent inner vertices $\{v_1, \ldots, v_m\} \in (\mathcal{IV} \setminus \mathcal{CV})$ have been removed between vertex v_s and v_e of polygon p_j in step 1. */

forall $((v_s, v_e)$ removed in step 1) $\{$

$\overline{n} = \overset{MIN}{n \in 1\ldots m}\left(\overset{MAX}{(v_{i_1},\ldots,v_{i_n}) \in \{v_1,\ldots,v_m\}, v_{i_l} \neq v_{i_k}} |A_s(p_j \cup \{v_{i_1}, \ldots, v_{i_j}\}) - A_s(p_j)|\right)$

$p_{tmp} = p_j \cup \{v_{i_1}, \ldots, v_{i_{\overline{n}}} : \overset{MAX}{(v_{i_1},\ldots,v_{i_{\overline{n}}}) \in \{v_1,\ldots,v_m\}, v_{i_l} \neq v_{i_k}\}} |A_s(p_j \cup \{v_{i_1}, \ldots, v_{i_j}\}) - A_s(p_j)|$

$\}$

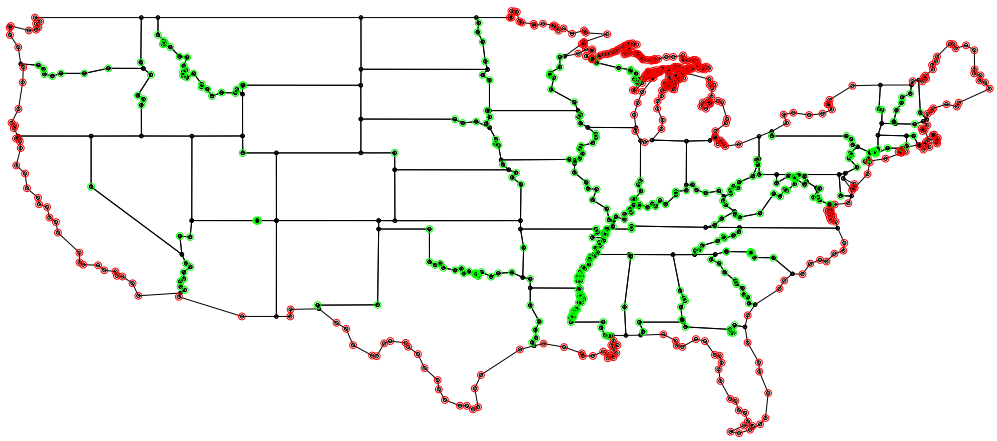
if $(|A_s(p_{tmp}) - A_s(p_j)| \geq MinAreaImpr)$

$p_j = p_{tmp}$;

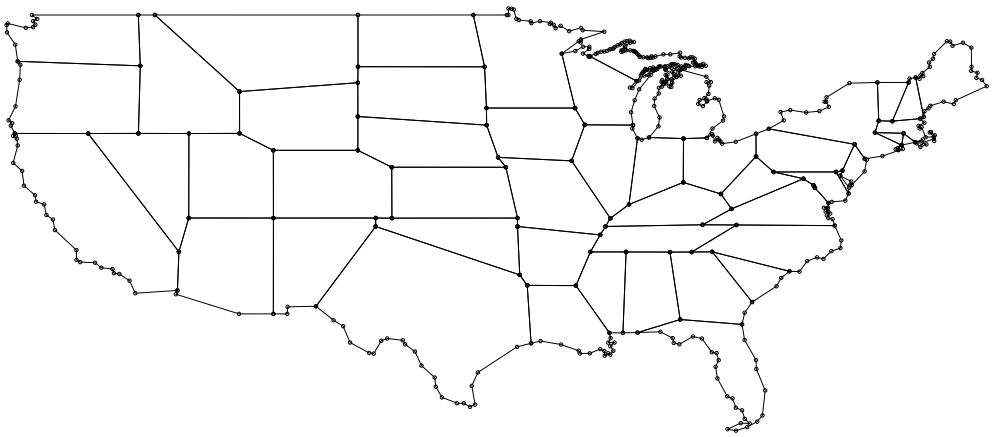
$\}$

$\}$

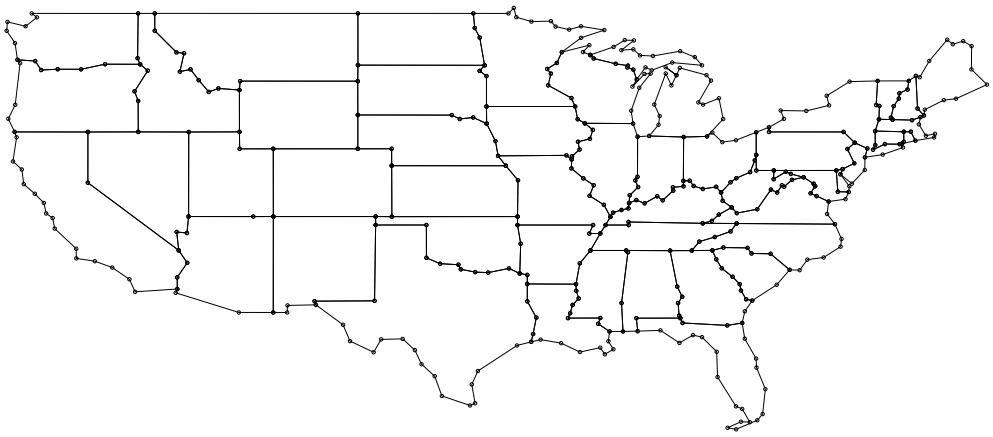
Algorithm 2: Reduction of interior vertices



(a) Un-reduced traditional U.S. map (#nodes: 808, #edges: 1286)



(b) Without inner connecting vertices (#nodes: 487, #edges: 644)



(c) Reduced Map (#nodes: 413, #edges: 736) where $I(v, 0.2) \geq 0.37$ and $MaxAreaDiff = 0.1$

Figure 3.6: U.S. map simplification – green - reducible inner node, red - reducible global nodes

4 *CartoDraw*: A Fast Algorithm for Generating Contiguous Cartograms

4.1 Introduction

The main objective of our new cartogram drawing algorithm is a fast generation of cartograms of acceptable quality. Because input maps often have far more vertices than are needed to compute good cartograms, the first step is an intelligent decimation and has been described in the previous chapter. The reduction step is now followed by the central heuristic, scanline-based repositioning of vertices. We first reposition vertices of the global polygon(s), and then interior vertices. Scanlines can be restricted to vertical and horizontal lines determined automatically, or may be arbitrarily positioned line segments of any length, entered interactively. This follows the human-guided local search paradigm proposed by Marks et al [3]. In each step, the shape of the modified polygon mesh is controlled by the *shape error* function. The last step is the fitting of the undecimated polygons to the decimated mesh to obtain the output cartogram. By exploiting the potential for pre-computation and fast local optimization, our algorithm runs quickly enough to support dynamic displays with high update rates on maps having dozens of polygonal regions.

4.2 Problem Definition

We can pose cartogram generation as a map deformation problem. Therefore we have to give a formulation of how to determine a near optimal cartogram $\overline{\mathcal{P}}$ which can be computed from a given planar map $\mathcal{P}$. The input map is a planar polygon mesh, and a value associated with each region (face) that is its desired fraction of the total cartogram area. The goal is to deform the map so that the area of each region is close or equal to the target while preserving the map's connectivity and the shapes of its faces, including the outermost one.

4.2.1 Constraints

When determining the cartogram $\overline{\mathcal{P}}$ we have to assure that the topology of the map regions is preserved. We call a cartogram feasible if the topology is preserved and denote the set of feasible cartograms by $\mathcal{M}$.

4.2.2 Objective Functions

Assuming the topology of the map is preserved the quality of the resulting contiguous cartogram depends on two aspects: First, is the map recognizable and second, does the area reflect the statistical value. Both requirement are difficult to comply since area and shape preservation are conflicting goals. The functions measure topology, area and shape are described as follows:

Topology Error Function Topology preservation means that $\mathcal{P}$ and $\overline{\mathcal{P}}$ must be homeomorphic. This may be defined in terms of Betti numbers [15, page 97], or by explicitly testing that there is one – to – one mapping of faces from $\mathcal{P}$ to faces of $\overline{\mathcal{P}}$ that preserves adjacencies as stated in equation 4.1.

$\psi : \mathcal{P}(V) \rightarrow \overline{\mathcal{P}}(V)$ with

$$\forall v, w \in \mathcal{P}(V) : v \neq w \Rightarrow \psi(v) \neq \psi(w) \wedge \psi^{-1}(\psi(v)) = v \tag{4.1}$$

The algorithm in that chapter strictly retains the topology of the map regions. Behind that it is possible to get a measure for topology violations. This can be done by summing up the differences of the cyclic orders $CE(v)$ and $CE(\psi(v))$ of a node $v \in \mathcal{P}$.

$$\begin{aligned} d_T &= d_T(\mathcal{P}, \overline{\mathcal{P}}) && (4.2) \\ &= \frac{1}{2} \sum_{v \in \mathcal{P}} \sum_{i=1}^{|CE(v)|} d_{CE}\left(CE(v)_{(i)}, CE(\psi(v))_{(i)}\right) && (4.3) \end{aligned}$$

where $d_{CE} : \mathbb{N} \times \mathbb{N} \rightarrow \{0,1\}$ is defined as:

$$d_{CE}(a,b) = \begin{cases} 0 \text{ if } a = b \\ 1 \text{ otherwise} \end{cases} \tag{4.4}$$

Note that $CE(v)$ gives an ordered set as result[1].

Area Error Function The objective of cartogram generation is to obtain a set of polygons where the area of the polygons corresponds to values given in a data vector $\mathcal{X}$. In each step of the algorithm, the *area error* function is needed to determine the reduction of the *area error* achieved by applying a given scanline. The relative *area error* $\tilde{d_A}(\overline{p_j})$ of a polygon $\overline{p_j}$ can be computed as:

$$\tilde{d_A}(\tilde{x_j}, \overline{p_j}) = \frac{|\tilde{x_j} - A(\overline{p_j})|}{\tilde{x_j} + A(\overline{p_j})} \tag{4.5}$$

Hence, the *area error* d_A for the set of polygons $\mathcal{P}$ is defined as

$$\begin{aligned} d_A &= d_A(\tilde{\mathcal{X}}, \overline{\mathcal{P}}) && (4.6) \\ &= \sum_{j=1}^{|\overline{\mathcal{P}}|} \left(\tilde{d_A}(\tilde{x_j}, \overline{p_j}) \cdot \frac{\tilde{x_j}}{\sum_{j=1}^{|\overline{\mathcal{P}}|} \tilde{x_j}} \right) && (4.7) \end{aligned}$$

Shape Error Function In addition to reducing *area error*, the cartogram generation process also aims at retaining the original shapes. To assess shape preservation, we need a shape similarity function that compares the new shape of a polygon with its original shape. Defining a useful shape similarity function is in itself a difficult problem, since the similarity measure should be:

- translation-invariant,
- scale-invariant, and
- partially rotation-invariant.

From CAD research it is known that the Euclidean distance in Fourier space is useful for measuring shape similarity [67, 8]. The Fourier transformation approach which we used in for the *CartoDraw* [75] algorithm is described in chapter A.

In the following we want to describe a variant which use some advantages of our transformation. Since the topology of the mesh does not change we can use equation 4.1 for the *shape error* computation. The *shape error* of to isomorph polygons can be determined as follows:

[1] We implement ψ as `node_array` using the LEDA library [97].

$$\tilde{d_S}(p,\overline{p}) = \frac{1}{2\pi}\sum_{j=1}^{|p|}|\underbrace{\measuredangle v_{j\ominus 1}v_jv_{j\oplus 1}}_{\in p} - \underbrace{\measuredangle \psi(v_{j\ominus 1})\psi(v_j)\psi(v_{j\oplus 1})}_{\in \overline{p}}| \tag{4.8}$$

The *shape error* d_S of the $\overline{\mathcal{P}}$ is:

$$d_S = d_S(\mathcal{P},\overline{\mathcal{P}}) \tag{4.9}$$

$$= \sum_{i=1}^{|\mathcal{P}|}\tilde{d_S}(p_i,\overline{p_i}) \tag{4.10}$$

4.2.3 Formulation of the Optimization Problem

After the observations mentioned in chapter 3 and having introduced the constraints and the objective functions, the Contiguous Cartogram Problem can be stated as optimization problem:

Definition 7 (The Contiguous Cartogram Optimization Problem).
Input: *A planar polygon mesh* $\mathcal{P}$ *consisting of polygons* $p_1,\ldots,p_k$*, values* $\tilde{\mathcal{X}} = (x_i)_{i=1,\ldots,k}$ *with* $x_i > 0, \sum_{i=1}^{k} x_i = 1$*, and* $\forall i,j \in \{1,\ldots,k\} \wedge i \leq j \rightarrow x_i \geq x_j$*, i.e. the elements of* $\tilde{\mathcal{X}}$ *are sorted by non-increasing values. Let* $A(p_i)$ *denote the normalized area of polygon* p_i *with* $A(p_i) > 0, \sum A(p_i) = 1$.

Output: *A* topology-preserving *polygon mesh* $\overline{\mathcal{P}}$ *consisting of polygons* $\overline{p}_1,\ldots,\overline{p_k}$ *such that the function* f *is minimized, where*

$$f = w_t \cdot d_T + w_a \cdot d_A + w_s \cdot d_S + w_{gs} \cdot \tilde{d_S}(GP(\mathcal{P}), GP(\overline{\mathcal{P}})) \tag{4.11}$$

Alternatively, topology-preserving in that manner means that $d_T = 0$. Using the weights w_t, w_a, w_s and w_{gs} with $(w_t, w_a, w_s, w_{gs} \geq 0)$ the user has an explicit control of *area error* and *shape error* functions.

We can not expect that the problem defined in definition 7 has an ideal solution. Of course not as discussed in chapter 3 this can only appear if the parameter vector equals to the areas of the map regions. A cartogram $\overline{\mathcal{P}}$ is called optimal if all objective functions are minimized simultaneously. Figure 4.1 shows the *region of objective function values* [101] where for illustration only the *area error* d_A and the *shape error* d_S are observed and we assume that the maps topology is preserved. The cartogram $\overline{\mathcal{P}}^{ideal}$ would be an ideal solution. On that map *shape error* and *area error* equals zero. The problem instance on the figure does not show an optimal solution but it shows two efficient points $\overline{\mathcal{P}}'$ and $\overline{\mathcal{P}}''$. $\overline{\mathcal{P}}'$ minimize the *area error* where $\overline{\mathcal{P}}''$ minimize the *shape error*. $\overline{\mathcal{P}}''$ is a local minimum.

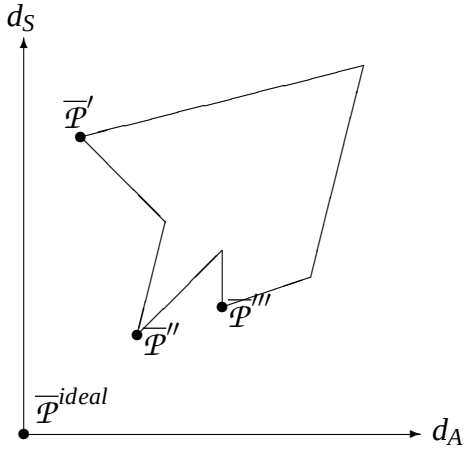


Figure 4.1: Region of the objective function

Behind all that constrains, as stated above, the computation time is a very important factor for cartogram generation. It does not make sense to find an optimal solution after hours or days, if we can find a solution close to the optimal solution after seconds or minutes. Especially in information visualization, were the users have many interactions with the visualization tools, we need a very fast cartogram generation procedure which allows us to get the result in a very short time period.

4.3 The *CartoDraw* Algorithm

4.3.1 Basic Idea

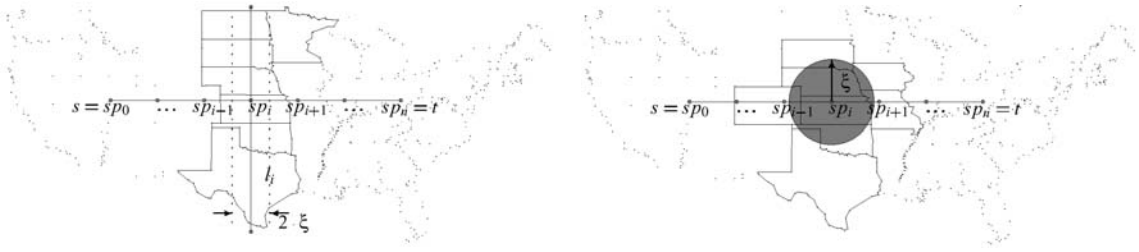
The main idea of the *CartoDraw* algorithm is to incrementally reposition the vertices along a series of scanlines. A scanline is a line segment of arbitrary length and position. Each scanline defines a scan section, orthogonal to the scanline. All points within a scan section are repositioned in a single step. For each section on a scanline, a target scaling factor for each of its polygons is determined according to their *area error* factors. Vertices are then repositioned according to the polygon scaling factors and distances to the scanline. The repositioning may be parallel or orthogonal to the scanlines. If the *shape error* introduced by applying a scanline exceeds some threshold, its candidate vertex repositionings are discarded.

Scanlines should be applied to parts of the map where the *area error* is large and there is still potential for improvement. A simple approach to scanline generation is to use horizontal and vertical line segments positioned on a regular grid. Significantly better results can be obtained by a manual scanline placement, guided by the shape of the input polygons and the local potential for improvement. Note that the incremental repositioning of vertices per scanline application is intentionally small, compared to the expected change in area. This means the same scanline may need to be applied many times to make large adjustments in an area.

Before we describe the main CartoDraw algorithm, we first introduce the *scanline algorithm.*

4.3.2 Scanline Algorithm

The key to the *CartoDraw* algorithm is the scanline heuristic, which incrementally repositions vertices along scanlines. A scanline *sl* is a line segment of arbitrary position and length and is partitioned into n portions of length $\frac{|sl|}{n}$. The scanline section points $(sp_i)_{i=0,\ldots,n}$ define $n+1$ sections of the polygon mesh, which are orthogonal to the scanline (see figure 4.2(a)). In one step of the scanline algorithm, all vertices



(a) Scanline section (b) Scanline section with limited range

Figure 4.2: Scanline algorithm notations and overview

$v \in V_i$ within a certain distance $(\xi = \frac{|sl|}{2n})$ of l_i are considered for incremental repositioning (see figure 4.2a). Let SP_i be the set of polygons (by index number) which have at least one vertex in scanline section $(i)_{i=0,\ldots,n}$.

Then, the scaling factor SF_i is determined according to the *area error* of all polygons p in section i:

$$SF_i = const \cdot \sum_{r \in SP_i} \left(\frac{\tilde{x}_r - A(p_r)}{\tilde{x}_r + A(p_r)} \cdot \frac{\tilde{x}_r}{\sum_{l \in S_i}^{n} \tilde{x}_l} \right). \tag{4.12}$$

Next, we have to determine the direction $o(v)$ of a vertex v and apply the scaling factor SF_i to reposition the vertex. The repositioning can be done either in the direction of the scanline $(direction = scanline)$ or in the direction of the section line l_i. The algorithm is shown in algorithm 3. Note that the scanline sections always span the full range orthogonal to the scanline of the polygon net. If we want to restrict the changes to be local in both directions, we can optionally limit the considered polygons to those close to the scanline (see figure 4.2(b)). This option is not reflected in the algorithm shown in algorithm 3. A demonstration of the scanline algorithm functionality is given in figure 4.3.

Scanline$(\mathcal{P}, \tilde{\mathcal{X}}, sl)\{$
  /* set of scanline section points which lie on sl */
  foreach $sp \in \{sp_i \mid sp_i = s + \frac{i}{n}(t-s), i = (0 \ldots n)\}$ {
    /* section line */
    $l_i = sp \perp \vec{sl}$
    /* vertices v in scanline section i, i.e. closer than ξ to l_i */
    $V_i = \{v \in V \mid |l_i - v| \leq \xi)\}$
    /* numbers of polygons which contain at least one vertex from V_i */
    $SP_i = \{j \in \mathbb{N} \mid \exists v : \; v \in V_i \wedge v \in p_j\}$
    /* compute scaling factor */
    $SF_i = const \cdot \sum_{r \in S_i} \left(\frac{\tilde{x}_r - A(p_r)}{\tilde{x}_r + A(p_r)} \cdot \frac{\tilde{x}_r}{\sum_{l \in S_i}^{n} \tilde{x}_l} \right)$
    foreach $v \in V_i$ {
      if $(direction = scanline)$
        $o(v) = \frac{\overrightarrow{v \perp l_i}}{|\overrightarrow{v \perp l_i}|} \cdot \frac{sl}{|sl|}$
      else
        /* direction = section line l_i */
        $o(v) = \frac{\overrightarrow{v \perp sl}}{|\overrightarrow{v \perp sl}|} \cdot \frac{l_i}{|l_i|}$;
      $v = v + SF_i \cdot o(v)$
      }
    }
  }

Algorithm 3: Scanline

4.3.3 The *CartoDraw* Main Algorithm

Having defined the components of the CartoDraw algorithm, we can now describe its main procedure. The algorithm assumes as input a set of polygons $\mathcal{P}$, a scaling vector of the desired statistical parameter $\tilde{X}$, and a set of scanlines SL, which can be determined automatically or manually (see subsection 4.3.4). Output is the modified set of polygons $\mathcal{P}$ which describes the cartogram. The algorithm works as follows (see figure 4). For each scanline, the algorithm applies the scanline transformation and checks the results. If the area difference d_A introduced by the scanline transformation is below a certain threshold ε_A and the shape distortion is below a certain threshold ε_s, then the changes are retained and otherwise discarded. Then, the algorithm proceeds with the next scanline until all scanlines are applied in the same way. At

(a) Contraction (b) Inverse contraction

Figure 4.3: A demonstration of the *scanline* idea in two different situations – In the figures, the green lines indicates the scanline and the grey lines indicates the eleven section lines. The red areas have to be contracted 4.3(a) in contrast to blue areas were the direction of the scanline transformation is reversed to stretch the mesh in each section line step 4.3(b).

this point, the algorithm checks whether in applying all scanlines an improvement of the *area error* has been obtained. If this is the case, the algorithm applies all scanlines again and repeats the entire procedure until no further improvement is reached (area improvement below ε). Since the *area error* improvement must be positive and above the threshold ε in each iteration, the *area error* is monotonously decreasing and termination of the algorithm is guaranteed. Note that in applying an individual scanline, we allow the algorithm to potentially increase the *area error*, to allow escaping local optima. Also, notice that after applying a scanline, all the other ones remaining to be processed must be transformed as well, so that they correspond properly to the transformed map.

```
CartoDraw(P, X̃, SL, ε_NumberOfIteration){
    do{
        AreaError = d_A(X̃, P);
        foreach (sl ∈ SL){
            P̄ = ScanLine (P, X̃, sl);
    /* make persistend iff topology, shape and area tests are passed */
            if (d_T == 0 and d_S(P, P̄) < ε_s  and  AreaError − d_A(X̃, P̄) > ε_A)
                P = P̄;
        }
    }while (
            /* no further area error improvement */
            AreaError−d_A(X̃, P) ≥ ε  and
            /* area error falls below a given threshold */
            d_A(X̃, P) > ε_AreaError  and
            /* loop reaches the maximal number of allowed iterations */
            IterationCount++ < ε_NumberOfIteration);
    return(P);
}
```

Algorithm 4: *CartoDraw*

4.3.4 Automatic versus Interactive Scanline Placement

So far we assumed that the set of scanlines *SL* used by the CartoDraw algorithm are given. In this subsection, we discuss how the scanlines can be obtained. Our implementation allows them to be defined

automatically or interactively. The *automatic generation of scanlines* uses a fixed grid of horizontal and vertical scanlines (see figure 4.4(a)). The grid's resolution can be varied, but within reason this has only a minor influence on the result. Because only those scanlines that do not induce a higher shape and *area error* are applied, generating many useless scanlines causes a potential loss in efficiency, but does not affect the quality of the result.

The best cartograms seem to be obtained when the scanlines are well adapted to the shape of the input polygons and are placed in areas with a high potential for improvement. Automatic placement based on these criteria is difficult to achieve, so we allow the user to *interactively position the scanlines* depending on the result of the previous steps. We can store all the scanlines specified by the human in generating a specific cartogram, and re-apply them later to different data on the same map. This makes it practical to generate a continuous time series of cartograms, without user interaction in each step. In our experience, manual positioning of scanlines is not difficult and can be done quickly. Figure 4.4(b) shows an example of a set of manually placed scanlines. It took about 5 minutes to enter these scanlines. Note that parts of the map that need large changes have many scanlines of varying lengths, while other parts have hardly any scanlines.[2]

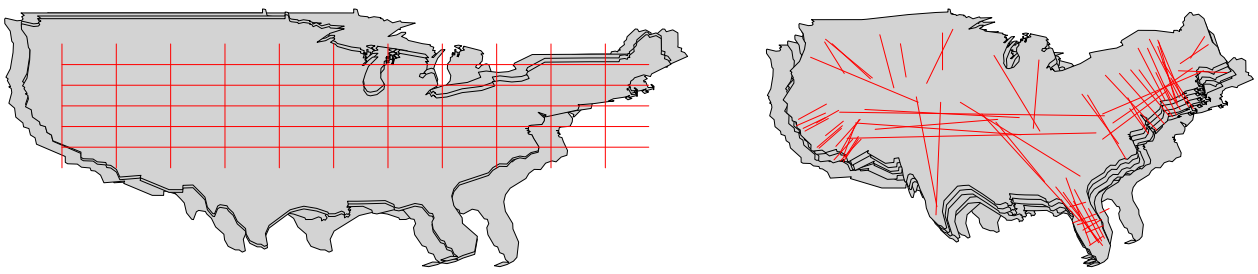
Figure 4.5 shows a few intermediate steps of incrementally applying the automatic scanlines shown in figure 4.4a to the U.S. population cartogram problem. The *area error* is encoded in red for polygons that should be smaller and blue for polygons that should be larger. The algorithm quickly provides nice results in some areas which are well adapted to horizontal and vertical scanlines (e.g., the mid-western states and New England states). In other areas, the improvement that can be reached by global horizontal and vertical scanlines seems limited (e.g. California or New York & Pennsylvania).

Figure 4.6 shows a similar sequence applying the interactive scanlines shown in figure 4.4b to the U.S. population cartogram problem. The local non-orthogonal scanlines specified interactively allow a better adaption of the algorithm to the shape and *area error* of the polygons, and therefore provide better results. The residual *area error* obtained from interactive scanline placement is much lower than that obtained by automatic placement, with both having about the same *shape error*. A detailed comparison of shape and *area error* of the automatic versus interactive scanline placements is provided in section 4.3.5.

4.3.5 Evaluation of the Algorithm

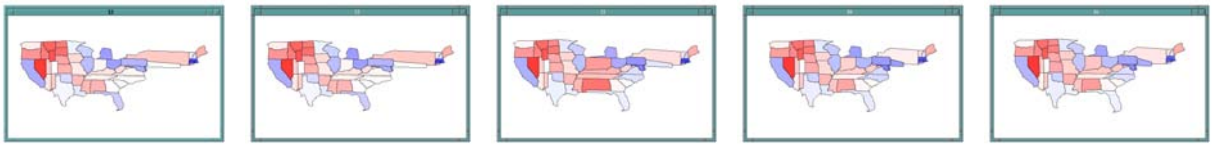
The algorithm as described in the previous section has been implemented in C using the LEDA library [97] and run on a number of different example applications. Unless noted otherwise, the tests were performed on a 1 GHz Pentium computer with 128 Mbytes of main memory. In this section, we report and discuss the results and compare the effectiveness and efficiency of the different approaches. Although our focus is on efficiency, the examples show that our *CartoDraw* algorithm also provides results of very high quality.

[2] A construction video sequence can be accessed here [76]



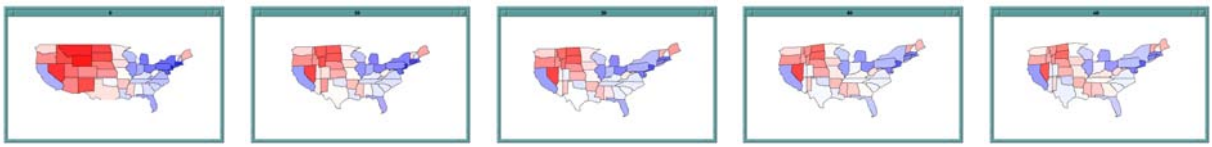
(a) Automatic scanlines (b) Interactive scanlines

Figure 4.4: Automatically versus interactively placed scanlines

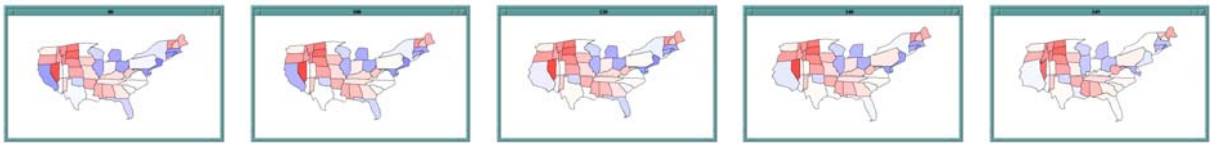


(a) Step 0 (b) Step 12 (c) Step 21 (d) Step 30 (e) Step 36

Figure 4.5: Cartogram construction steps with automatically placed scanlines



(a) Step 0 (b) Step 10 (c) Step 20 (d) Step 40 (e) Step 60



(f) Step 80 (g) Step 100 (h) Step 120 (i) Step 140 (j) Step 149

Figure 4.6: Cartogram construction steps with interactively placed scanlines

For most of the examples, we continue to use a state map of the continental U.S. as a running example.

Time Complexity Let n corresponds to the number of map nodes. In each iteration of the *CartoDraw* main loop and on each *section line* step each node $v \in \mathcal{P}$ has to be accessed on time. Since the number of iteration and section line steps is constant the time complexity is at most $O(n)$.

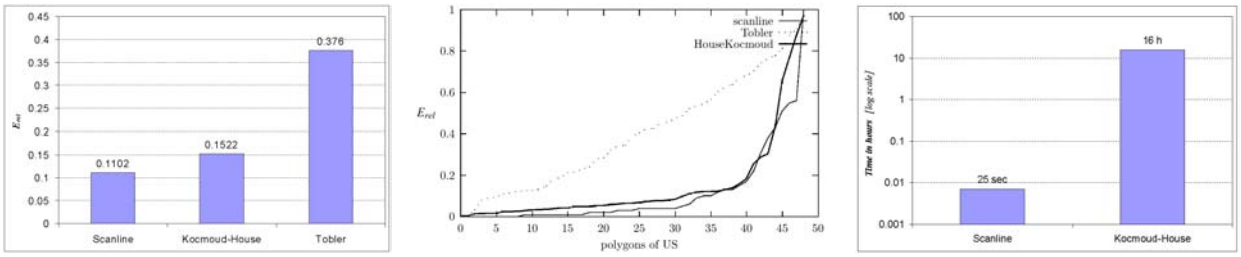
Comparison with Previous Methods Figure 4.7 shows population cartograms generated by our algorithm and by the techniques proposed by Tobler [121] and by Kocmoud and House [90]. A visual comparison shows that our approach offers comparable if not better visual results, with the geography of the United States being clearly perceivable.

(a) Tobler [121] (b) Kocmoud & House [90] (c) Scanline-Based algorithm

Figure 4.7: Comparison of cartogram drawing algorithms

To evaluate the results analytically, figure 4.8(a) shows the total *area error* d_A for all three approaches. Figure 4.8(a) shows that our proposal provides even better results than the complex optimization-based approach by Kocmoud and House [90]. Since the total *area error* is basically an average over the state-wise *area error* , in figure 4.8(b) we show the *area error* state by state, sorted according to the *area error*. Figure 4.8(b) reveals that for most states our approach provides a much better *area error* than the

Tobler cartogram and a slightly better *area error* than the Kocmoud & House cartogram, with very few exceptions.

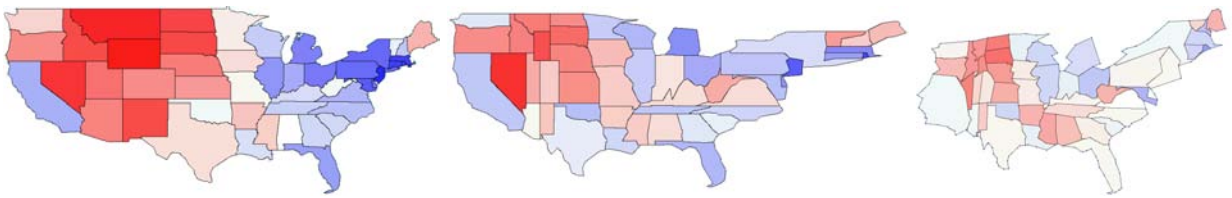


(a) Total area error (b) Area error state-wise sorted (c) Efficiency comparison

Figure 4.8: *Area error* and efficiency comparison (1980 U.S. population cartogram)

In terms of efficiency, our approach is much faster than existing techniques. While previous approaches need hours or even days to compute a solution, our implementation runs in a matter of seconds. Figure 4.8(c) shows that our scanline-based heuristic needs about 25 seconds while the Kocmoud & House approach needs about 16 hours, making our approach about 2000 times faster.[3]

Comparison of the *CartoDraw* Variants One important aspect of the *CartoDraw* algorithm is the specification of the scanlines. As mentioned previously, we allow scanlines to be determined automatically or interactively. In this subsection, we compare these two approaches with respect to effectiveness (quality of the results) and efficiency (time needed to produce the results).

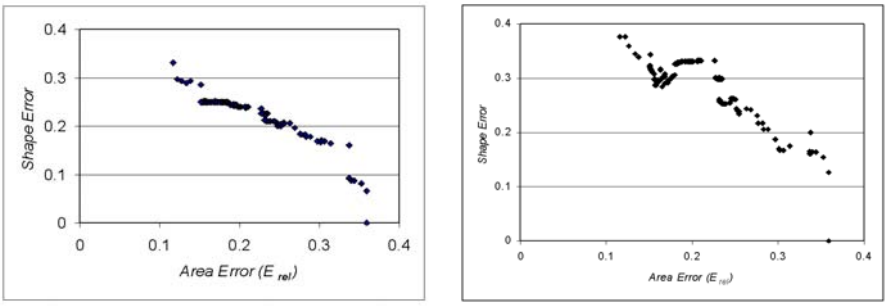


(a) Traditional (b) Automatic scanlines (c) Interactive scanlines

Figure 4.9: The figure display U.S. population cartograms as results of *CartoDraw* with automatically and interactively placed scanlines. The *area error*s d_A in 4.9(a), 4.9(b), and 4.9(c) are 0.36, 0.21, and 0.1, respectively.

Effectiveness In figure 4.9, we show the original U.S. map (figure 4.9(a) with the results of the *CartoDraw* algorithm using automatically generated scanlines (figure 4.9(b)) and interactively generated scanlines (figure 4.9(b)). Both approaches provide high quality cartograms. Figure 4.9 shows that the *area error* d_A is much lower for the interactive scanlines, but shape distortion seems to be higher.

[3]The comparison assumes that both algorithms run on a 120 MHz computer with 32 Mbytes RAM.



(a) State polygons (b) Global polygon

Figure 4.10: *shape error* versus *area error* comparison (interactive scanlines)

To measure the shape distortion, we use the Fourier-based shape similarity function (see subsection 4.2.2). In figure 4.10, we compare the tradeoff between area and *shape error* for each incremental step of the algorithm. Each point in figures 4.10a and b corresponds to one intermediate result of the *CartoDraw* algorithm (with interactive scanlines). The result shows the trade-off between *area error* and shape distortion: In the beginning, there is a large *area error* $d_A = 0.36$. By applying a scanline, the *area error* is improved but the shape becomes more distorted. It is therefore natural that the curve goes from the lower right to the upper left until the area error is small enough or the shape distortion reaches some threshold. A similar behavior can be observed for the global shape. There is however a slight difference: While the *area error* still improves from one step to the next, the distortion global shape in some cases does not get worse.

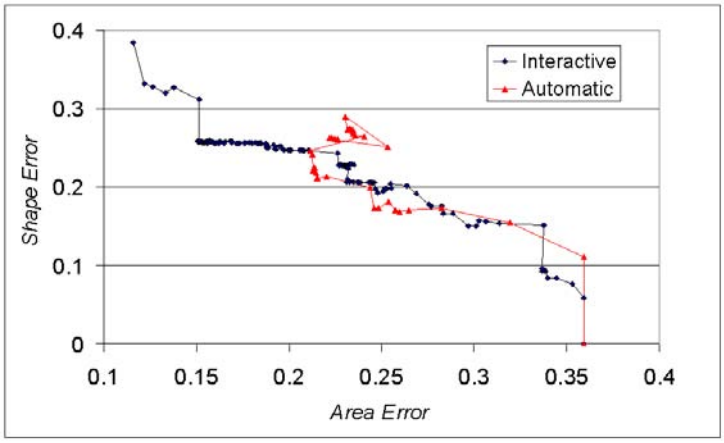
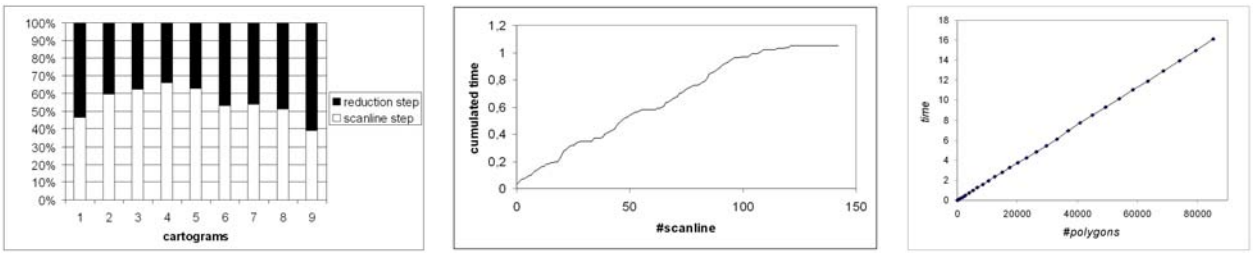


Figure 4.11: Comparison of automatic and interactive scanlines

Comparing the *area error–shape error* tradeoff of interactive versus automatic scanlines reveals some interesting properties of our algorithm (see figure 4.11). In the beginning, both approaches have a similar trend in shape-*area error* tradeoff. At a certain point, however, the automatically generated scanlines lead to a deterioration in *area error* which subsequent scanlines are not able to improve. In case of interactively generated scanlines, the *area error* continues to improve by smaller and smaller increments. Note the jump in *shape error* for an *area error* of about $d_A = 0.15$. At this point we switched the direction from *scanline* to *section line* (see scanline algorithm in subsection 4.3.2), which leads to a continued improvement of the *area error* but a considerable deterioration of the *shape error*.

Efficiency We also performed extensive experiments to evaluate the efficiency of the *CartoDraw* algorithm. The time needed to run the algorithm on the U.S. population data is about 2 seconds. If we change the parameter vector, the time needed for the reduction step of chapter 3.5 versus the scanline execution

varies slightly between 40% and 60%. Figure 4.12(a) shows the percentages needed for the two steps of the algorithm for nine different parameter vectors, namely long-distance telephone call volume data by state for nine time steps during a day. Note that the reduction step can be pre-computed so that it does not have to be re-run each time the algorithm is executed.



(a) Reduction versus scanline step (b) No. of scanlines (U.S. states) (c) No. of polygons

Figure 4.12: Efficiency tests

We also analyzed the effect of changing the length of scanlines. Figure 4.12(b) shows the results for the 144 interactively defined scanlines for the U.S. population data. The time needed to process a scanline depends only on the number of scanline sections which in turn depends only on the length of the scanlines. This means that a steep increase corresponds to long scanlines and a shallow increase corresponds to short scanline. Figure 4.12(b) reveals that shorter scanlines are more likely toward the end of the process and are used for fine tuning some portions of the polygon. Nevertheless, some shorter scanlines are applied regularly in the process as indicated by the irregularities in the curve.

Our final efficiency analysis was aimed at testing the dependency of the *CartoDraw* algorithm on the number of polygons. Since we do not have many different real data sets with a widely varying number of polygons, we generated synthetic data sets, namely checker boards with an increasing number of rectangular polygons. We then used random numbers for initializing the parameter vectors. Figure 4.12(c) shows the results of these tests, revealing, as expected from the time complexity analysis above, a clear linear dependency on the number of polygons. The algorithm needs about 16 seconds for a polygon net consisting of 90,000 polygons. Note, however, that in this case the number of vertices per polygon is very low (four) and a reduction of vertices is not necessary.

4.4 *M-CartoDraw*–Using Medial Axes as Skeleton

CartoDraw can compute cartograms using manual placed scanlines or by running the the scanline method over a regular grid. In this part of the chapter we want to introduce an extension, called *M-CartoDraw*, to compute cartograms fully automatic.

The basic idea of our algorithm is to incrementally reposition the vertices of the polygon mesh using medial axis segments as scanlines. The medial axis or skeleton of a 2D region are the loci of the centers of its maximal inscribed circles. There are many publications and equivalent definitions of the medial axis transformation [11, 13, 103, 20, 97, 24, 20, 28]. For example, is is also defined as the centers of the circumcircles of a Delaunay triangulation. A more intuitive definition is the prairie fire transformation [103]. Imagine that the interior of the polygon is dry grass and the exterior is unburnable wet grass. Suppose a fire is set simultaneously at all points on the polygon's boundary. The fire propagates at uniform speed toward the middle of the figure. At some points, however, different fire fronts meet and extinguish each other. These points are called quench points of the fire; the set of quench points defines the skeleton of the figure.

The medial axes can be gained by generalizing the Voronoi diagrams (see [103, page 179] and [34, page 319]) allowing infinite set of points on the boundary of the polygons. This generalization is illustrated in

figure 4.13.

To illustrate the application of medial axis to cartograms, figure 4.14(a) shows a U.S. map and its medial axis. We will use census population data for the target area vector $\mathcal{V}$. *Area error* is encoded with a bipolar red/blue colormap. Blue regions should be larger, red regions smaller. Color intensity indicates magnitude.

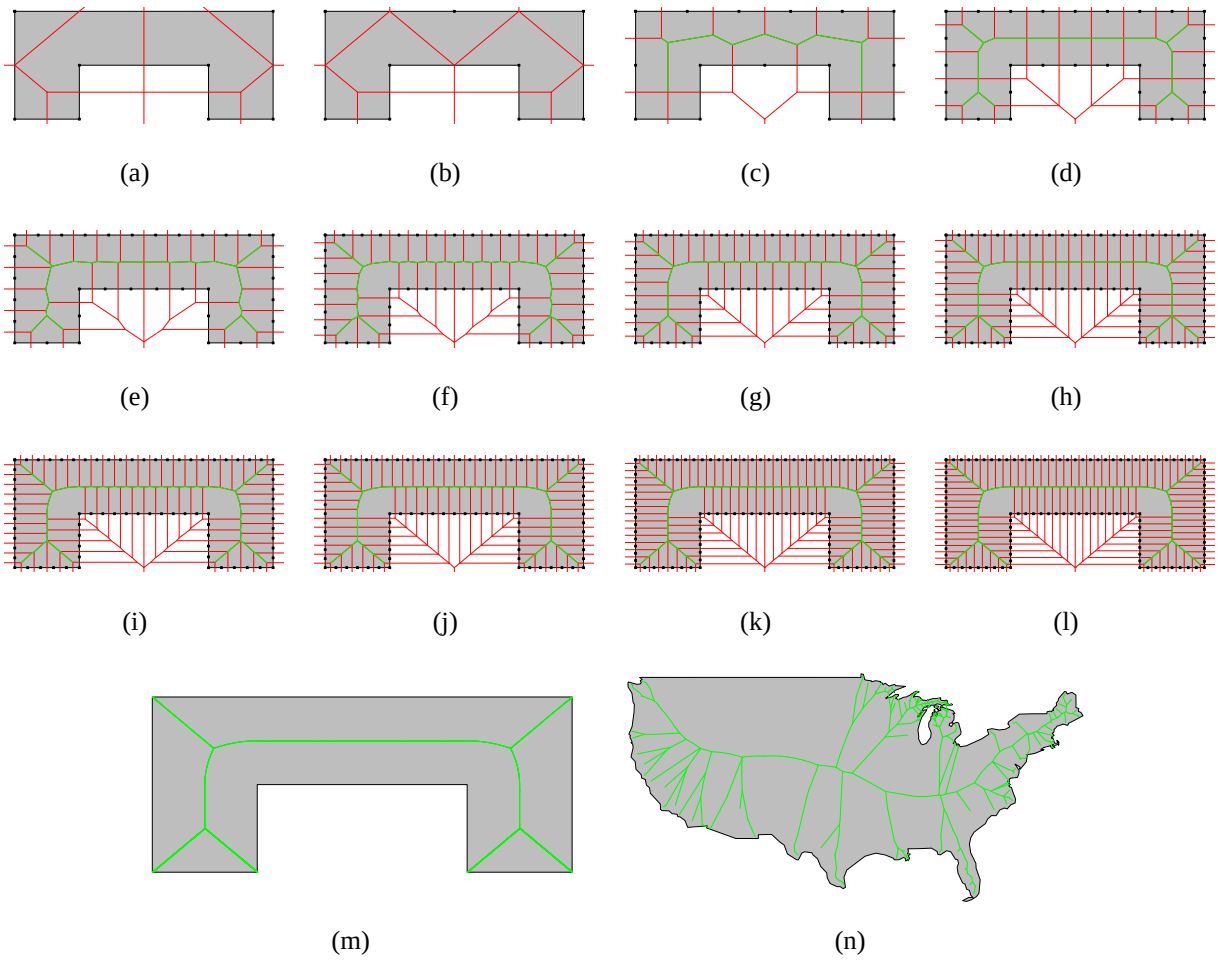
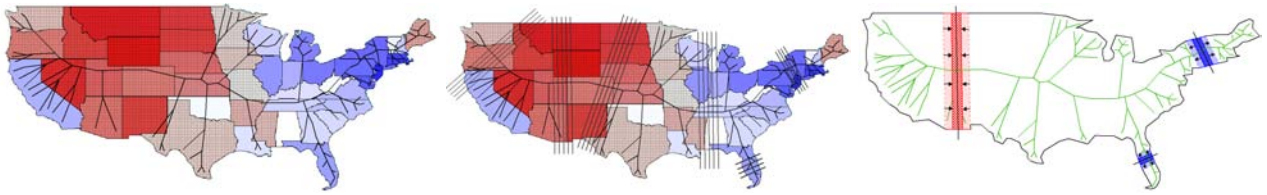


Figure 4.13: The picture displays 12 different levels of the medial axes computation (4.13(a) – 4.13(l)) of a sample polygon. On each figure, we computed the Voronoi diagram (red segments, see [97, 24] for details) of the black points on the border of the polygon. All edges which lie completely inside the bounded region form the medial axes (green segments) of the polygon. Subfigures 4.13(m) and 4.13(n) show the resulting "skeleton" for the sample polygon and the U.S. boundary, respectively.

4.4.1 Basic Idea

Consider a line (called a scanline) drawn inside a polygon. Our algorithm computes line segments (called *section lines*) perpendicular to the scanline at regular intervals. Consider the two edges on the boundary of the polygon intersected by a section line on either side of the scanline. These edges divide the polygon boundary into two chains. If the polygon is to be expanded, the algorithm applies a translation *parallel* to the scanline to each vertex on chains (in opposite directions) to stretch the polygon. If it is being contracted, the direction of translation is reversed.

Our algorithm repeatedly applies medial axis segments as scanlines, thus using the shape of the polygon to make local expansions or contractions. Figure 4.14(c) shows three examples of this process. In the



(a) Map with area error and medial axes (b) Example section lines (c) Stretching and contracting the global polygon

Figure 4.14: The basic idea of the cartogram algorithm

mid-west a section line contracts the global shape while in the north-east and Florida the global shape is stretched. The section lines are inserted in regular intervals on the medial axis which are the "scanlines" of the algorithm. Figure 4.14(b), shows a few section lines at six different map locations.

4.4.2 The *M-CartoDraw* Main Algorithm

Beside the description of the *M-CartoDraw* algorithm in [77], in this thesis we will make use of the previous section.

The processing of a single medial axis segment is one scanline step as described in Algorithm 3. As mentioned earlier, the scanline function determines whether the mesh is to be stretched or contracted, and the amount of adjustment. It is computed as a weighted average of the *area error*s of the polygons cut by the section line, weighted by their scale factors. Note that the algorithm does not calculate new positions of all vertices for each section line. Instead, it aggregates the distortion vectors for each point and applies the aggregate vector after all section lines of a medial axis segment have been considered.

The basic structure of *M-CartoDraw* is presented in Algorithm 5. The main function takes as input the polygon mesh $\mathcal{P}$ and the desired parameter vector $\mathcal{X}$. The main loop iterates until the *area error* falls below a given threshold $\varepsilon_{AreaError}$. In each iteration, the medial axes of the global polygon is determined and assigned to set of scanlines SL. Next, the main loop calls the *CartoDraw* function using only one iteration to stretch and contract the polygon mesh however it depends on the parameter vector.

The function can be extended by a control over the *shape error* of all polygons and the *shape error* of the global polygon. Since *CartoDraw* itself checks the shape error and because to be sparingly with computation time, further "check" are dispensable at that branch of the code.

Furthermore it is possible to call different variants of the *CartoDraw* function.[4] An alternative cartogram transformation could be, that we just stretch and contract the global polygon of the input map and after that we use a bilinear interpolation to transform the inner nodes of the mesh. An advantage of that method is that the transformation is a very smooth especially for a cartogram sequence. This variant of *CartoDraw* has been used for the AT&T call volume series which can be seen in chapter 7 (see also [75, 76]).

Observe that in processing an individual medial axis segment, we allow the algorithm to potentially increase the *area error* to escape local minima. However, in each iteration of the main loop the *area error* decreases monotonically, so termination is guaranteed.

Figure 4.15 shows a few steps in incrementally applying *M-CartoDraw* to a U.S. population cartogram. As before, the blue polygons should be larger, and red ones smaller. The algorithm quickly provides nice results in areas which are well suited to the proposed approach(*e.g.* the Midwest, New England, California, New York, and Pennsylvania).

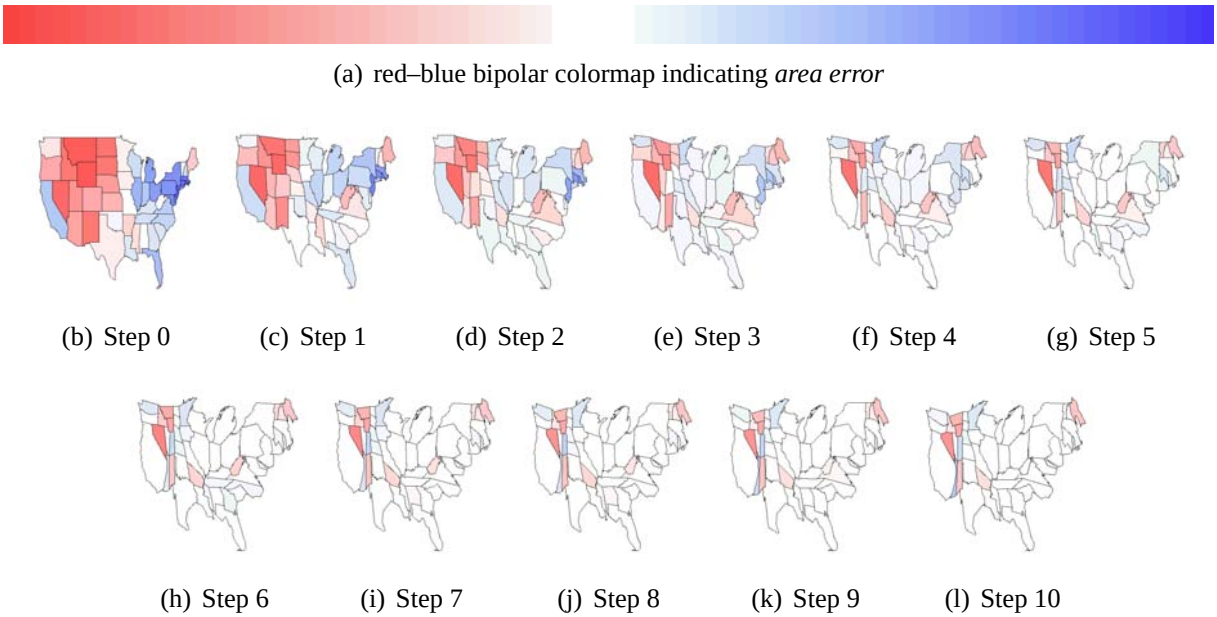
[4] It should be mentioned that we have experimented with at least ten different *CartoDraw* and scanline variants and sub-variants during the last four years. In this thesis we will only present the most two successful versions.

```
M-CartoDraw(P, X̃){
    /* since the area error is larger than a given threshold */
    while(d_A(P, X̃) ≥ ε_AreaError){
        /* compute the global polygon */
        G =computeGlobalPolygon(P);

        /* compute the medial axes of the global polygon */
        SL =computeMedialAxes(G);

        P =CartoDraw(P, X̃, SL, 1);
    }
    return(P);
}
```

Algorithm 5: *M-CartoDraw*



(a) red–blue bipolar colormap indicating *area error*

(b) Step 0 (c) Step 1 (d) Step 2 (e) Step 3 (f) Step 4 (g) Step 5

(h) Step 6 (i) Step 7 (j) Step 8 (k) Step 9 (l) Step 10

Figure 4.15: *M-CartoDraw* construction series; *area error* in step 10 is less than 3%.

4.4.3 Robustness and Stability

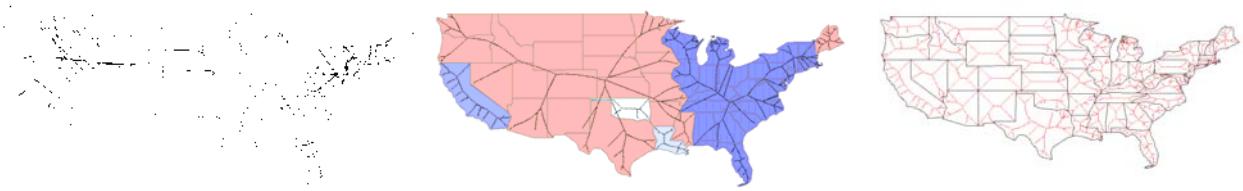
Medial axis transforms are susceptible to problems due to noise and error in the input data set. Although a treatment of this is outside the scope of our work, practical solutions based on pruning strategies and simplification have been studied extensively[45]. Also, as noted by several authors, it is reasonable to generate cartograms from decimated maps, mitigating some of these issues.

4.4.4 Extensions of the Cartogram Algorithm

One problem with the proposed algorithm is that the medial axis of the global polygon may not allow local adjustment of certain regions, though they may have high *area error*. An example of this in the United States map can be found in the upper Midwest (see figure 4.14(a)). Previous experiments with an interactive scanline-based cartogram algorithm suggest that manually placing additional scanlines in such regions can improve the resulting cartograms.

- Clustering Regions
  One approach is to cluster regions that have *area error*s in the same direction, *i.e.* they all need to expand or contract (see figure 4.16(b)). We compute medial axis for each such cluster and then apply Algorithm 3. The cluster regions are processed in order of decreasing aggregate *area error*.

- All Polygons
  The cluster-based approach can be extended further by computing the medial axis of each polygon in the input map. Figure 3 shows the polygons and their scanlines. Again, the scanlines of each polygon are considered in order of decreasing *area error*.



(a) Interactive scanlines (b) Cluster region medial axis (c) All polygon medial axis

Figure 4.16: Extensions of the cartogram algorithm

4.4.5 Evaluation of the Algorithm

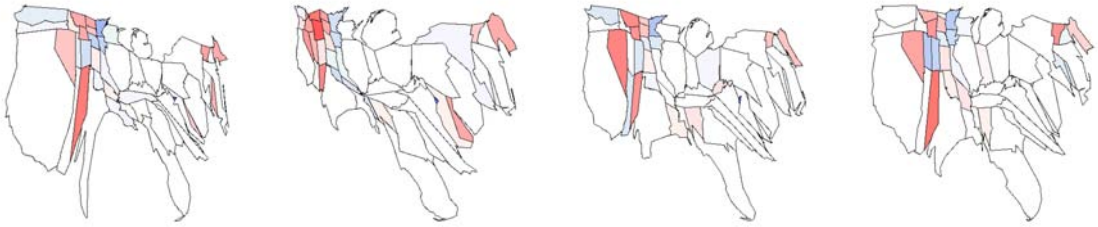
The algorithm described above was implemented in *C++* using the LEDA library [97] and runs on Microsoft Windows and Linux. Tests were performed on a 1,5 GHz Intel Xeon server with 4 GBytes of main memory (although only 15MB were needed) under Linux. In this section, we discuss the results and compare our approach with some alternatives.

On the whole, our method provides cartograms competitive with previous approaches (see figure 3.4), with the geography of the United States being clearly recognizable.

Time Complexity Let m corresponds to the number of global nodes of the mesh $m = |GP(\mathcal{P}|)$. For getting the medial axes[5], *M-CartoDraw* needs to compute the Voronoi-diagram. This can be done in $O(m \log m)$. Since the number of medial axis segments depend linearly on the number of global nodes m of the input set and the number of iteration is constant, the algorithm for generating the cartogram needs at most $O(m \log m + n \cdot m)$ time.

[5] Assuming simple polygons, there are also solutions for getting an approximation of the media axes in linear time [20].

(a) Red–blue colormap used to demonstrate the *area error*

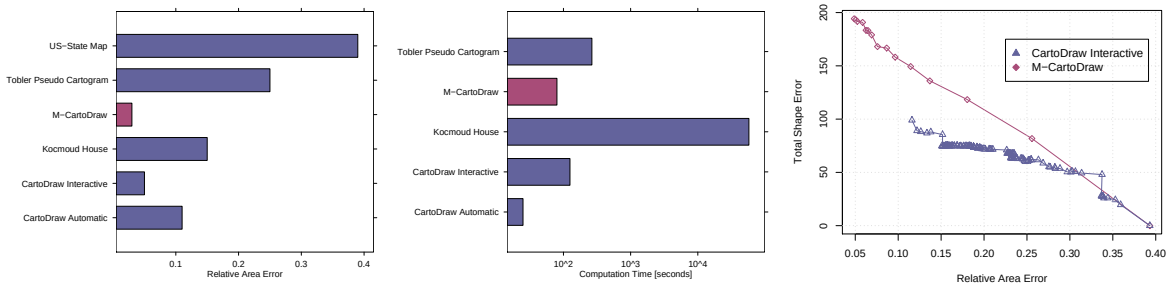


(b) 0:00 am (EST) (c) 6:00am (EST) (d) 12:00pm (EST) (e) 6:00pm (EST)

Figure 4.17: U.S. telephone call volume data over 24 hours. The color of each polygon represents the *area error*. White polygons are distorted with an *area error* close to 0, blue polygons should be made larger and red polygons should be made smaller.

Effectiveness Figure 4.17 shows the output of *M-CartoDraw* with call usage in the U.S. The picture shows cartograms of volume of a telephone service at four different time points (midnight, 6 a.m., noon, 6 p.m. EST) of one day.

All the cartograms provide high quality in the sense that the geography of the U.S. is clearly recognizable, while the *area error* is less than 5% in each. The color of each polygon represents the *area error*. White polygons are perfectly distorted with an *area error* close to 0, blue polygons should be larger, and red should be smaller. The visualizations show interesting patterns of phone service usage that reflect the different time zones of the U.S.



(a) Area error comparison (b) Computation time comparison (c) Shape versus *area error*

Figure 4.18: Effectiveness and efficiency comparison using the U.S. state map

Compared to the *scanline* approach in section 4.3, *M-CartoDraw* yields better results in relation to the area *shape error* trade–off as can be seen in figure 4.18(c).

To measure shape distortion, we employed the Fourier–based method introduced in [75]. In figure 4.18(c), each point corresponds to the intermediate solution found in one *M-CartoDraw* step. At the beginning, the *area error* is larger than 36% for all maps. With increasing number of iterations, the *area error* decreases and the *shape error* grows due to distortions that are introduced. As expected, the curve traces from the lower right corner up to the left corner until the *area error* is small enough, the *area error* difference is less than its threshold, or the shape distortion is larger than a given threshold. In most cases, *shape error* and *area error* have an inverse relationship. The diagram also shows that the final *shape error* depends on the *area error* at the beginning. That is because maps starting with a high *area error* need to be distorted more heavily than those with lower *area error*. The diagram also implies that the slope of the curve corresponding to *M-CartoDraw* is much more constant than that of the *scanline* approach of section 4.3, which can be attributed to extensive human interactions. Figure 4.18(a) shows the total *area error* for *M-CartoDraw* (with 3% *area error*) and [75, 90]. This figure shows that the proposed approach

is preferable to the in section 4.3 described technique and to the hybrid optimization-based approach of [90].

The multi panel plot [21] on figure 4.19 demonstrates the state wise trade off between *shape error* and *area error* over the number of iterations using *M-CartoDraw* for the distortion and the U.S. population data as input. On each panel (bottom to top) we doubled the displayed number of iterations starting with the initial map (bottom) with no *shape error*s. Each point on the figure represents one state. All states which has to be smaller (which means that $\tilde{x_j} - A(\overline{p_j}) \leq 0$) were colored red and blue otherwise. The plot reverifies the theoretic work in chapter 3. It can be seen that it is impossible to eliminate the *area error* without allowing *shape error*. For some states it is even impossible to get the *area error* below a threshold. Furthermore, on an increasing number of iteration the *area error* improvement decrease.

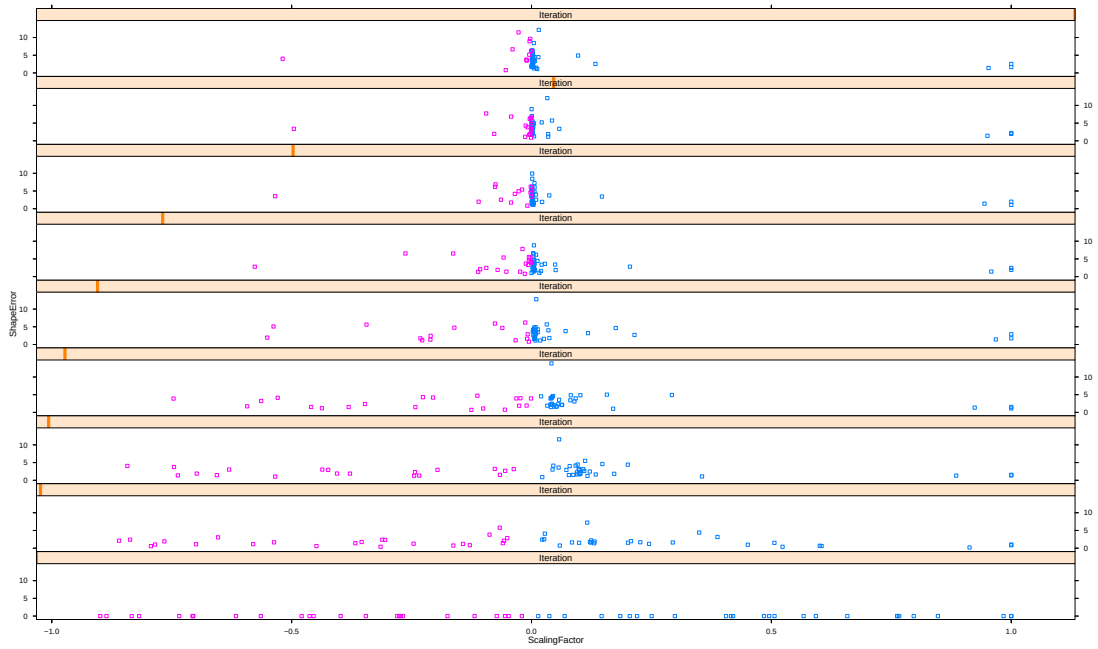


Figure 4.19: *Shape error* versus *area error*–On an ideal plot, on the top panel, all points have been moved in the middle with no *shape error*.

Efficiency We performed experiments to evaluate the efficiency of the proposed algorithm. For this study we did not include the computation time needed to simplify (decimate) the input map, because we treat this an external one-time pre-computation. The main advantage of our approach is its low running time; timings range from six seconds for the U.S. state map to five minutes for the U.S. county map (with about 3000 polygons). Finding the medial axis is about half the total running time. This compares favorably with the prior best known approach [90], which (adjusted for current CPUs) takes about two orders of magnitude longer to compute a cartogram. This is demonstrated in figure 4.18(b), comparing *M-CartoDraw*'s running time with that of [90]. The test assumed that the algorithm runs on a $120MHz$ computer with $32MByte$ RAM. Note that the Y scale is logarithmic.

4.5 Conclusions

In this study we analyzed and discussed the problem of efficient contiguous cartogram drawing, and proposed two optimistic algorithm that outperforms previous techniques by orders of magnitude and provides results that are at least as correct.

The first algorithm is enhance by a medial axis-based techniques which is used for computing a skeleton of the input mesh.

Experiments show that the proposed algorithms offers good results for a variety of applications and scales to a large number of input polygons. For medium sized data sets, the performance is sufficient for an interactive display of network traffic levels in telecom applications.

Although the proposed algorithm is a significant step toward fast, reliable, and effective cartogram generation, there remain several promising directions for further research, including the dependency of the results on the selected scanlines and the improvement of automatic scanline placement.

It would be interesting to study general methods for computing a *morph* (homotopy) with specified boundary properties that optimizes some function of the interior. Our method of using medial axis segments as scanlines can be further generalized (for example, applying vectors in the direction of flow from the medial axis to the boundary).

The proposed algorithm can be enhanced in the 3D space for graph layouts.

5 *RecMap*: An Algorithm for Generating Rectangular Map Approximations

5.1 Introduction

Contiguous cartograms are marvelous and their main advantages is that shape of the map is preserved as much as possible. Nevertheless, as it was proven in chapter 3, there exist maps and parameter values where it is impossible to reduce the *area error*. Practically *CartoDraw* works well for map up to 100 polygons. If we have more than 500 map regions it becomes more and more difficult to reduce the *area error*.

Especially for data analysts it is important to have an accurate presentation of the data. Therefore, we will introduce a new cartogram technique called *RecMap* to compute cartograms with – no *area error* – and we will give a wide variety of constrains to the user to produce high quality visualizations.

Cartographers and geographers used *cartograms* or *value–by–area maps* long before computers were available (see the introduction by Daniel Keim and Stephen North in [77]). As mentioned earlier the basic idea of a cartogram is to distort a map by resizing its regions according to some external geography–related parameter. First hand-made cartograms can be found in [107, pp. 216–217]. Because constructing cartograms manually is a very cost-intensive and time consuming task, researchers oversimplified the shapes of the map. The map of figure 5.1 shows a partitioning of the U.S. into rectangles. Here, the area of a region corresponds to its population in 1958. A detailed description of how to construct rectangular cartograms manually can be found in [107].

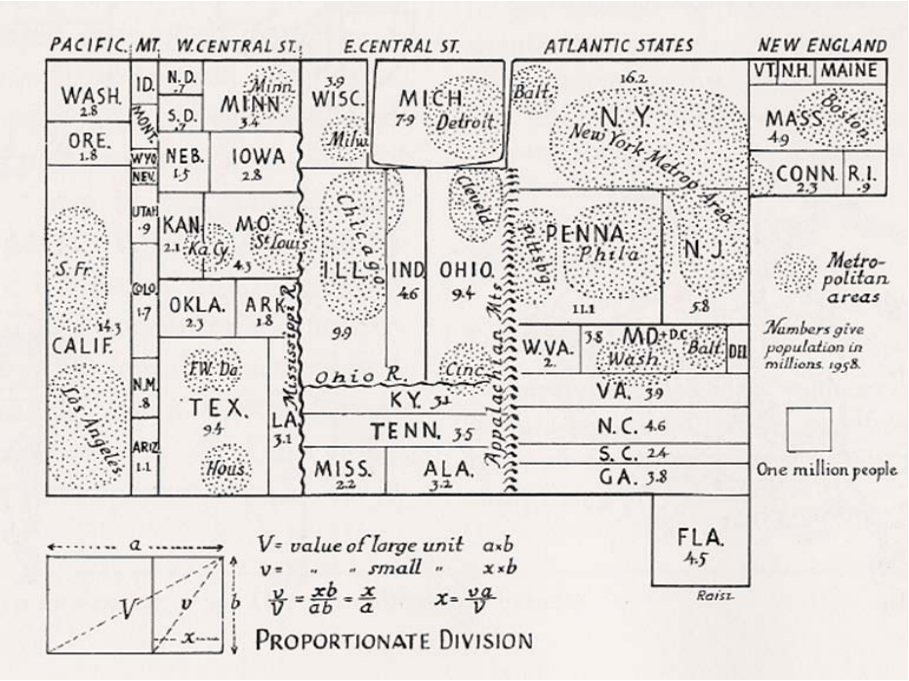


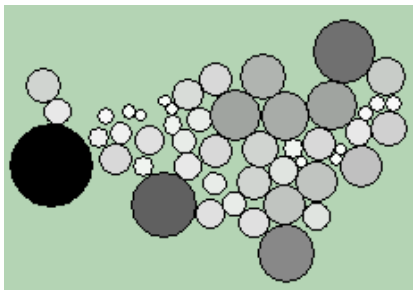
Figure 5.1: Erwin Raisz's hand-made *value–by–area cartogram* shows the population of the U.S. in 1958, (*Erwin Raisz, Principles of Cartography, 1962, McGraw-Hill Book Company, Inc., Fig. 21.4 permission by courtesy of* ©*McGraw-Hill Education*)

The construction of cartograms is a very difficult task because, on the one hand, one has to resize the regions according to their geo–spatial statistical values and, on the other hand, one has to take into account that the (original) shapes of the regions and their neighborhood relationships (*topology*) are preserved as much as possible.

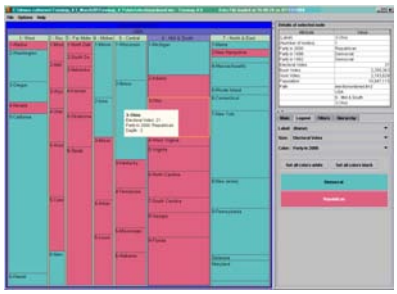
Consequently, the study of automated methods for drawing cartograms is of considerable interest. In the meantime, many automatic visualization techniques have been developed (for an overview see chapter 3). In the following we will give a short overview of related cartogram techniques and space filling visualization techniques.

Circular cartograms [30] (see figure 5.2(a)) ignore the shape of the input polygons completely and represent them as circles. In many cases, the area and topology constraints have to be relaxed, too. The general applicability of this technique is open to question.

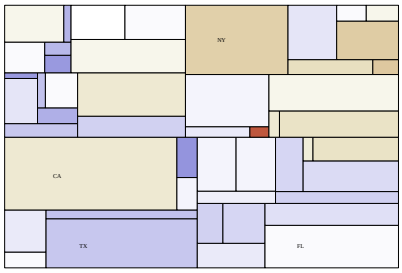
Tree-Maps [62], which are a well–known information visualization technique, are an appropriate method to display data with a given hierarchic order. They divide the display area into rectangles such that the area of each rectangle corresponds to its statistical value. Figure 5.2(b) displays an example where *Tree-Maps*[1] are used for visualizing U.S. census data. To the best of our knowledge there do not exist any automatic procedures which compute the split hierarchy of the map. This work has to be done by the user via interaction.



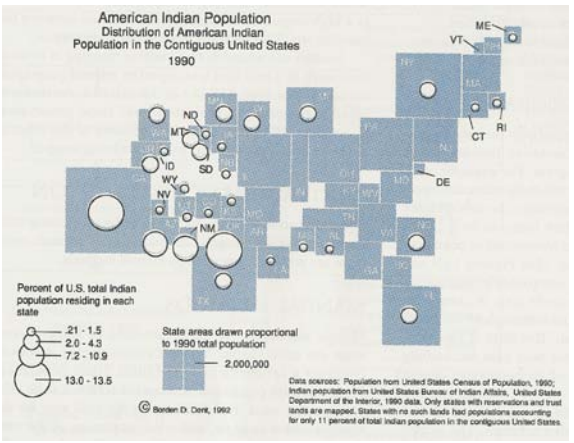
(a) *Circle cartograms* [30, 32]



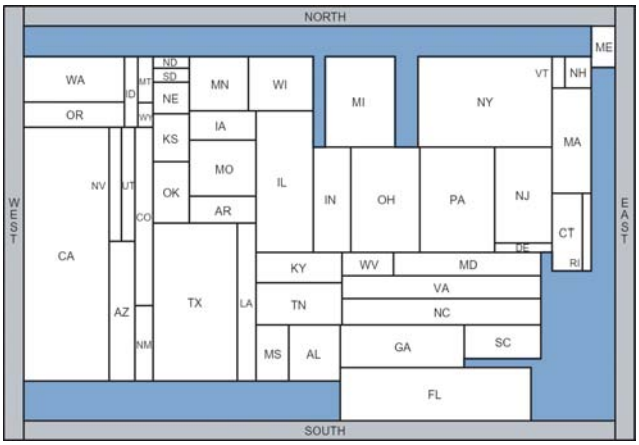
(b) *Tree-Map* [102]



(c) *Quad trees* [108]



(d) Hand drawn rectangular cartogram (*Borden D. Dent, Cartography: Thematic Map Design, 1999, McGraw-Hill Book Company, Inc., Fig. 11.10 permission by courtesy of ©McGraw-Hill Education*) [27].



(e) Automatic generated rectangular cartogram (*permission by courtesy of Bettina Speckmann*) [125, 127].

Figure 5.2: Related work on *RecMap* using the U.S. map and the corresponding population data as input.

Quad trees [42, 108, 24] are well known from the computational geometry and used for storage and retrieval for higher dimensional points. In Figure 5.2(c) the faces in the *Quad tree* corresponds to the

[1] We thank Catherin Plaisant for preparing the *Tree-Map* picture in figure 5.2(b)

statistical value. The computation of the tree structure is linear in the number of nodes. A drawback of that technique is that it does not take into account the regions neighborhoods and their shapes (aspect ratios).

An interesting approach of computing rectangular cartograms has been currently introduced [125, 127] and can be seen in figure 5.2(e). This approach is based on existing VLSI layout algorithms. The basic idea of the algorithm works as follows. In a first step the cartogram procedure creates a *4-connected triangulated plane graph* of its *pseudo dual* input map[2]. The second step computes the rectangular layout by employing a linear time rectangular edge labeling REL algorithm by He and Kant[63]. While the REL algorithm computes an arbitrary layout without taking the desired area values, aspect ratio, and polygon neighborhood into account, the generated layout has to be stretched or contracted perpendicular to the x and y axes according to a given parameter vector in the final step. Therefore Speckmann and van Kreveld implemented a *segment moving heuristic* which is similar to the *CartoDraw* algorithm described in the previous chapters. Since the quality of the REL algorithm result depends on the triangulation of the 4TP graph, several rectangular map partitions are possible. There are some major drawbacks with this technique. First in general the *area error* of the in [125, 127] cited approach can not be eliminated completely which means that there exists a polygon mesh and a parameter vector so that the resulting map has at least one region where the area does not reflect the statistical value. Since our motivation of approximating the regions by rectangles is based on eliminating the *area error*, the techniques is not an improvement in the sense of information visualization. Furthermore it seem that the algorithm induces a large set of possible solutions (more than 4000 for the U.S. map which are more than 40 time more then what *RecMap* produces) which have to be evaluated. The computation time for that can be very high and it should therefore not be underestimated. Figure 5.2(e) shows a population cartogram of the U.S. using a modified variant which provides almost no area error [126, 127].

Most of the techniques which have been presented so far do not take the shape and the topology of the map into account, e.g., [42, 102, 30], or the area error in contiguous cartograms cannot be eliminated completely, e.g., [75, 125].

The idea of this work is to approximate familiar land covering map region by rectangles and to find a partition of the available screen space where the areas of these rectangular regions are proportional to given statistical values. In order to support the understanding of the information represented by a cartogram we try to place the rectangles as close as possible to their original positions and as close as possible to their neighbors. We define two variants of this optimization problem and present two corresponding algorithms called $MP1$ and $MP2$ which generate space filling partitions of the screen space with respect to the given geo–locations. Both algorithms construct cartograms where the area of each rectangle of the cartogram is proportional to its area within the map. The difference between these construction procedures is that the first method does not allow empty space, whereas the second one preserves the shapes of the polygons. Both algorithms runs within a meta heuristic.

The remainder of this chapter is organized as follows: Section 5.2 is devoted to a formal description of the (variants of the) *map partition problem* or *cartogram problem*. In Section 5.3, we present two solution procedures. The efficiency of our new approach is shown in Section 5.4. Applications of this work will be shown in chapter 7.

5.2 Problem Definition

In this section, we give a formulation of the problem of determining a near–optimal cartogram $\overline{\mathcal{P}} = \{\overline{p_1}, \ldots, \overline{p_R}\}$ for a given map $\mathcal{P} = \{p_1, \ldots, p_R\}$ consisting of R *polygons* or *regions* and vector $\tilde{\mathcal{X}} = (\tilde{x_r})_{r=1,\ldots,R}$ of *spatial data values* $\tilde{x_r} \geq 0$ with $\sum_{r=1}^{R} \tilde{x_r} = 1$. For this, we first refer to the constraints

[2] A 4-connected triangulated plane graph, also called 4TP graph, is a graph where each inner face is a triangle, the exterior face is a quadrangle, and the degrees of all inner nodes is at least four [63].

which have to be met during the optimization process. Hereafter, we turn to the single components of the objective function.

5.2.1 Constraints

When determining $\overline{\mathcal{P}}$ we can choose among several possibilities of representing the regions of $\mathcal{P}$. We have decided to use rectangular polygons as in this way the expressiveness of $\mathcal{P}$ is not impaired by insignificant details of the shapes of the polygons of $\mathcal{P}$. As indicated before, we name this type of cartogram *rectangular map*. Hence, we have to meet the following constraints in any case:

- $\mathcal{P}$ is planar,
- each polygon $\overline{p} \in \mathcal{P}$ is a rectangle, and
- each polygon $\overline{p} \in \mathcal{P}$ is neighbor of at least one different polygon $\overline{p}' \in \mathcal{P}$.

A cartogram $\overline{\mathcal{P}}$ obeying these constraints is called *feasible*. The set of feasible cartograms is denoted by $\mathcal{M}$.

5.2.2 Objective Function

The quality of $\overline{\mathcal{P}}$ depends on two aspects: First, we have to evaluate whether the polygons of $\overline{\mathcal{P}}$ can be easily recognized in $\mathcal{P}$. Second, the areas of the polygons of $\mathcal{P}$ have to reflect the geo–spatial data values given by $\mathcal{X}$. In general, these requirements represent conflicting goals. Based on these aspects, we use five criteria in order to evaluate the quality of $\overline{\mathcal{P}}$. These criteria, which correspond to the components of the objective function, are presented in the following.

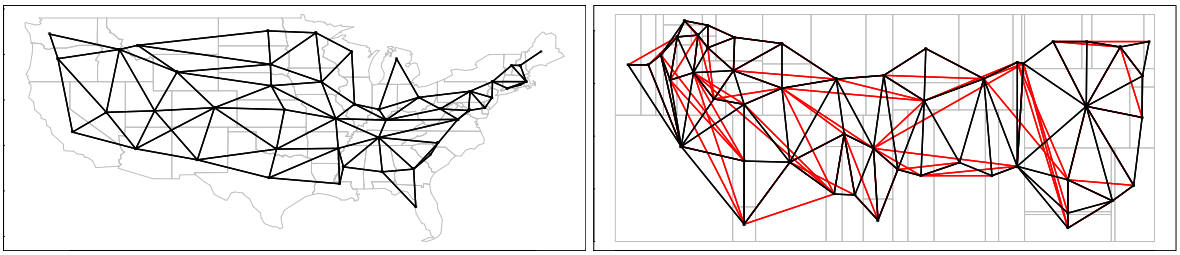


Figure 5.3: The figures show the adjacency graphs of the U.S. map (left) and a corresponding map partition (right). The red colored segments indicate the topology error.

Area

The quality of $\overline{\mathcal{P}}$ w.r.t. the criterion “area” is measured similar to equation 4.7 using equation 4.5 by the *area error* d_A with

$$d_A = d_A(\tilde{\mathcal{X}}, \overline{\mathcal{P}}) \tag{5.1}$$

$$= \sum_{j=1}^{R} \left(\tilde{d}_A(\tilde{x_j}, \overline{p_j}) \cdot \frac{\tilde{x_j}}{\sum_{j=1}^{R} \tilde{x_j}} \right) \tag{5.2}$$

Shape

The *shape error* d_S reflects the average relative deviation of the shape of a polygon $\overline{p_r} \in \overline{\mathcal{P}}$ from that of its corresponding polygon $p_r \in \mathcal{P}$ and is determined as follows:

$$d_S = d_S(\mathcal{P}, \overline{\mathcal{P}}) \tag{5.3}$$

$$= \frac{1}{R} \cdot \sum_{r=1}^{R} \frac{|s(p_r) - s(\overline{p}_r)|}{s(p_r)}. \tag{5.4}$$

The shape $s(\overline{p}_r)$ or $s(p_r)$ of a polygon $\overline{p}_r \in \overline{\mathcal{P}}$ or $p_r \in \mathcal{P}$ is measured by the ratio of its maximum extension in horizontal direction and its maximum extension in vertical direction, respectively.

Topology

The *topology error* d_T is an indicator of the deviation of the neighborhood relationships given by $\overline{\mathcal{P}}$ from those given by $\mathcal{P}$. To obtain d_T, we first have to compute the *adjacency graphs* or *pseudo dual graphs* $\overline{G}_a$ and G_a of $\mathcal{P}$ and $\overline{\mathcal{P}}$, respectively. An adjacency graph reflects the neighborhood relationships between the polygons of a polygon mesh (see [101], p. 267). To obtain that graph, we first introduce a vertex for each polygon of the polygon mesh. Next, for each pair of neighbored polygons, we add an edge between the corresponding vertices.

$$d_T = d_T(\mathcal{P}, \overline{\mathcal{P}}) \tag{5.5}$$

$$= \frac{|\overline{E}_a \backslash E_a| + |E_a \backslash \overline{E}_a|}{|\overline{E}_a \cup E_a|}, \tag{5.6}$$

where $\overline{E}_a$ and E_a denote the set of edges of $\overline{G}_a$ and G_a, respectively. The *topology error* reflects the number of neighborhood relationships being solely contained in one of both polygon meshes and is normalized to the interval $[0,1]$. An example for the calculation of d_T is given by figure 5.3.

Relative Polygon Positions

An important criterion for the recognizability of the polygons in $\overline{\mathcal{P}}$ are their relative positions. But since they are only partially reflected by the adjacency graph we use the (*relative*) *position error* with

$$d_R = d_R(\mathcal{P}, \overline{\mathcal{P}}) \tag{5.7}$$

$$= \frac{2}{R \cdot (R-1)} \cdot \frac{1}{180^\circ} \cdot \sum_{r=1}^{R-1} \sum_{\rho=r+1}^{R} \alpha_{r,\rho}, \tag{5.8}$$

which is normalized to the interval $[0,1]$. $\alpha_{r,\rho} = \arccos(\vec{\overline{u}}_{r,\rho} \cdot \vec{u}_{r,\rho}) / (|\vec{\overline{u}}_{r,\rho}| \cdot |\vec{u}_{r,\rho}|)$ measures the deviation of the relative positions of polygons p_r and p_ρ from those of $\overline{p}_r$ and $\overline{p}_\rho$ with the help of vectors $\vec{\overline{u}}_{r,\rho} = c(\overline{p}_\rho) - c(\overline{p}_r)$ and $\vec{u}_{r,\rho} = c(p_\rho) - c(p_r)$ where $c(\overline{p}_r)$ and $c(p_r)$ stand for the centers of gravity of $\overline{p}_r$ and p_r, respectively.

Empty Space

As we make use of rectangular maps it might happen that $\overline{\mathcal{P}}$ contains "holes" or *empty space* which comprises those areas which are completely surrounded by filled space, i.e. by polygons of $\mathcal{P}$. Consequently, we also measure the quality of $\overline{\mathcal{P}}$ by the *empty space error* $d_E(\overline{\mathcal{P}})$ with

$$d_E = d_E(\overline{\mathcal{P}}) \tag{5.9}$$
$$= \frac{A_t(\overline{\mathcal{P}}) - A_f(\overline{\mathcal{P}})}{A_t(\overline{\mathcal{P}})}, \tag{5.10}$$

which equals the share of empty area in the total area $A_t(\mathcal{P})$ of $\mathcal{P}$. $A_t(\mathcal{P})$ stands for the space being enclosed by the boundary of $\mathcal{P}$. Again, $E(\mathcal{P})$ is normalized to the interval $[0,1]$.

5.2.3 Formulation of the Optimization Problem

To give the user full control over the visualization goals we have developed two variants of the map partition problem each of them focussing on different components of objective function

$$f(\overline{\mathcal{P}}) = (d_A, d_S, d_T, d_R, d_E)^T \tag{5.11}$$

.

Variant 1 (*MP*1) Since one of the most important aspects w.r.t. the expressiveness of cartograms is that spatial data is represented by area, we require that d_A equals zero. In order to use the full screen space, we demand that d_E equals zero, too. Hence, using the constraints and the components of f which have been introduced above, we can state the first variant of the map partition problem being considered in this paper as the following *vector minimum problem*:

$$\text{Min.} \quad f(\overline{\mathcal{P}}) \tag{5.12}$$
$$\text{s.t.} \quad \overline{\mathcal{P}} \in \mathcal{M},\ d_A = 0,\ \text{and}\ \ d_E = 0. \tag{5.13}$$

Variant 2 (*MP*2) Like for (*MP*1), we demand that no area error occurs. Second, in order to take the recognizability of the polygons into account, we do not allow any shape error. Consequently, we obtain the following optimization problem:

$$\text{Min.} \quad f(\overline{\mathcal{P}}) \tag{5.14}$$
$$\text{s.t.} \quad \overline{\mathcal{P}} \in \mathcal{M},\ d_A = 0,\ \text{and}\ \ d_S = 0. \tag{5.15}$$

It is likely that (*MP*1) and (*MP*2) represent $\mathcal{NP}$–hard optimization problems. Recently a $\mathcal{NP}$–hard proof appeared for a rectangular cartogram problem variant [10].

5.3 The *RecMap* Algorithm

In the following, we are going to present heuristic solution procedures for both variants of the map partition problem. First, we refer to a heuristic for (*MP*1). Hereafter, we present a method which computes a near–optimal solution for (*MP*2).

To obtain cartograms of high quality, we repeat the construction of cartograms using a *genetic algorithm* (see [51], 2000) which guides the optimization process. In each iteration of this *meta heuristic*, a set or *generation* of cartograms or *individuals* is generated. An individual is characterized by three aspects: the *genotype*, the *construction algorithm*, and the *phenotype*. The genotype stands for the information

needed to generate the corresponding phenotype using a certain construction algorithm. In our context, the genotype corresponds to an array of nonnegative integers and the phenotype to a (feasible) cartogram.

The individuals of a generation are evaluated by means of a weighted objective function $\widehat{f}$ which is derived from f. Then, we *select* a predefined number of best individuals and determine the next generation out of their genotypes by applying *replication* and *mutation*. This process is repeated until a predefined number of generations has been generated or a given amount of time has elapsed. The best cartogram which has been found so far is returned.

***RecMap*_Meta-Heuristic()**

/* **STEP 1** (*initialization step*) */
Create I_0; $I^\star = (-1,\ldots,-1)$; $\widehat{F}^\star := \infty$
/* **STEP 2** (*main step*) */
FOR $\nu := 0$ **TO** n **DO**
  FOR $I \in I_\nu$ **DO**
    /* compute a candidate cartogram */
    Determine_$MP1$ $\mathcal{P}(I)$
    IF $\widehat{f}(\mathcal{P}(I)) < \widehat{f}^\star$ **THEN**
      $I^\star := I$; $\widehat{f}^\star := \widehat{f}(\mathcal{P}(I))$
    IF $\nu < n$ **THEN**
      Create $I_{\nu+1}$
RETURN $I^\star$

Algorithm 6: Genetic algorithm

The weights w_s, w_e, w_t, w_r of $\widehat{f}$ can be set by the user according to her or his visualization goals. In this way, the user gains control over the visualization process and result. The effect of different weights on the resulting cartograms is demonstrated in figure 5.4 w.r.t. our heuristic for ($MP2$). For example, figure 5.4(a) shows the cartogram which is obtained if the weight for d_T is set to one and the other weights are set to zero. (Figures 5.4(b) and 5.4(c) have to be interpreted in an analogous manner.) The cartogram of figure 5.4(d) is obtained if all weights are set to one.

5.3.1 Variant 1

Basic Idea

In order to achieve $d_E = 0$, our heuristic is based on the procedure of [79] which — starting with a single rectangle — computes a sequence of *partial cartograms* $\widetilde{\mathcal{P}}$ by adding a rectangle in each step in such a way that no empty space can occur. In the following, we first give a detailed description of this basic procedure which serves as the construction algorithm within our genetic algorithm.

Initialization Step

In the *initialization step*, we first draw the initial partial cartogram which consists of a rectangle, denoted by p_1, with an horizontal extension of

$$d_x = \max_{r=1,\ldots,R} \max_{i=1,\ldots,\bar{n}_r} \bar{x}_i^r - \min_{r=1,\ldots,R} \min_{i=1,\ldots,\bar{n}_r} \bar{x}_i^r \tag{5.16}$$

(a) $w_{\{e,t,r\}} = (0,1,0)$ (b) $w_{\{e,t,r\}} = (0,0,1)$

(c) $w_{\{e,t,r\}} = (1,0,0)$ (d) $w_{\{e,t,r\}} = (0.3,0.3,1)$

Figure 5.4: Cartograms resulting from different weights for the components of $\widehat{f}$

and a vertical extension of

$$d_y = \max_{r=1,\ldots,R} \max_{i=1,\ldots,\overline{n}_r} \overline{y}_i^r - \min_{r=1,\ldots,R} \min_{i=1,\ldots,\overline{n}_r} \overline{y}_i^r. \tag{5.17}$$

($\overline{n}_r$ stands for the number of nodes of $\overline{p}_r$ and $(\overline{x}_i^r, \overline{y}_i^r)$ for the position of the ith vertex of $\overline{p}_r$ in clockwise order.) Second, we compute the center of gravity $c(\overline{p}_r)$ for each polygon $p \in \mathcal{P}$ and plot this point into the starting rectangle. In the following, the polygons of $\mathcal{P}$ are represented by their centers of gravity.

Main Step

In the *main step*, we perform a sequence of so–called *splits*. Each split refers to a rectangular polygon $p \in \widetilde{\mathcal{P}}$, which contains at least two centers of gravity, and divides it into two new rectangular polygons, each of them containing at least one center of gravity. In this way, we construct a sequence of partial cartograms $\widetilde{\mathcal{P}}$ with no empty space error. Each $p \in \widetilde{\mathcal{P}}$ represents the aggregation of those polygons of $\mathcal{P}$ which correspond to the centers of gravity $c(p_r)$ being contained in $\overline{p}$ (i.e. $c(p_r) \in \overline{p}$).

The main step ends, when each rectangle contains exactly one center of gravity and hence no further split can be done. Consequently, after $R-1$ splits we obtain a partial cartogram $\widetilde{\mathcal{P}}$ with R polygons. This final (partial) cartogram corresponds to a (complete) cartogram: $\mathcal{P} = \widetilde{\mathcal{P}}$. The polygons of $\overline{\mathcal{P}}$ have to be re–numbered because, as indicated before, a rectangle $\overline{p} \in \overline{\mathcal{P}}$ represents that polygon $p_r \in \mathcal{P}$ which corresponds to the single center of gravity being included in $\overline{p}$. Therefore, $\overline{p}$ gets the index r.

We differ between two types of splits: a *horizontal split* and a *vertical split*. A horizontal (vertical) split introduces a horizontal (vertical) *splitting line* into the rectangle $p \in \widetilde{\mathcal{P}}$ to be split, which results in two new polygons p' and p'' with p' being below (left) of p''. Splitting is done in such a way that, after each split, we have for the current partial cartogram $\widetilde{\mathcal{P}}$

$$\frac{A(p_r)}{\sum_{\rho=1}^{|\widetilde{\mathcal{P}}|} A(p_\rho)} = \sum_{c(p_r)\in p} \tilde{x_r} \qquad (r = 1,\ldots,|\widetilde{\mathcal{P}}|), \tag{5.18}$$

i.e. the area of $p_r \in \widetilde{\mathcal{P}}$ is proportional to the sum $\sum_{c(p_r)\in p} \tilde{x_r}$ of the spatial data values of the polygons $p_r \in \mathcal{P}$ being associated with $c(p_r) \in p$ $(r = 1,\ldots,|\widetilde{\mathcal{P}}|)$. We try to split a polygon $p \in \widetilde{\mathcal{P}}$ as equally as possible: If we do a horizontal (vertical) split, we scan the points $c(p_r) \in p$ from bottom to top (left to right), and add them to p' until we have

$$\sum_{c(p_r)\in p'} \tilde{x_r} \geq \frac{1}{2} \cdot \sum_{c(p_r)\in p} \tilde{x_r}. \tag{5.19}$$

Those centers of gravity of p which have not been added to p' are added to p''. If p contains two centers of gravity, we stop after having scanned the first. Provided that we perform a horizontal or vertical split of $p_r \in \widetilde{\mathcal{P}}$, the splitting line is placed into p_r such that the height or the breadth of p' equals

$$\frac{\sum_{c(p_r)\in p'} \tilde{x_r}}{\sum_{c(p_r)\in p} \tilde{x_r}} \cdot (y_3^r - y_1^r) \quad \text{or} \quad \frac{\sum_{c(p_r)\in p'} \tilde{x_r}}{\sum_{c(p_r)\in p} \tilde{x_r}} \cdot (x_3^r - x_1^r), \tag{5.20}$$

respectively[3].

The *RecMap* –Algorithm for Variant 1

A major drawback of the procedure described in the previous section is its rigidity. This means, that the polygons resulting from a horizontal split have to be split vertically in any case and vice versa. But in this way, no special attention is paid to the shapes of the polygons and the neighborhood relationships between them. For example, if the majority of the polygons $p \in \mathcal{P}$ possesses a longish shape (i.e. $s(p) < 1$) the procedure might lead to seriously deformed cartograms, i.e. cartograms $\overline{\mathcal{P}}$ with high values for d_S and d_T. In such a case, it would be indicated to prefer vertical splits.

This drawback can be avoided by using *split sequences*. For example, let a cartogram $\overline{\mathcal{P}}$ be obtained by performing a horizontal split and two vertical splits afterwards. If we associate a horizontal split with 0 and a vertical split with 1, we get the split sequence $(0,1,1)$. This split sequence can be conceived as the genotype of $\overline{\mathcal{P}}$. In general, the genotype of a cartogram is a vector $(I_\lambda)_{\lambda=1,\ldots,R-1}$ of (binary) values $I_\lambda \in \{0,1\}$.

To use the construction algorithm described above w.r.t. a given split sequence $(I_\lambda)_{\lambda=1,\ldots,R-1}$, it has to be adapted accordingly. For this, we introduce variable λ which stands for the split position which is currently considered. I_λ represents the splitting type to be chosen for the λth split. At the end of the algorithm, λ equals $R-1$. The adapted construction algorithm is given by Algorithm 7.

The split sequence can be also represented as a tree (see figure 5.5(a)). Each node of the tree represents one polygon. The leaf-nodes represents the final cartogram and all other nodes represents one polygon during the construction process. In this example the split type is encoded as red/black color. The node

[3]Please note, that the notation of a spacial data value $\tilde{x_r}$ differs from the (x_r, y_r) location of the center of gravity of a polygon $p_r \in \mathcal{P}$.

Determine $MP1$ **(**$\mathcal{P}(I)$**)**

/* **STEP 1** (*initialization step*) */
$\widetilde{P} := \{p_1\}$; $\mathcal{S} := \{p_1\}$; $\lambda := 1$
/* **STEP 2** (*main step*) */
WHILE $\mathcal{S} \neq \emptyset$ **DO**
  Remove p from $\mathcal{S}$
  IF $|\{\overline{p}_r \in \overline{\mathcal{P}} | c(\overline{p}_r) \in p\}| > 1$ **THEN**
    IF $I_\lambda = 0$ **THEN**
      Split p horizontally into p' and p''
    ELSE
      Split p vertically into p' and p''
    $\mathcal{S} := \mathcal{S} \cup \{p''\}$; $\mathcal{S} := \mathcal{S} \cup \{p'\}$
    $\widetilde{\mathcal{P}} := \widetilde{\mathcal{P}} \backslash \{p\}$; $\widetilde{\mathcal{P}} := \widetilde{\mathcal{P}} \cup \{p', p''\}$
    $\lambda := \lambda + 1$
RETURN $\widetilde{\mathcal{P}}$

Algorithm 7: The *RecMap* $MP1$ construction procedure

with the number 1 is the root of the tree. While red nodes force vertical splits, black nodes force horizontal splits. The split sequence for our demonstration is defined as follows:

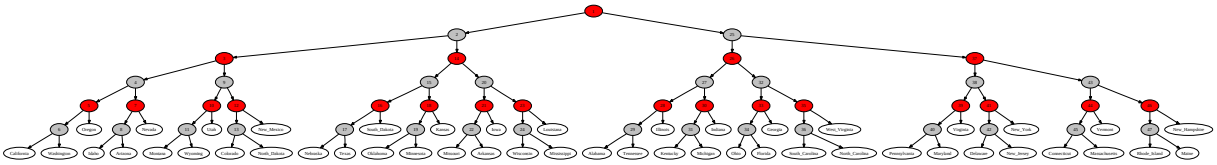
$$
\begin{aligned}
(I_\lambda)_{\lambda=1,\ldots,R} \quad = \quad & (1,2,3,4,5,6,7,8,9,10, && (5.21)\\
& 11,12,13,14,15,16,17,18,19,20, && (5.22)\\
& 21,22,23,24,25,26,27,28,29,30, && (5.23)\\
& 31,32,33,34,35,36,37,38,39,40, && (5.24)\\
& 41,42,43,44,45,46,47) && (5.25)
\end{aligned}
$$

Please note, that the split sequence is usually not as regular as in the example above. The applied split sequence on $MP1$ can be seen in figure 5.5(b) using the U.S. state population data set.

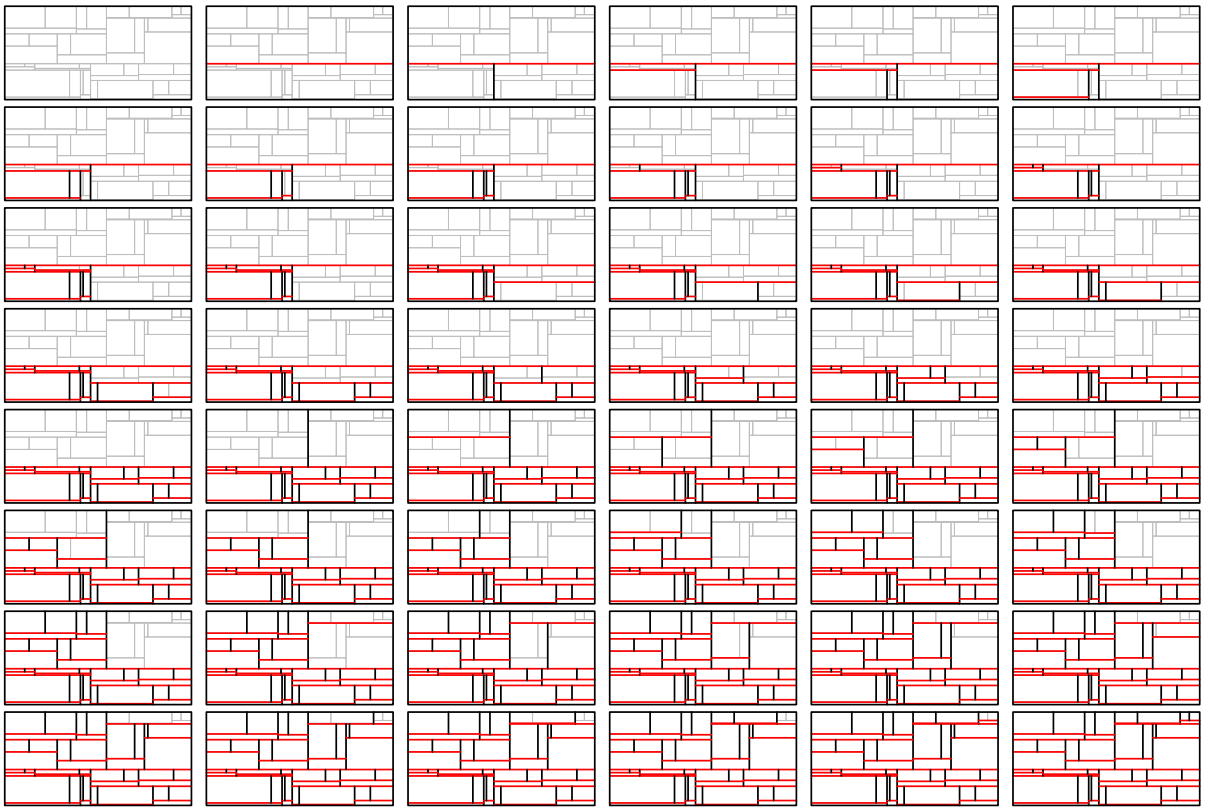
5.3.2 Variant 2

Basic Idea

In literature, we find an optimization problem in the context of *inner–plant layout planning* (see [101], p. 255 et seqq.) which shows certain similarities to ($MP2$). This *layout problem* can be roughly stated as follows: We are given a rectangular site, a set of machines (being described by their ground plans), and the amounts of material which have to be transported between them. The objective is to find a *layout*, i.e. a plan with the positions of the machines on the site, such that the transportation costs are minimized, i.e. that machines with a high transportation intensity in between are located as close as possible. The map partition problem considered here is similar to the layout problem in so far as there are also given a set of rectangular areas (i.e. polygons) which have to be placed such that the distances between them are taken into consideration.



(a) The figure displays the split tree of the figure 5.5(b). Red nodes indicate horizontal and black nodes indicate vertical splits. The tree has been drawn using the `dot` program from *Graphviz* [49].



(b) All 47 *MP*1 splits for a alternating *x*–*y* splitting sequence.

Figure 5.5: A demonstration of *RecMap MP*1' construction using U.S. census 2000 population data on state level and a regular split sequence.

The construction algorithm of our heuristic for (*MP*2) relies on two ideas of the procedure of [46] which is derived from a graph theoretical model of the layout problem. First, in the *initialization step*, we choose a specific polygon, called the *core polygon* p^c, to be the center of the *layout* or *cartogram*. Second, in the *main step*, we construct a sequence of *partial layouts* or *partial cartograms* $\widetilde{\mathcal{P}}$, i.e. starting with p^c, the remaining $R-1$ polygons are placed around it one after the other until we have found a (complete) cartogram $\overline{\mathcal{P}} = \widetilde{\mathcal{P}}$.

Initialization Step

As the *area error* and the *shape error* ought to be zero, the breadth $b(p_r)$ and height $h(p_r)$ of each polygon $p_r \in \mathcal{P}$ are given by

$$b(p_r) = \sqrt{\tilde{x}_r \cdot \sum_{r=1}^{R} a(p_r) \cdot s(p_r)} \text{ and} \tag{5.26}$$

$$h(p_r) = \sqrt{\frac{\tilde{x}_r \cdot \sum_{r=1}^{R} a(p_r)}{s(p_r)}}, \tag{5.27}$$

respectively. The core polygon p^c is determined with the help of an extension G_{a_x} of G_a which is obtained by introducing an additional node $R+1$ for the outer region of $\mathcal{P}$. p^c corresponds to a polygon $\overline{p}_r$ which has the maximum distance $d_{r,R+1}$ from the outer region p_{R+1} where the distance between two polygons is measured by the minimum number of edges between their corresponding nodes in G_{a_x}.

Main Step

As indicated before, the main step consists of $R-1$ partial steps. In each of these steps, we choose an index r among the set of indices of those polygons which have not yet been created and added to $\widetilde{\mathcal{P}}$. Let Q denote the set of indices of the polygons $p_r \in \widetilde{\mathcal{P}}$. Then, the index of the newly created polygon has to be taken from $\{1,\ldots,R\}\backslash Q$. Since the maintenance of the neighborhood relationships of $\mathcal{P}$ is of high importance w.r.t. the recognizability of the polygons of $\mathcal{P}$, we have a look at G_a in order to determine the index r' of the polygon to be added next. We demand that the corresponding polygon $p_{r'} \in \mathcal{P}$ is a neighbor of at least one of those polygons $p_r \in \mathcal{P}$ the indices of which are contained in Q because otherwise, the adjacency graph of the resulting partial cartogram $\widetilde{\mathcal{P}}$ would not be connected and we could no longer guarantee $\overline{\mathcal{P}} \in \mathcal{M}$ for the final cartogram $\overline{\mathcal{P}}$. Let $\mathcal{N}(p_r)$ denote the set of neighbors of p_r in $\mathcal{P}$, i.e. $\mathcal{N}(p_r) = \{p_\rho | (r,\rho) \in E_a\}$. If there are several indices which could be selected, we choose the lowest one. Thus, we set

$$r' \quad = \quad \min\{r \in \{1,\ldots,R\}\backslash Q | \overline{p}_r \in \bigcup\nolimits_{\rho\in Q} \mathcal{N}(p_\rho)\}. \tag{5.28}$$

After the determination of r' we have to decide where to place the corresponding polygon $p_{r'}$. In general, there exists an infinite number of possible positions for $p_{r'}$. In order to keep the computational time low, we have to restrict ourselves to a finite subset.

Pretests have revealed that the following procedure is favorable: We scan the edges e of the boundary $\mathcal{E}$ of $\widetilde{\mathcal{P}}$ and determine a set Π_e of possible positions for $p_{r'}$ w.r.t. e. For example, we add the end points and the middle point of e to Π_e as possible positions of the lower left corner of $p_{r'}$. To keep the number of possible positions low and to exclude infeasible positions, each position $(x,y) \in \Pi_e$ is checked within a multi–stage test. For instance, we remove those positions from Π_e which cause a violation of the planarity or which could lead to unacceptable high values of d_T or d_R.

Subsequent to the determination of the sets Π_e we select the best position $(x^\star, y^\star)$ and create $p_{r'}$ at that position. $(x^\star, y^\star)$ is found as follows: Let Π denote the set of all feasible positions which have been found so far. For each position $(x,y) \in \Pi$, we temporarily extend $\widetilde{\mathcal{P}}$ by adding the newly created polygon $p_{r'}$ at (x,y) and compute a weighted sum $\widehat{f}(\widetilde{\mathcal{P}})$ of the values of the components of f. The position associated with the minimum value of $\widehat{f}$ equals $(x^\star, y^\star)$.

The *RecMap* –Algorithm for Variant 2

As the order in which the polygons are added to $\widetilde{\mathcal{P}}$ is of high importance we have to encode this information by the genotype of $\mathcal{P}$ which equals a vector $(I_\lambda)_{\lambda=1,\ldots,R}$ of values $I_\lambda \in \{1,\ldots,R\}$ with $I_\lambda \neq I_{\lambda'}$ for $\lambda \neq \lambda'$ $(\lambda, \lambda' \in \{1,\ldots,R\})$. For the use within our genetic algorithm–based heuristic, we have to adapt the choice of r' accordingly. This means, that among the set of indices of those polygons which are neighbors of polygons p_r with $r \in Q$, we select that index r' which is the first of them in I. Hence, (5.28) has to be modified as follows:

$$\begin{aligned} r' \quad = \quad & \min\{\lambda \in \{1,\ldots,R\} | I_\lambda \in \{1,\ldots,R\}\backslash Q \text{ and} \\ & \overline{p}_{I_\lambda} \in \bigcup_{\rho\in Q} \mathcal{N}(\overline{p}_\rho)\}. \end{aligned} \tag{5.29}$$

The construction algorithm is given by algorithm 8. An example sequence for generating an U.S. state population cartogram can be seen figure 5.6.

Determine *MP*2 **(**$\mathcal{P}(I)$**)**

/* **STEP 1** (*initialization step*) */
Create $\overline{G}_{a_x}$
$r' := \min\{r \in \{1,\ldots,R\} | d_{r,R+1} = \max_{\rho=1,\ldots,R} d_{\rho,R+1}\}$
Create $p_{r'}$ at $(0,0)$; $\widetilde{\mathcal{P}} := \{p_{r'}\}$; $Q := \{r'\}$; $p^c := p_{r'}$
/* **STEP 2** (*main step*) */
WHILE $Q \neq \{1,\ldots,R\}$ **DO**
    $r' := \min\{r \in \{1,\ldots,R\} \setminus Q | \overline{p}_r \in \bigcup_{\rho \in Q} \mathcal{N}(\overline{p}_\rho)\}$
    $\Pi := \emptyset$; $Q := Q \cup \{r'\}$
    FOR $e \in \mathcal{E}$ **DO**
        Determine Π_e; $\Pi := \Pi \cup \Pi_e$
    $\widehat{f}^\star := \infty$
    FOR $(x,y) \in \Pi$ **DO**
        Create $p_{r'}$ at (x,y); $\widetilde{\mathcal{P}} := \widetilde{\mathcal{P}} \cup \{p_{r'}\}$
        IF $\widehat{f}(\widetilde{\mathcal{P}}) < \widehat{f}^\star$ **THEN**
            $(x^\star,y^\star) := (x,y)$; $\widehat{f}^\star := \widehat{f}(\widetilde{\mathcal{P}})$
        $\widetilde{\mathcal{P}} := \widetilde{\mathcal{P}} \setminus \{p_{r'}\}$
    Create $p_{r'}$ at $(x^\star,y^\star)$; $\widetilde{\mathcal{P}} := \widetilde{\mathcal{P}} \cup \{p_{r'}\}$
RETURN $\widetilde{\mathcal{P}}$

Algorithm 8: The *RecMap MP*2 construction procedure

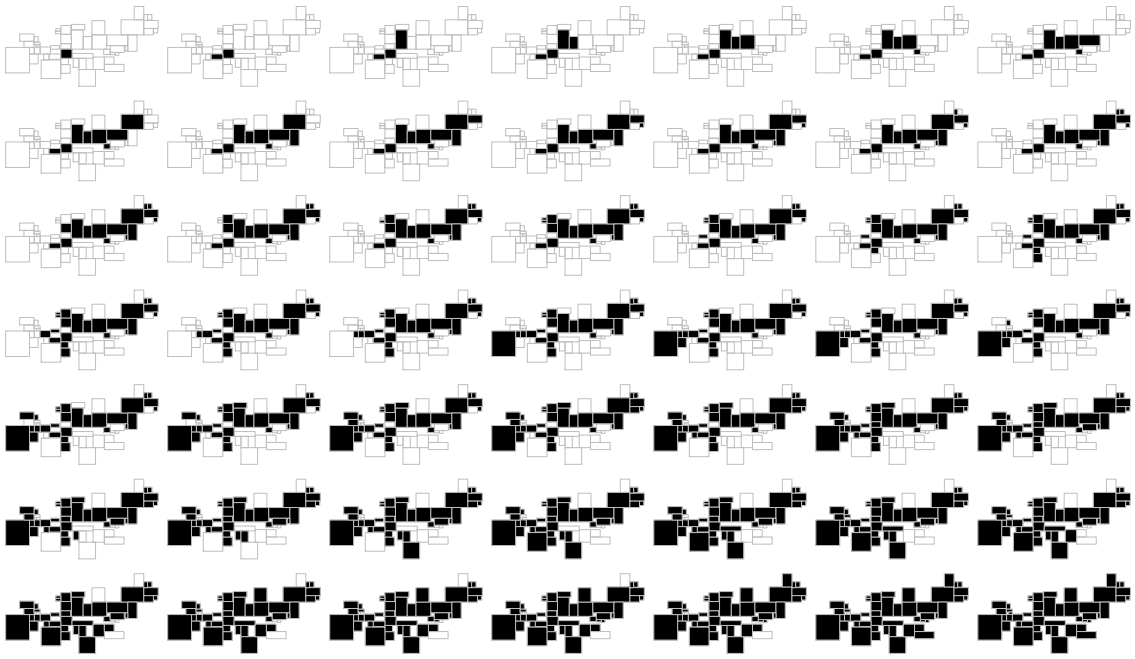


Figure 5.6: The maps display all 49 partial cartograms of *RecMap MP*2 construction procedure starting with Kansas as core polygon.

5.4 Evaluation of the Algorithm

The algorithm described above were implemented in *ANSI–C* and hence it runs on a various architectures (Intel's 386, Apple's PPC, and SUN's SPARC64) and operating systems (Microsoft Windows, OpenBSD, and Linux)[4]. The tests were performed on a 1.5 GHz Intel XEON compute server with 4 GB main memory under Linux (only 1 MB was needed by the algorithms). In this section, we will show some results using the U.S. census data base and U.S. election data.

5.4.1 Effectiveness and Efficiency

Time Complexity The most expensive parts of the algorithms are the computation of the relative position error (see equation 5.8) which takes $O(n^2)$, where n is the number of polygons. The error function has to be computed in each iteration of the meta heuristic. The number of iterations is constant. Therefore, the complexity for the *MP*1 procedure is $O(n^2)$. Since the error function has to be computed for each of n partial cartograms using *MP*2, the complexity here is $O(n^3)$.

Real data Figure 5.7 shows, for each generation within the meta heuristic of *RecMap*, the best cartogram which has been found by the construction procedure *MP*2. Figure 5.8 illustrates the respective values of the errors for both construction procedures. The scatterplot shows the conflicting goals of the error functions. The whole computation time for 10 iterations equals 0.33 seconds for (*MP*1) and for 11 iterations 55 seconds for (*MP*2) using a 2.5GHz clocked Intel XEON CPU. Each iteration step of one *RecMap* variant need the same time. Since *RecMap* gives us a useful visualization even after the first iteration a dynamic exploration of the data is guaranteed (see the cartogram yielded after Step 1 in figure 5.7).

Figure 5.9 shows how computational time and several errors such as the topology and shape error depend on each other. The items being considered here are the time span which has elapsed since the start of the optimization process ("time"), d_T ("topologyError"), $R(\overline{\mathcal{P}})$ ("relativePositionError"), d_E

[4]For an overview see figure 6.13 on page 87

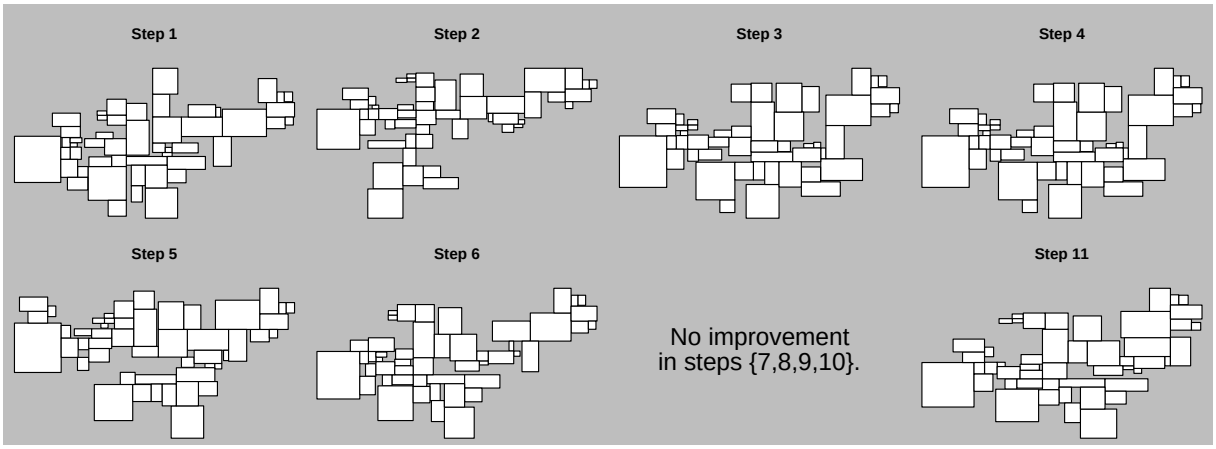


Figure 5.7: This figure demonstrates the continuous improvement of the feasible solutions for (*MP*2) with increasing number of generations. The computation time for each iteration equals 5 seconds.

"emptySpaceError", and $\widehat{f}(\overline{\mathcal{P}})$ ("weightedError"). As the values of d_A and d_S are constant, we have omitted them in this visualization. From this plot, we can conclude, that — as expected — the amounts of improvements decrease over time. Furthermore, from the plot w.r.t. d_T (x–axis) and d_E (y–axis) we can conclude that these components are negatively related (which confirms our finding in the theory part of this chapter). The results of the experiments with different parameter setting of *RecMap*'s meta heuristic (see Algorithm 6) for the *MP*1 construction procedure and the *MP*2 construction procedure are illustrated by figures 5.10(a) and 5.10(b), respectively. This kind of visualization is called *levelplot* [21, pp. 264]. It consists of two parts: On the left-hand, a matrix is shown, where the x–axis and the y–axis correspond to selection rate and mutation rate, respectively. The value of a combination is represented by a color. On the right-hand, a bar shows which value (in per cent) is associated with a color: The best value is represented by the color magenta and the worst by the color cyan. In this way, we can easily identify the best combination and — what is also important — "regions" of good and bad combinations. For the *MP*1 construction procedure, the best value for $\widehat{f}(\overline{\mathcal{P}})$ equals 0.243 with

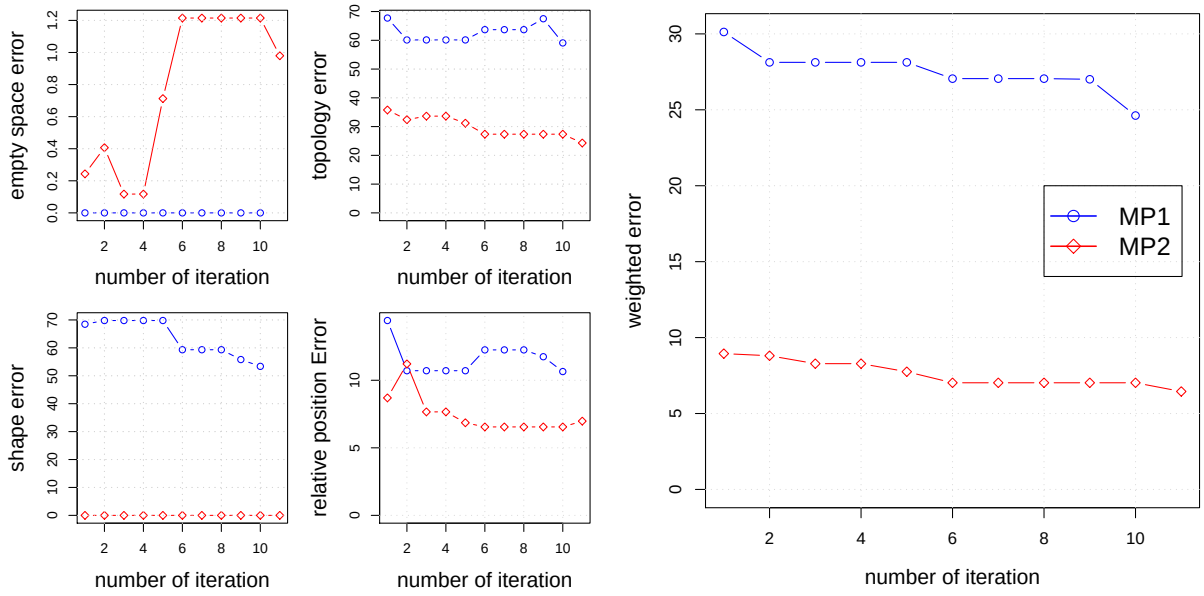
$$d_A = 0 \tag{5.30}$$
$$d_S = 0.524 \tag{5.31}$$
$$d_T = 0.582 \tag{5.32}$$
$$d_R = 0.109 \tag{5.33}$$
$$d_E = 0 \tag{5.34}$$

which is obtained for a selection rate of 0.8 and a mutation rate of 0.4 after 0.3 seconds. For the *MP*2 construction procedure, the best value for $\widehat{f}(\mathcal{P})$ equals 0.064 with

$$d_A = 0 \tag{5.35}$$
$$d_S = 0.0 \tag{5.36}$$
$$d_T = 0.245 \tag{5.37}$$
$$d_R = 0.070 \tag{5.38}$$
$$d_E = 0.006 \tag{5.39}$$

which is obtained for a selection rate of 1.0 and a mutation rate of 0.1 after 60.7 seconds.

As we can see in figure 5.10(a), good and bad combinations are nearly evenly distributed. We can merely conclude that combinations with a selection rate of 0.0 and a mutation rate of 0.0 tend to yield bad results. For the *MP*1 based procedure, we have decided to fix the selection rate 0.8 and the mutation rate at 0.4. Whereas in figure 5.10(a) no region of good combinations can be identified, we can do so



(a) Objective functions

(b) Weighted objective function

Figure 5.8: The scatterplot display the errors over the number of iteration yielded by the ($MP1$) and ($MP2$) heuristics for U.S. state level.

in figure 5.10(b): These are those combinations where the selection and mutation rates are in $[0.1, 0.4] \times [0.1, 0.4]$. From this figure, we can also clearly conclude that bad values are associated with regions where the selection rate is 0.0 and mutation rate is 1.0, i.e. where cartograms have been created randomly. On the other hand, it is important that the genotypes are "refreshed" to a certain degree, because combinations with a mutation rate of zero also tend to yield bad results. If we do not use only the best individuals of a generation but all of them (i.e. selection rate is 1.0), we mostly obtain bad values, too.

Finally, figure 5.11 illustrates the input and the results for the $MP1$ and $MP2$ construction procedures. We used the U.S. state level map and the census [124] population data. The selection and mutation rates are 0.4 and 0.3. The weights of the weighted objective function are $w_{s,e,t,r} = \{1.0, 0.3, 0.3, 1\}$. Many application examples using different data and maps of *RecMap* can be found in chapter 7.

Synthetic data Beside "real world maps" we used *RecMap* to gain artificial maps. First we generate a regular 3×3 checker board (see chapter C). We tried all combination of parameter settings for the soft constraints topology, relative position, and empty space for the $MP2$ construction procedure. As postulated earlier, on all rectangular cartograms in figure 5.12 the hard constraints area and shape were abided, which means there is no *area error* and no *shape error* in the resulting map. The core polygon on the 3×3 checkerboard map is obviously the region with the id five. All polygons are placed around it. The parameter setting of figure 5.12(e), 5.12(f), and 5.12(h) achieve the best results. However the parameter setting needs some experience with the algorithm. Since we get a visual result after a couple of seconds, an interactive visualization is guaranteed.

In figure 5.13 we increased the number of regions up to 49 and generated rectangular cartograms using the $MP1$ and $MP2$ construction procedures. On the mesh the core polygon is the region assigned with number 25. The computation time is the same as for the U.S. map. Using arbitrary sized maps we were able to analyze the computation time dependency of the number of map regions. The result can be seen in figure 5.14. The charts shows that in contrast to $MP1$ the computation time for $MP2$ increase non linear. This can be explained by the increasing search space for the $MP2$ construction procedure and the complexity of the relative position error function.

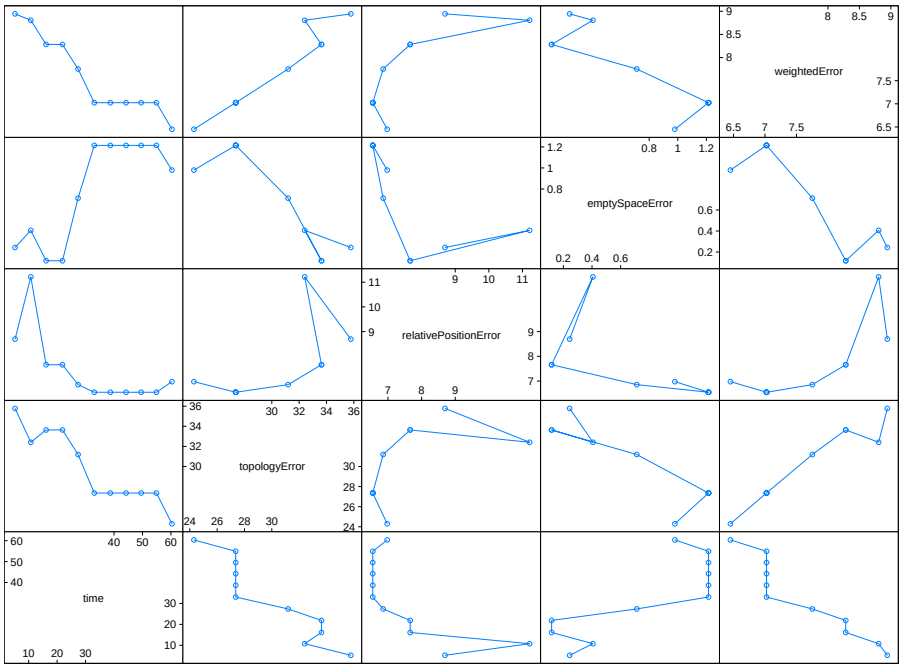


Figure 5.9: The figure illustrates the scatterplot matrix of *RecMap MP*2 using U.S. state map and the U.S. census population data.

5.4.2 Discussion

RecMap is a very fast algorithm for computing rectangular cartograms. Nevertheless, further improvements can be made. The following guideline will give some ideas for a further improvement of computation time.

First, a genetic algorithms, as Algorithm 6 is, are perfectly practical for parallel computing. Inside the meta heuristic we can run each candidate transformation call on one node. Often modern compute servers have Intel XEON systems and furthermore they have more than one single CPU. So here we can run each candidate transformation as single thread. Assuming a four processor XEON System, the same input, and the same parameter setting as in figure 5.11 we will get a result after 4 seconds.

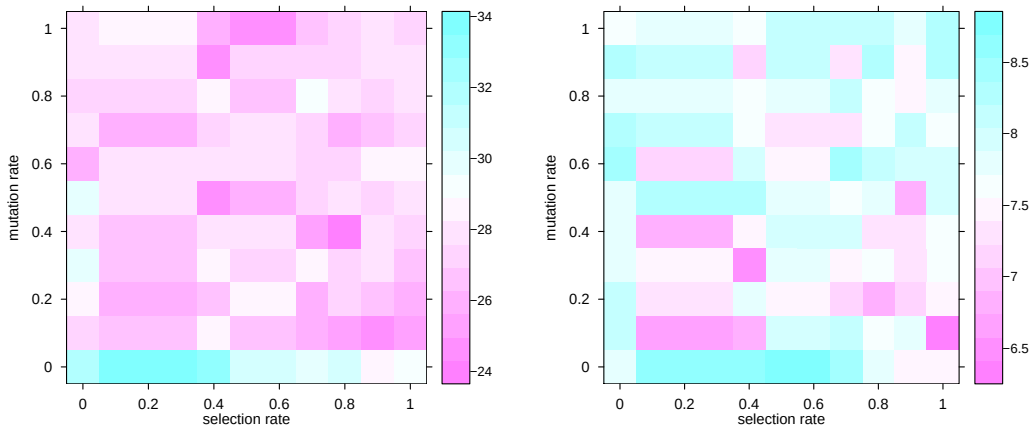
Additionally, we can reduce the search space of *MP*2 by using hierarchical layouts. This is especially useful if we have a high number of map regions (see U.S. county level mesh in the application chapter). This can be done when we group the map into subregions, e.g., assuming the U.S. map, we divide it into northeast, south, Midwest, etc. Or we use a hierarchy layout. e.g., using the U.S. map, we compute a state level cartogram and inside each state we draw a county level cartogram of the corresponding state. This grouping reduces the computation and it will make the resulting cartogram easier to understand.

The results, especially the computation time, are promising to use the *MP*1 as split procedure for the *PixelMap*-algorithm (see chapter 2) for generating high quality pixel maps.

5.5 Conclusion

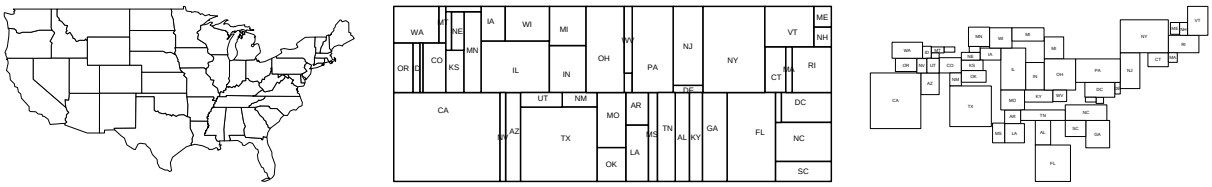
In this study we have analyzed and discussed the problem of efficient map partitioning and have proposed two automatic, scalable, and flexible algorithms called *RecMap* for generating rectangular map partitions. Here, the user has an explicit control of all visualization constraints. Our approach is novel because its features (no area error, explicit control of shape, topology, empty space, and relative position constraints) are not provided by previous approaches. This new technique enables interactive views of detail at various levels to find very fast interesting patterns or subsets.

The experiments show that our algorithms offer good results for a variety of applications, and their



(a) *RecMap* (*MP*1) (b) *RecMap* (*MP*2)

Figure 5.10: Analysis of the genetic based meta heuristic – Levelplots show the results of the selection- and mutation rate parameters. The weighted objective function is encoded by color.

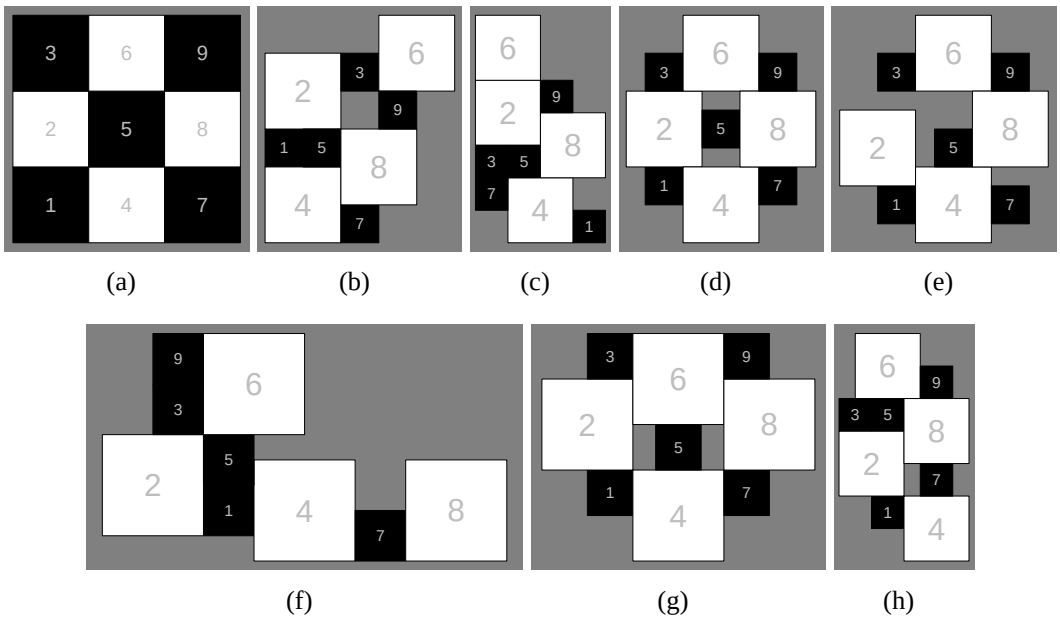


(a) U.S. map (b) *MP*1 (c) *MP*2

Figure 5.11: Results of *RecMap* for the U.S. population data

speed even allows an interactive display.

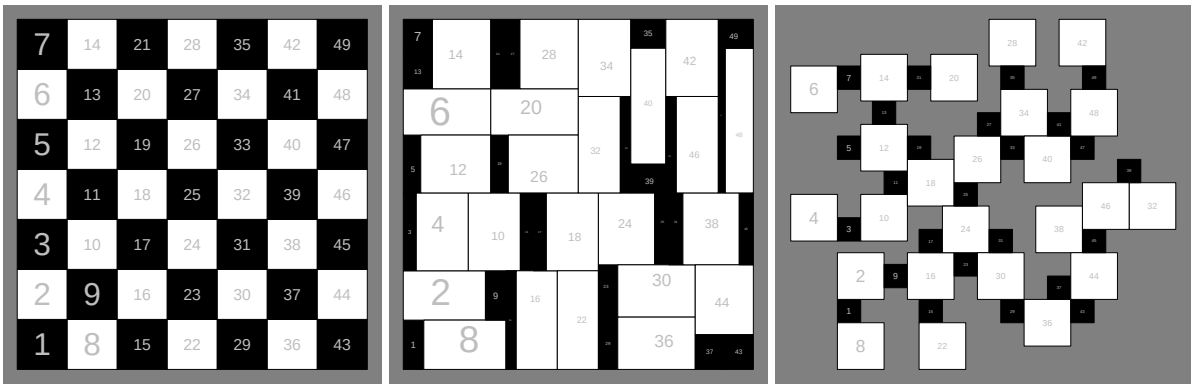
Further research could comprise the combination of our approach and other visualization techniques such as pixel–oriented techniques (where the pixels could be placed directly into their corresponding map partitions). Such a combination would allow to visualize areas with high information density. Additional material (e.g., an executable file) and ongoing work can be found on our web site `http://dbvis.inf.uni-konstanz.de/~panse/recmap`.



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

(f) (g) (h)

Figure 5.12: *RecMap MP*2 on a regular 3×3 checkerboard map using different weights – 5.12(a) the input map; 5.12(b) topology preserving; 5.12(c) empty space preserving; 5.12(d) relative position preserving; 5.12(e) relative position and empty space error preserving; 5.12(f) topology preserving and empty space error; 5.12(g) topology preserving and relative position; 5.12(h) all constraints



(a) Synthetic checkerboard map (b) Result of *MP*1 (c) Result of *MP*2

Figure 5.13: *RecMap* on synthetic 7×7 checkerboard map

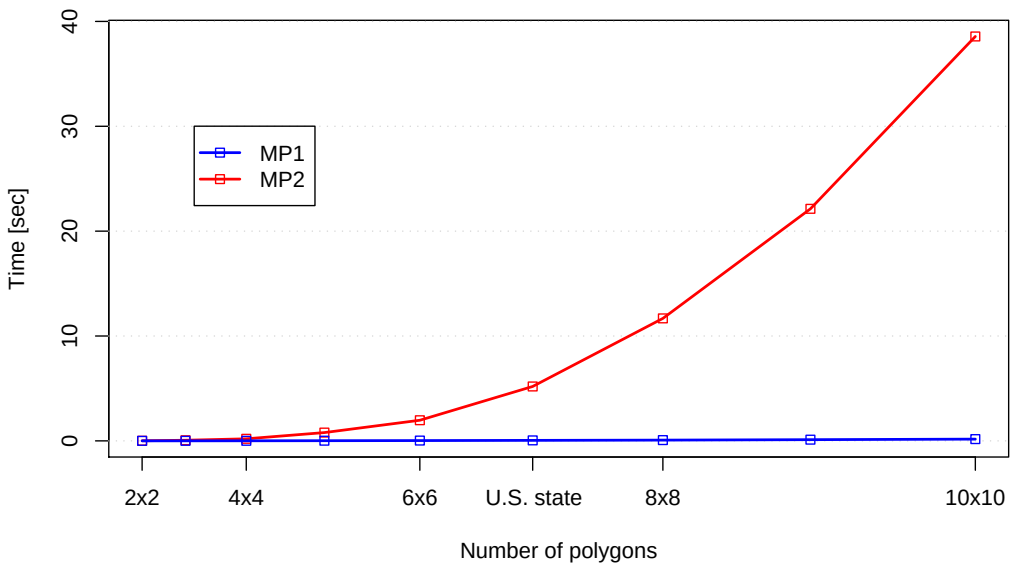


Figure 5.14: The figure displays the time versus number of polygon comparison for a single construction step for both construction heuristics using different sizes of checker boards as input mesh. The test has been performed on a Intel Pentium M 1.6 GHz CPU using 10 geno and 10 pheno types.

6 Extensions and Combinations

6.1 The *Visual Points* Solution

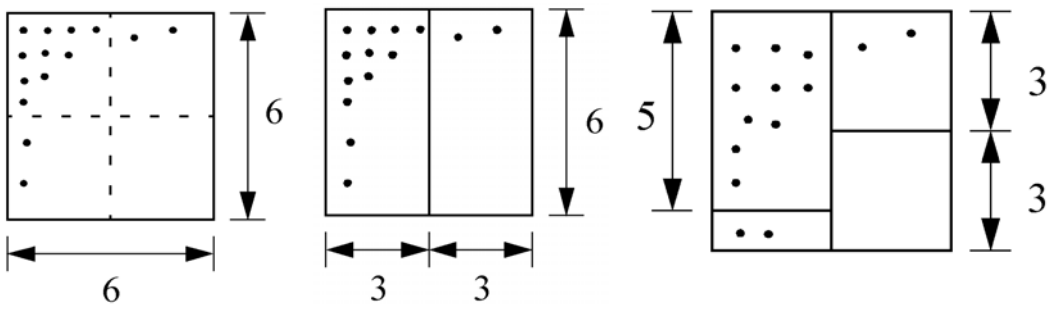
The *Visual Points* system [71] was developed to address the problem of over-plotting spatially referenced data. It works by moving points that would be drawn on already-occupied pixels to nearby unoccupied pixels, instead of over-plotting them. *Visual Points* assumes a hierarchical partitioning of the data space, to support efficient repositioning of the data points while preserving their distances and positions. In this study, we show how a similar idea called *VP-Carto* can be applied to efficient cartogram generation. The basic idea is to insert multiple points for a polygon, whose count is proportional to its target area. The points are inserted into the hierarchical data structure, and the distortion implied by the data structure is then applied to reposition the vertices of the map. Several different pixel insertion strategies are described, yielding different cartograms. Parts of this section have been published in [80].

6.1.1 The *VP-Carto* Algorithm

In each step of the *VP-Carto* construction, the data set is recursively partitioned into four subsets containing the data points in four equally-sized subregions. Since the data points may not fit into the four equally size subregions, we have to determine new extensions of the four subregions (without changing the four subsets of data points) such that the data points in each subset can be visualized in its corresponding subregion. For an efficient implementation, a quadtree-like data structure manages the required information for the recursive partitioning. The partitioning is determined as follows. Starting with the root of the quadtree, in each step the data space is partitioned into four 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 (see figure 6.1).

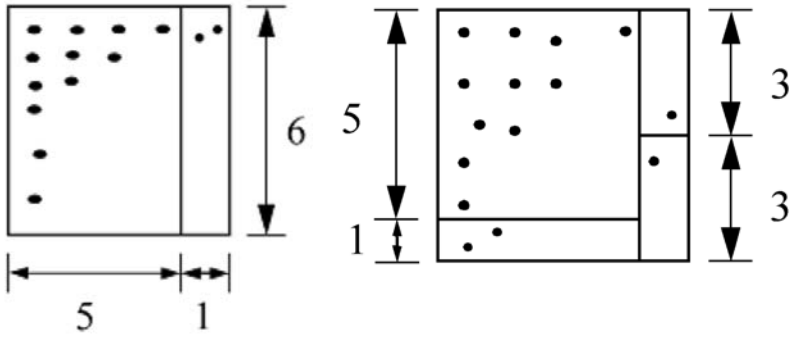
6.1.2 Generating Cartograms with *VP-Carto*

To adapt the *VP-Carto* technique to cartogram generation, a few changes need to be made to the original algorithm. The modified algorithm is shown in algorithm 9. The most important changes will be explained in more detail.



(a) Example data (b) Horizontal (c) Vertical

Figure 6.1: Original *VP-Carto* algorithm



(a) Horizontal (b) Vertical

Figure 6.2: *VP-Carto* algorithm for cartograms

Partitioning Strategy In cartogram generation we are interested in distorting maps instead of placing pixels, so the modified algorithm has a different *partitioning strategy*. In the original *VP-Carto* system, the borders between the quadtree partitions are only shifted as much as needed to accommodate all pixels in the quadrant. For cartogram generation, the borders are shifted according to the ratio of the number of pixels in the neighboring quadrants. For example, with the original *VP-Carto* algorithm, there is no change in the first step of figure 6.1(b), since there is enough space (18 pixels) in the left partition to accommodate all twelve data points. In the modified algorithm, the border shifts proportionately to the number of data points, i.e. in a ratio of 12:2 resulting in the partition shown in figure 6.2(a). Note that the result of the second step is also different (compare figure 6.1(c) and 6.2b).

```
VPcarto(𝒫, X̃){
  Quadtree Q; /* empty initialized Quadtree */
  foreach (polygons P ∈ 𝒫){
      point cur = FindStartPoint(P);
      while (pc < P.DesiredArea(𝒳)){
          /* area is represented as pixels */
          now = P.ComputeNextPosition(cur); /* depends on insert strategy */
          Q.InsertQuadtree(cur);
          pc = pc + 1;
      } /* while */
  } /* foreach */
  TransformQuadtree(Q);
  /* moves the borders of the quadtree */
  foreach (polygons P ∈ 𝒫){
      foreach (point p ∈ P){
          qnode node = Q.FindNode(p);
          p = scale(node, p);
          /* depends on new height and width of node */
      } /* foreach */
  } /* foreach */
```

Algorithm 9: *VPCarto*

No Pixel Placement. A second difference between the original and the modified *VP-Carto* algorithm is that pixels don't need to be positioned. They are only needed to determine an optimal partitioning of the

modified quadtree for the subsequent transformation of map polygons. It is also no longer necessary to search for free space to avoid overlapping pixels. Since pixels don't need to be positioned, we can further optimize the space and time complexity of the algorithm by storing a pixel at a given position only once.

Pixel Insertion Strategies. To scale the polygons according to their desired size, we represent the polygons by pixels. If a polygon needs to shrink, we insert fewer pixels than what its shape accommodates, thus creating free space; if a polygon needs to expand, an excess of pixels are inserted, leading to overlapping pixels. The idea is to distort the map such that all pixels can be placed without overlap. In the best case the overlapping pixels of the growing polygons use the free space of neighboring shrinking polygons. The pixel insertion strategy determines where the pixels are placed for growing and shrinking polygons. We tried the following strategies:

- **Bottom – Top:** Shrinking polygons are filled with pixels starting at the top and going downward until all pixels are set, and the overflow pixels are positioned at the top of the expanding polygons (see figure 6.3(a)).
- **Left – Right:** Shrinking polygons are filled with pixels starting on the left going right, and the overflow pixels are positioned at the right sides of the expanding polygons (see figure 6.3(b)).
- **Center – Outside:** Shrinking polygons are filled with pixels from the center going outward, and the overflow pixels are positioned at the edges of the expanding polygons (see figure 6.3(c)).

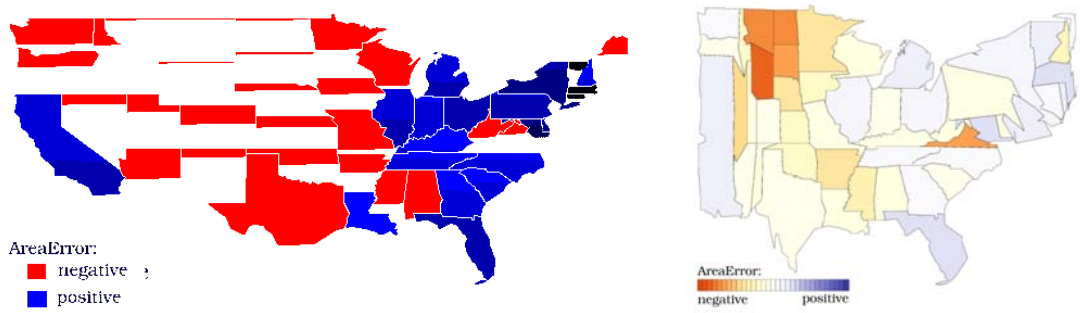
Observe that pixels are only used to construct the quadtree-like data structure but are not actually positioned as in the case of the *VP-Carto* system, so the exact position of each pixel is not important. As figure 6.3(a–c) shows, the pixel insertion strategy is of great importance for the quality of the resulting cartograms, especially with respect to the shape of the polygons and the overlap of edges. The differences result from the different partitioning of the quadtree induced by the insertion strategies.

Determination of the Polygon Mesh After the quadtree is constructed, it is applied to distort the vertices of the polygon mesh. Each vertex is repositioned separately: First, the cell of the quadtree containing the vertex is found. Then the new position of the vertex is calculated by scaling the cells of the quadtree on each level according to the desired size of the cells (corresponding to the number of pixels). By repositioning each vertex, we iteratively construct the distorted polygon mesh.

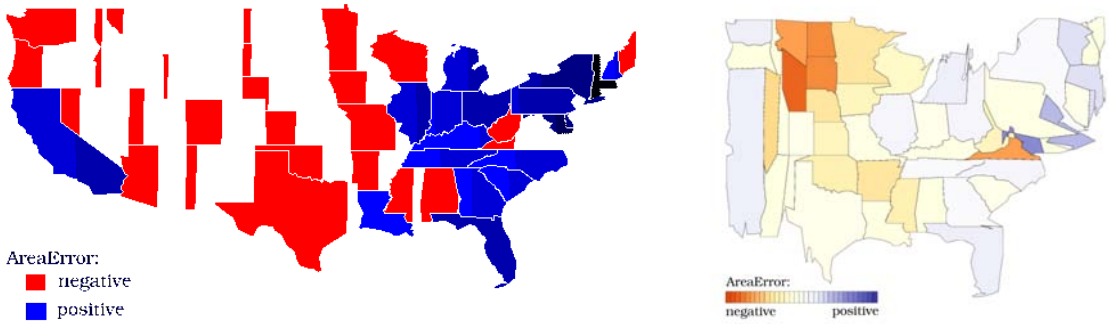
6.1.3 Efficiency and Effectiveness

The *CartoDraw* algorithm described in chapter 4 was implemented in *C++* using the LEDA library [97] and the *VPCarto* algorithm described in section 6.1 was implemented in Java. The tests reported in this section were performed on a 1 GHz Pentium computer with 512 Mbytes of main memory. In the following, we report and discuss the results and compare the effectiveness and efficiency of both approaches.

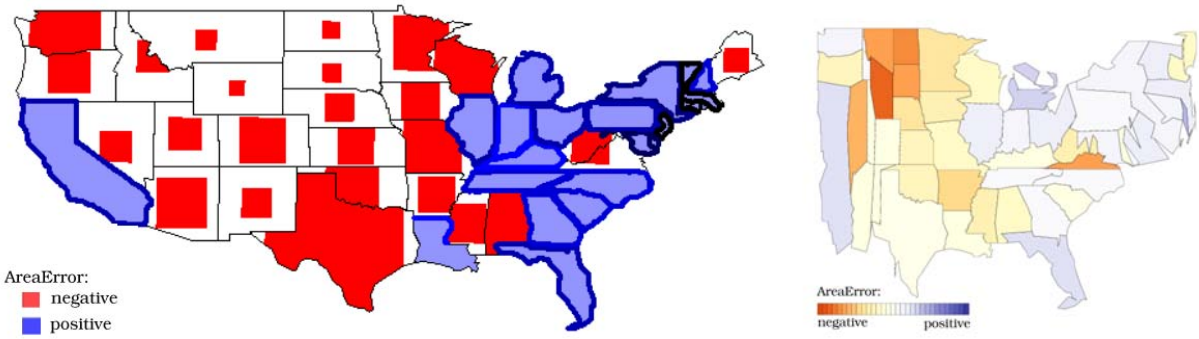
Figure 6.4 shows the measured efficiency and effectiveness results. The total run time was 3 seconds for the new *VP-Carto* approach, 25 seconds for the automatic scanline approach, and 16 hours for the non-linear optimization approach by Kocmoud and House [90] (The comparison assumes that all algorithms run on a 120 MHz computer.). Note that the scale on the y-axes of figure 6.4 is logarithmic. The *VPCarto* approach is more than four orders of magnitude faster than the Kocmoud and House approach, about two orders of magnitude faster than the interactive scanlines, and about one order of magnitude faster than the automatic scanlines. Since the *VPCarto* algorithm has no explicit notion of shape, its shape preservation is not as good as that of *CartoDraw*. Figure 6.4(b) compares shape versus area error for population cartograms made with *VPCarto* and interactive scanlines, measured on the four population



(a) Top – bottom



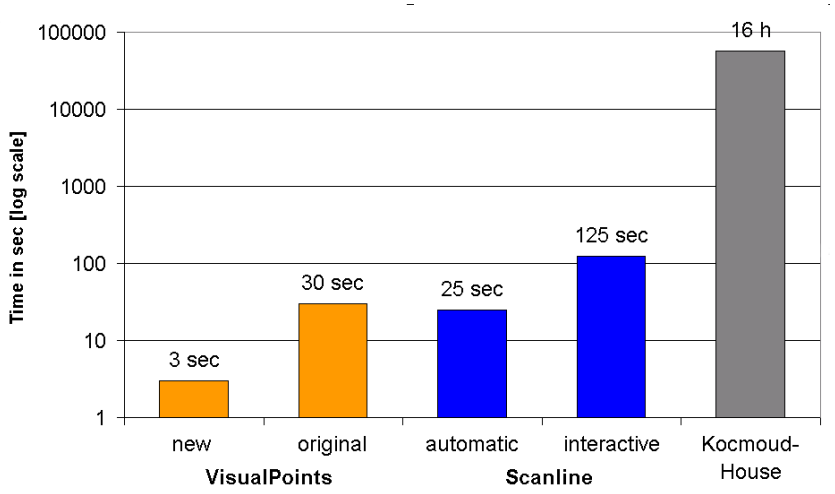
(b) Left – right



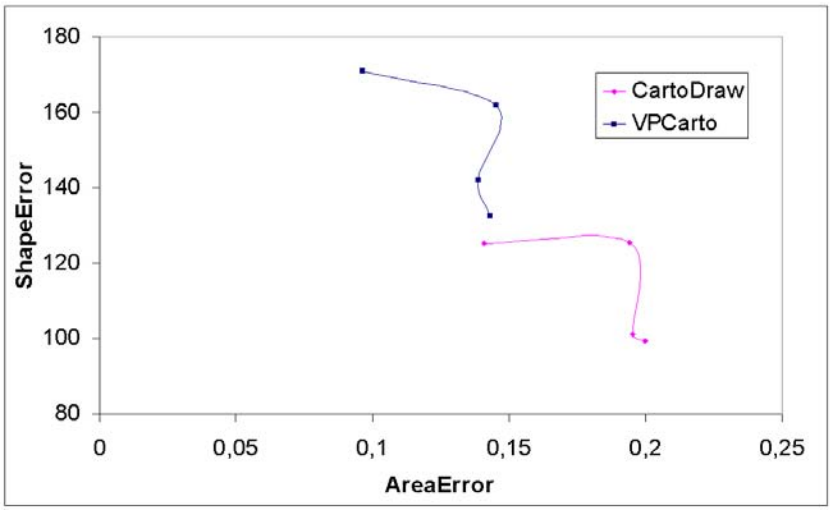
(c) Center – outside

Figure 6.3: Insertion strategies

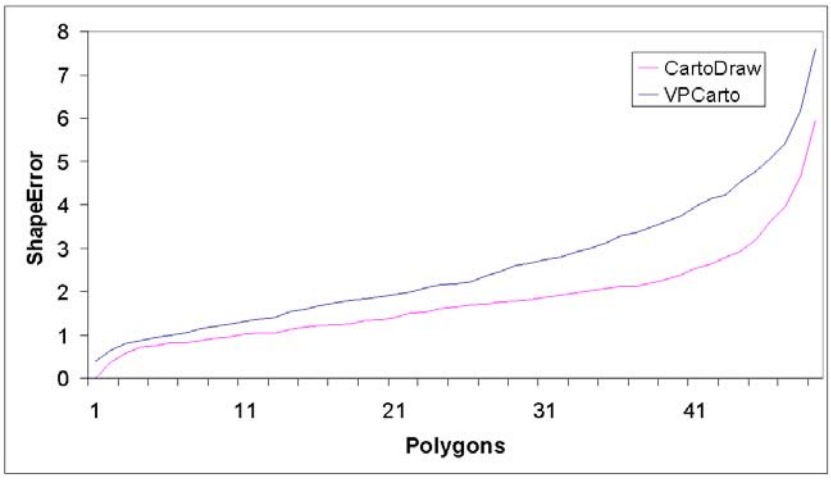
cartograms. The results clearly indicate that the shape error of the *CartoDraw* (interactive scanlines) is always considerably better than that of the *VPCarto* results, and slightly worse for the area error. Since the total shape error is basically an average over the state-wise area error, figure 6.4(c) shows the shape error by state, sorted by shape error. Figure 6.4(c) reveals that the *CartoDraw* algorithm consistently provides a lower shape error than the *VPCarto* algorithm.



(a) Run time comparison



(b) Shape error versus area error



(c) Sorted shape

Figure 6.4: Efficiency and effectiveness results

6.2 *HistoScale*

6.2.1 Introduction

In addition to the classical applications mentioned in the chapter above, a key motivation for cartograms as a general information visualization technique is to have a method for trading-off shape and area adjustments. *pseudo-cartograms* provide an efficient and convenient approximation of contiguous cartograms, since a complete computation of cartograms is expensive. In this section, we propose an efficient method called *HistoScale* to compute Pseudo-Cartograms. Parts of this section have been published in [78].

6.2.2 HistoScale Approach

The basic idea of the *HistoScale* method is to distort the map regions along the two Euclidean dimensions x and y. The distortion depends on two parameters, the number of data items which are geographically located in this map area, and the area covered by this map region on the underlying map. The new pixel position (x,y) of a geo-location (λ,ϕ) can be computed by solving the integrals (see Tobler [121])

$$x = \int_{-\pi}^{\lambda} d_x(\lambda)d\lambda \tag{6.1}$$

$$y = \int_{-\pi}^{\phi} d_y(\phi)d\phi. \tag{6.2}$$

The distortion operations can be efficiently performed by computing two histograms with a given number of bins in two Euclidean dimensions x and y to determine the distribution of the geo-spatial data items in these dimensions. The two histograms are independent from each other, that means, the computation of the histograms can be random. The two consecutive operations in the two Euclidean dimensions x and y realize a grid which is placed on the map. The number of histogram bins can be given by the user.

Lets consider an example here we want to transformation along a x direction. As *input* we have a polygon mesh $\mathcal{P}$ and a parameter vector $\mathcal{X} = (x_i)_{i=1,\ldots,n}$ where $x_i = (\lambda_i, \phi_i, z_i)$.

The area of each bin corresponds to the statistical value. The transformation is separated into two steps.

In a pre-processing step the histogram h_x is computed by summing up the statistical values. The histogram bins are realized as an integer array. Next, we cumulate the cells of the integer array and achieve d_x.

In a second step, the transformation of the map is to be done. For each data point of the mesh the new position is determined. The transformation of each point is made by a bilinear interpolation between the histogram bins h_x and the cumulated histogram d_x (similar to the *Scanline*-step of *CaroDraw*).

As *output* we achieve a map transformation $\overline{\mathcal{P}}$ here the area of each region is approximated according to the histogram bins.

For a practicable visualization we suggest a number of 256 histogram bins for both histograms.

Figure 6.5 demonstrates the idea of *HistoScale*.

6.2.3 Evaluation

The resulting output maps are referred to as pseudo-cartograms, since they are only approximations to the true cartogram solution. On the other hand our approach generates interesting maps and good solutions in least square sense.

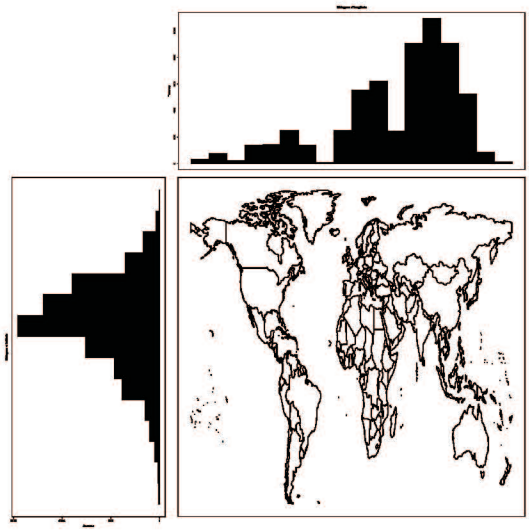
Efficiency The computation of pseudo-histograms using our *HistoScale* algorithm can be done in real-time (see figure 6.6). Due to the run time behavior, *HistoScale* can be used as a pre-processing step for other cartogram algorithms. The complexity of the *HistoScale* approach is $O(|\mathcal{P}| + |\mathcal{X}|)$.

HistoScale-longitude$(\mathcal{P},\mathcal{X})\{$
  /* STEP 1 */
  for$(i=0;i\leq|\mathcal{X}|;i++)\{$
    $h[round(\lambda_i]=h[round(\lambda_i)]+z_i;$
  $\}$
  for$(i=1;i\leq|h|;i++)\{$
    $d[i]=h[i]+d[i-1];$
  $\}$
  /* STEP 2 */
  foreach$((\lambda,\Psi)\in\mathcal{P})\{$
    $(x,y)=cmptBlnrntrpltn(h,d,(\lambda,\Psi));$
    print$(x,y);$
  $\}$
$\}$

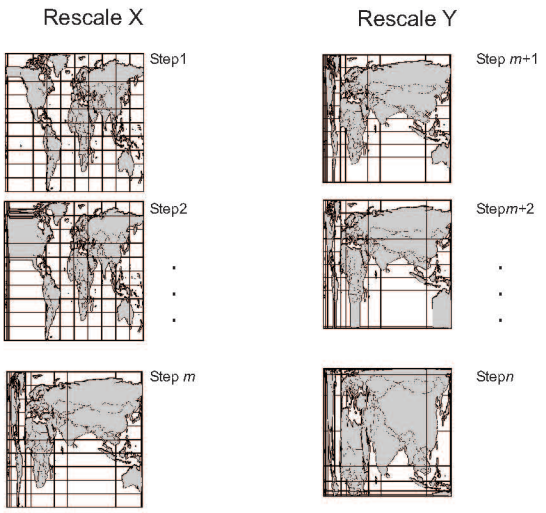
Algorithm 10: *HistoScale*

Figure 6.6 shows, that the computation time of the *CartoDraw* algorithm can be reduced without losing any quality. In the application part of this thesis (see chapter 7) the reader will find several interesting applications using our *HistoScale* algorithm.

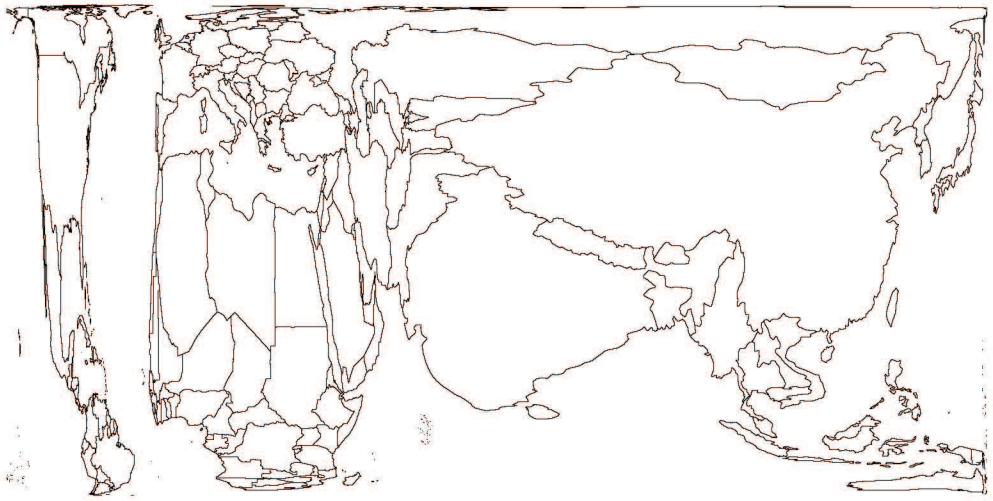
The world population pseudo-cartogram shows clearly, that China and India are the most populated world regions. This fact has e.g., an important influence on the evolution of epidemics such as SARS, as unknown epidemics in such areas can be dangerous for the whole world population.



(a) Step 1: Determine an efficient approximation of the geo-spatial data distribution using histograms with a given number of bins in two Euclidean dimensions x and y



(b) Step 2: Rescaling the map regions: every histogram bin defines a minimal bounding box on the underlying map area covered by the minimal bounding box.



(c) Result: The figure displays a world population cartogram as a result of the *HistoScale* algorithm.

Figure 6.5: *HistoScale*-computation steps – the figures demonstrate the main idea of the algorithm using the world population data.

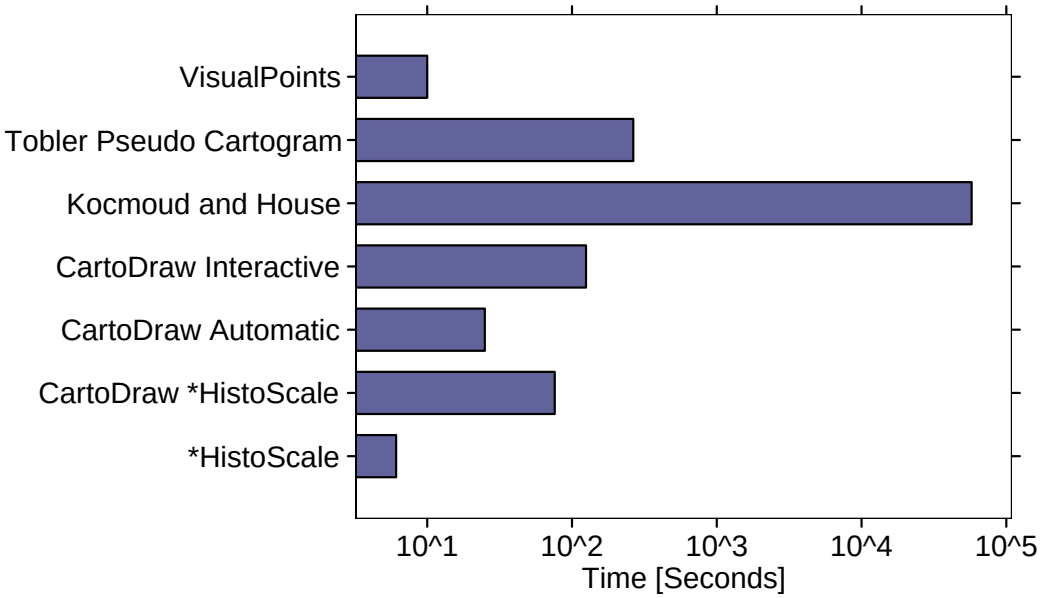


Figure 6.6: Time comparison - we have assumed a 120MHz Intel CPU to compute the U.S. state cartograms

6.3 *HistoMap*: A Combination of *HistoScale* and *RecMap*

Often data analysts have only x-y-location as input and want a value-by-area cartogram as visualization. We can combine the *HistoScale* and *RecMap* approaches and get a new method called *HistoMap*.

The visualization goals are as follows:

1. using the full screen size for the visualization (no holes),
2. each location should be realized as a rectangle or at least as a display pixel,
3. the relative geographic position of each location should be preserved, and
4. the aspect ratio of each box should be equalized.

6.3.1 Problem Definition

As *input* we have a point set $\mathcal{P} = \{p_1, \ldots, p_n\}$ and $(p_i)_{i=1,\ldots,n} \in \mathbb{R}^2$ and a vector $\mathcal{X}$ of statistical values where $\mathcal{X} = (x_i)_{i=1,\ldots,n}$ with $x_i > 0$ and $x_i \in \mathbb{N}$.

The quality of $\overline{\mathcal{P}}$ depends on two aspects. First, we want a space filling visualization where the area of each rectangle corresponds to the statistical value of each location and second, the "map" should be easily recognized as in $\mathcal{P}$.

These goals can be achieved using three objective functions which are:
The *absolute point position* distance measures the difference between the x-y-location and the center of the resulting rectangle. The *absolute point position* d_{AP} can be expressed by

$$\begin{aligned} d_{AP} &= d_{AP}(\mathcal{P}, \overline{\mathcal{P}}) && (6.3)\\ &= \sum_{i=0}^{n-1} |p_i - \tilde{p}_i| && (6.4) \end{aligned}$$

and the *relative point position*

$$\begin{aligned} d_{RP} &= d_{RP}(\mathcal{P}, \overline{\mathcal{P}}) && (6.5)\\ &= \sum_{i=0}^{n-1} \sum_{j=0, j\neq i}^{n-1} (|p_i - p_j| - |\tilde{p}_i - \tilde{p}_j|)^2 && (6.6) \end{aligned}$$

which may be used as measures for the achievement of the neighborhood. The *aspect ratio error* d_{AR} reflects the average relative deviation of the aspect ratios of the rectangles in $\overline{\mathcal{P}}$ and can be determined as follows:

$$\begin{aligned} d_{AR} &= d_{AR}(\mathcal{P}, \overline{\mathcal{P}}) && (6.7)\\ &= \frac{1}{n} \cdot \sum_{i=1}^{n-1} |1 - \frac{dy_i}{dx_i}| && (6.8) \end{aligned}$$

The distance function $|p_i - p_j|$ can be defined by an L^m-norm ($m = 1$ or 2)

$$|p_i - p_j| = \sqrt[m]{(p_i^x - p_j^x)^m + (p_i^y - p_j^y)^m}. \quad (6.9)$$

The *output* can be defined as an optimization problem. The *output* is a

- non-overlapping,
- planar, and

- space filling

map partition $\overline{\mathcal{P}}$ where $\{w_{rp}, w_{ap}, w_{ar}\} \in \mathbb{R}$, $w_{rp} + w_{ap} + w_{ar} = 1$, $\{w_{rp}, w_{ap}, w_{ar}\} \geq 0$, and

$$\sum_{i=1}^{n-1} A(\tilde{p}) = x_i \tag{6.10}$$

$$w_{rp} \cdot d_{RP} + w_{ap} \cdot d_{AP} + w_{ar} \cdot d_{AR} \text{ is minimized.} \tag{6.11}$$

6.3.2 Solution

We provide two solutions for the problem mentioned above. The first solution is based on the previously described *HistoScale* approach. The algorithm is extended in that way, that we have an arbitrary number of histogram bins which divide the screen space. For each bin we alternate between horizontal and vertical direction of the binning until each bin corresponds to one x-y-location and the area to the statistical value. In each binning step we have a flexible number of binning. Using a meta heuristic as described in the *RecMap* chapter it is possible to minimize to the weighted position and aspect constraints.

Our second approach based completely on the *RecMap MP*1 construction procedure. Hence we compute a candidate splitting sequence of all x-y-location, determine the map partition, evaluate the candidate transformation. The transformation with the lowest error of all candidate transformations will be made persistent.

6.3.3 An Example

As in many research groups, the communication in our group is based on emails. First, we applied the solution to email data. We visualized all emails which have been classified as SPAM [113] during the last two years. Therefore, we had to determine the x-y-location of the senders address. For that we used a Geo-IP-DB [96]. The result can be seen in figure 6.7

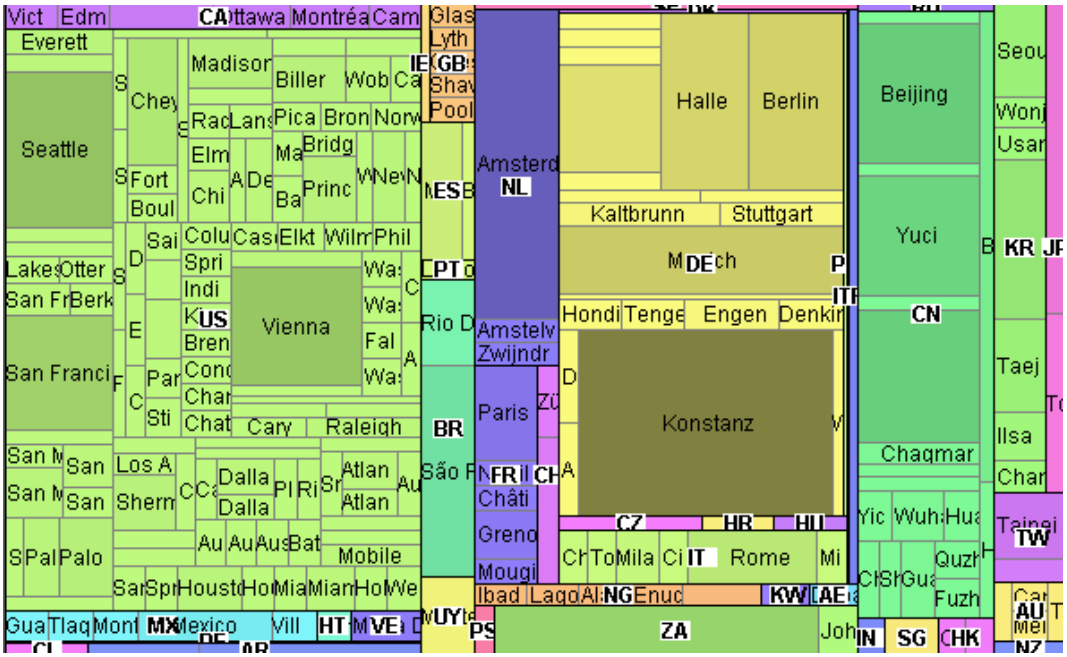


Figure 6.7: This image shows the distribution of SPAM mail reached our IMAP server. Each country has a unique color. The brightness is adjusted to the number of SPAM mails. (Thanks to Florian Mansmann for implementation and generation.) [72]

6.4 Combining *PixelMap* and *RecMap*

The *RecMap* technique is useful for data exploration. The map is distorted in a way that each area corresponds to a first statistical value and hereafter, color is used for expressing quantities of a second statistical value.

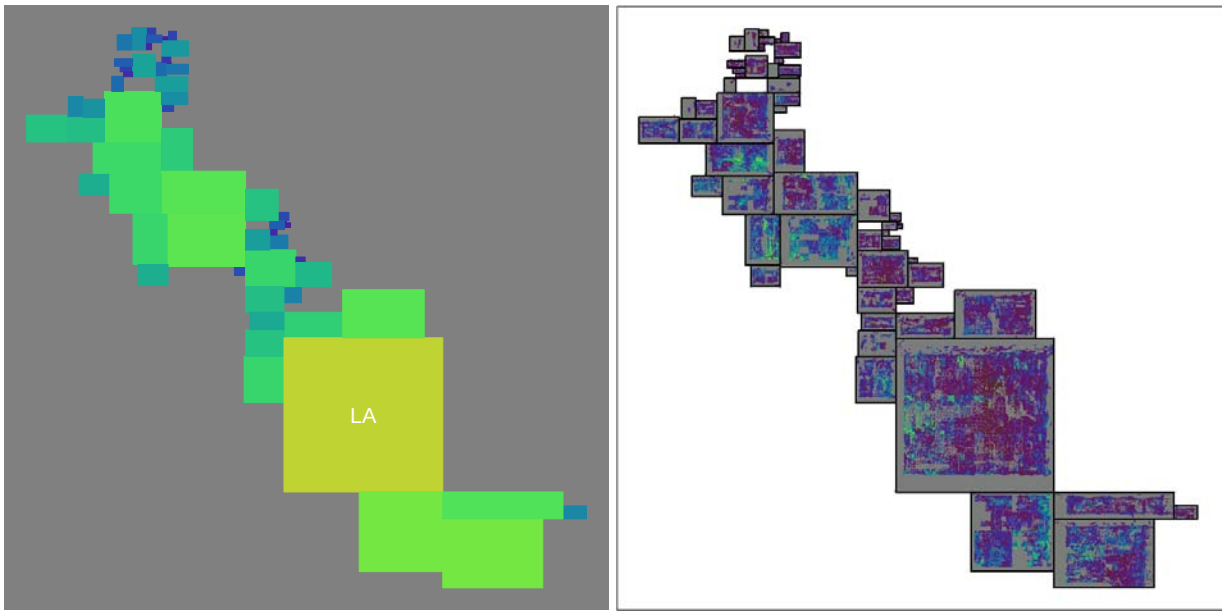
If we have a high number of x-y-locations, we can combine *RecMap* cartograms with the *PixelMap* technique which is a pixel-based visualization (see also [111] or chapter 2). This combination is useful because a major drawback of the *PixelMap* technique is that from the resulting pixel visualization it can be difficult to identify the shape of the input map regions.

The combination described here combination is promising because on one hand *RecMap* produces a set of rectangles as layout and on the other hand *PixelMap* can only place pixels inside a rectangle. Furthermore, using *RecMap* we have a user control to achieve the relative position of the map regions which can be useful for a fast exploration. Our experience with *RecMap* has shown that users are able to recognize familiar regions very quickly.

The input data has to be separated into regional categories. The input map must be distorted according to the number of pixels multiplied by a positive constant number.

Next, for each region the *PixelMap* procedure is used to place each x-y-location in the pre-defined rectangle. As an additional parameter we give *PixelMap* the aspect ratio of the map region.

Figure 6.8 shows a result for the U.S. state California. The area corresponds to the number of households (multiplied by 1.5) while each pixel in figure 6.8(b) represents one single household. The pixels are placed according to the median household income of the U.S. census data base [124]. The uni-polar color map identifies eight different income classes. The here described combination has one drawback that can



(a) Rectangular cartogram (b) Combination of *PixelMap* and *RecMap*

Figure 6.8: *PixelMap*-cartogram of California

also be seen in the figure. Since *RecMap*' layouts depend on the axes parallel transformation screen place is squandered if we have a diagonal direction of the map regions.

6.5 Texture Cartograms

6.5.1 Introduction

Usually most people are not very familiar with maps drawn from unknown territory. If there is an additional distortion on the map, probably a lot of people do not understand the visualization. Textures can help to fix this problem. On a texture the user has an additional orientation to navigate through the map and find well-known positions. These can be for example mountains, rivers, lakes, or even big streets; things which can be seen from the birds perspective.

Furthermore, apart from generating traditional cartograms it is of interest what happens to the relief after the distortion. Let's consider a population cartogram which would enlarge the relief surface where people live and contract these parts of the relief where less people live. Figure 6.10 shows the reliefs during the construction steps of the cartogram. It can be seen that during the construction the mountain regions are contracted in contrast to the coastline where the relief has been extracted.

Another application could be the design of route maps algorithms. In this case population cartograms could be used again for drawing street details. Areas with a high number of street crossings could be enlarged and undeveloped area could be contracted. Using cartograms it is possible to get an overall view of the map.

6.5.2 The Algorithm

Texture Mapping The technique is well known from the computer graphics [135, 44, 131]. Textures can indicate rendering informations such as contours, colors, or even images. These so-called textures can be pinned on arbitrary surfaces. As an example one could put an image on a wall in an ,,info cube" as it can be seen in figure 6.9(a).

Texture Mapping techniques can be applied in 1D, 2D, 3D or even in higher dimensions. For our usage we are just interested in *2D-Texture Mapping*. Texture mapping first maps the *image space* to the *texture space*. In a second step the *texture space* has to be mapped to the *object space*. The mapping can be done by bilinear interpolation.

Cartogram Texture Mapping The techniques can be used in combination with any topology preserving contiguous cartogram drawing algorithm. The mapping described above can be done using the mapping function:

$$f_C : \mathbb{R}^2 \rightarrow [0,1] \times [0,1] \tag{6.12}$$

where f_C is basically that what our cartogram algorithm does and has the following properties:

$$f_C(p) = p' \leftrightarrow p \in \mathcal{P} \wedge f_C^{-1}(f_C(p)) = p. \tag{6.13}$$

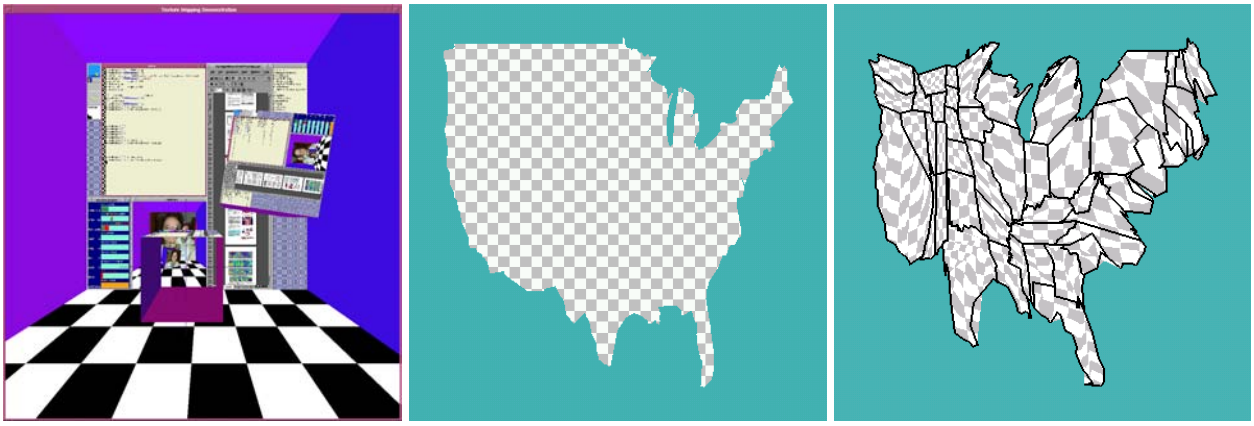
f_C can transform each point of the mesh $\mathcal{P}$ into its new position. In practice the mapping can be realized using an array which stores for each point of the mesh the new position.

With help of this function we are now able to compute the texture cartogram in two steps.

1. In a pre-processing step we first have to pin each point of the mesh $\mathcal{P}$ onto the texture which is in our case an image. An result of that step can be seen in figure 6.9(b) where the mesh of the U.S. has been pinned on a checker board texture.

2. After that initialization we iterate over each polygon $P \in \mathcal{P}$. In a second loop we map each point $p \in P$ to its new position p' using f_C.

The *texture mapping* of the graphic library has to ensure that the texels are right placed on the resulting visualization.



(a) Info cube (b) Step 1: pinning the U.S. on a checker board (c) Step 2: Transform the mesh

Figure 6.9: Demonstration of *Texture Mapping*

Implementation *Texture mapping* are supported by many computer graphic libraries. We have used *OpenGL*, Version 1.2 [135] and *ANSI-C*[89].

The algorithm reads as input an arbitrary SGI formatted image with a resolution of $2^i \times 2^i$ where $i \in \{1,\ldots,12\}$ and a file which contains the polygon meshes $\mathcal{P}$ and $\mathcal{P}'$ as well as the one-to-one mapping function f_C. In the file the points of the polygons are saved line by line. The one-to-one mapping is realized by saving p and p' in one line separated by a delimiter.

In a pre-processing step the image has to be read into the texture buffer. It has to be ensured that the texture has been clipped with the minimal bounding box to $\mathcal{P}$; Using the method shown in algorithm 11 the *cartogram texture mapping* can begin.

Please note: *OpenGL* does not support rendering concave filled polygons. Before the rendering non convex polygon has to be tessellated. *OpenGL* has routines doing that. Because of clearance we did not include that in algorithm 11. Alternatively, the cartogram drawing algorithm can give a triangulate mesh as output.

Example Figure 6.10 displays a *multi-transformation view* of the U.S. population data using a relief map.

Figure 6.10: The figure shows 16 of 100 frames of a *multitransformation view* of the U.S. population data using a relief map [41]. All 100 frames can be downloaded as mpeg stream from [76]

6.5.3 Conclusion

In that section, we described how we can apply *texture mapping* to cartogram drawing. We also give an implementation of the algorithm. Apart from that, we have seen that *texture mapping cartograms* are a

```
static void drawTextureCartogram()
{
  FILE *CartogramMappingFile;
  float x, y, vx, vy, x2, y2, vx2, vy2;

  determineMinMax();

  if ((CartogramMappingFile = fopen(MappingFileName, "r")) != NULL) {
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, texName[1]);

  glBegin(GL_POLYGON);
  /* foreach (polygon P ∈ 𝒫){foreach (point p ∈ P){...}} */
  while (fscanf(CartogramMappingFile, "%f %f %f %f", &x, &y, &x2, &y2) != EOF) {
  if (x != NA && y != NA) {
     /* Normalize p = (vx,vy) ∈ [0,1] and (vx2,vy2) = f_C(p) ∈ [0,1] */
    vx = (x - xmin) / (xmax - xmin); vy = (y - ymin) / (ymax - ymin);
    vx2 = (x2 - xmin2) / (xmax2 - xmin2); vy2 = (y2 - ymin2) / (ymax2 - ymin2);

    glTexCoord2f(vx, vy);
    if (ORGINALMAP == 1)
      glVertex2f(vx, vy);
    else
      /* draw(f_C(p)) */
      glVertex2f(vx2, vy2);
    } else {
    glEnd();
    glBegin(GL_POLYGON);
    }
  }
  glEnd();

  fclose(CartogramMappingFile);
  glDisable(GL_TEXTURE_2D);
  } else {
  printf("Can't open file %s", MappingFileName);
  exit(1);
  }
}
```

Algorithm 11: This algorithm computes the cartogram *texture mapping* using the *OpenGL*, Version 1.2 library. The texture mapping routine assumes that all polygon are convex. If that is not the case, the polygon tessellation must be used (see [135, pp. 467] for more details)

powerful tool for *multi-scale viewing* which can help the user for an inductive orientation. Some examples of the techniques described above can be seen in the application part of this thesis.

6.6 *CartoDraw*-System

6.6.1 The Graphical User Interface

All of the described algorithms are integrated into one *graphical user interface (GUI)* called *CartoDraw-System*. Figure 6.11 demonstrates a screen-shot of the *CartoDraw-System*.

The main goal of this design was to have a handy program where the user can select the input data and determine the desired algorithm to compute the cartogram transformation.

Because of the development history we used the LEDA [97] window and panel objects for realizing the GUI [1].

An additional feature of LEDA is that we have a very powerful graph editor for debugging the input map. An extraordinary fast processor and a high resolution wall (see 2.6) are of value, especially for the "graphical debugging" of large maps as the U.S. continental county map.

The GUI provides standard input and output file-system browsing menus, many menus for parameter settings (see e.g., figure 6.11(a), 6.11(b)), and a graphical representation of the map. All data can be labeled. One of the main features is the color map setting. We have implemented the linear, logarithmic, square-root, and area error color mapping functions (see chapter B). Using a quantile function it is also possible to stress some polygons using brightness. Because of the fact that humans have different perceptional taste concerning color our frame work can load arbitrary color maps. The standard color maps which are included into the *CartoDraw-System* can be see in chapter B.

Furthermore, the GUI provides several pre-processing scripts, such as data consistence checks, and several drawing routines.

6.6.2 Extensions

During the development of algorithms we were interested to integrate our software into existing GIS components to make it usable for a wide range of users. Beside a number of open source projects [48, 116, 52], a well-known commercial product is GIS ArcView from ESRI [39]. In a feasibility study we developed a routine which can export and import the desired format that our *CartoDraw*-Sytem understands. A result can bee seen in figure 6.12 were we used the *RecMap* algorithm and the U.S. population data in combination with some ESRI bar chart visualizations.

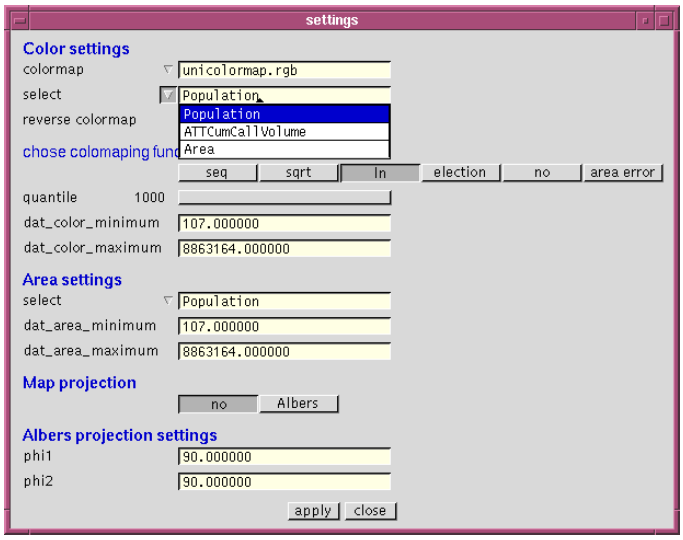
6.6.3 System Portability

Since *CartoDraw* uses the LEDA-library the portability is restricted to the systems where LEDA runs. *CartoDraw* runs on a number of architectures (SUN'sparc64, i386, SGI'MIPS) and a number of operation systems (SUNOS 5.8, SUNOS 5.9, SGI'IRIX, Linux, OpenBSD, MS'Windows) using the GNU'`g++` and Microsoft'`vc` compiler. *RecMap* was implemented later and the designer decided to use *ANSI-C* as programming language to make it as portable as possible. Figure 6.13 illustrates a time comparison using different platforms and operation systems.

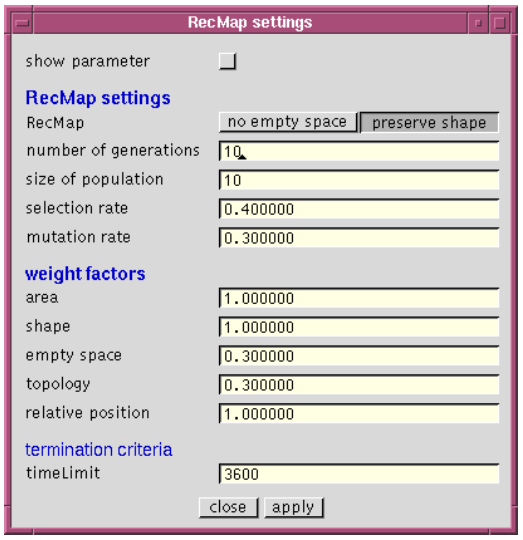
6.6.4 Outlook

To make the described algorithm and pre-processing steps accessible we have designed a client server based architecture. The idea is that users can send the input data via a remote procedure call to an application server which computes the cartogram. The dot-net environment seems to be a promising tool. Further information about our *CartoDraw*-System, including downloadable maps and data are available from `http://cartodraw.org`. Future developments will also be posted to this site.

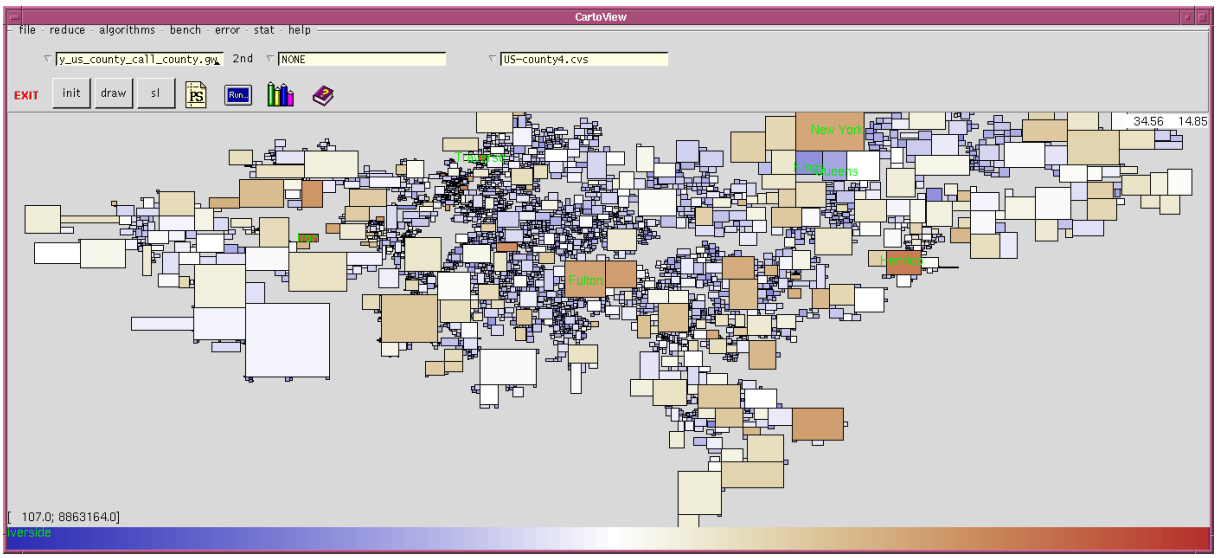
[1] Since all algorithms can be run from the command prompt (see chapter C) we can use any GUI environment, e.g., TCL/TK[43]



(a) Color setting



(b) *RecMap* settings



(c) *CartoDraw*-System – main window

Figure 6.11: *CartoDraw*–System: The figure displays the *CartoDraw*-system in action. Figure 6.11(c) shows a rectangular cartogram where the area is proportional to the cumulated call volume of an U.S. phone company. The color indicates the proportion between customers and call volume. In white areas the call volume is proportional to the number of customers. Blue area indicates unexhausted regions while red regions highlights region where the call volume is extraordinary high. The visualization shows directly the locations of interest which can never be achieved by a regular visualization such as a quantile-quantile-plot. The usability can be enhanced by standard labeling of interesting regions or by using specialized color mappings. For a more efficient data analysis the geo-visualization in the window of sub-figure 6.11(c) can also be combined with classical information visualization techniques such as scatter plots (including q-q-plots), parallel coordinates and interaction techniques such as zooming, linking, and brushing.

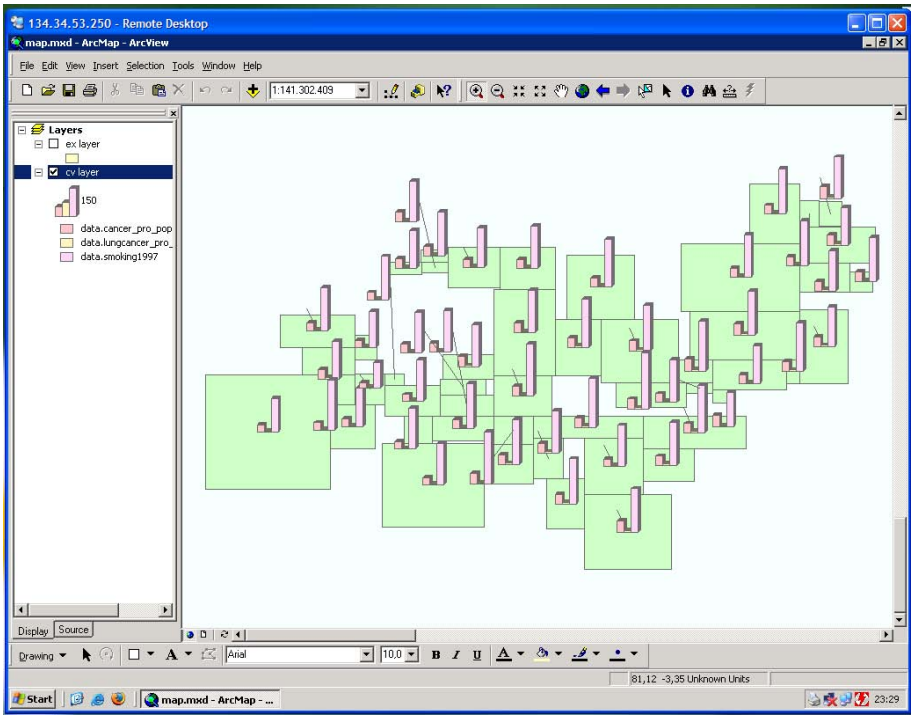


Figure 6.12: ESRI ArcMap plugin using *RecMap* and the U.S. population data (thanks to Tobias Müller for the implementation)

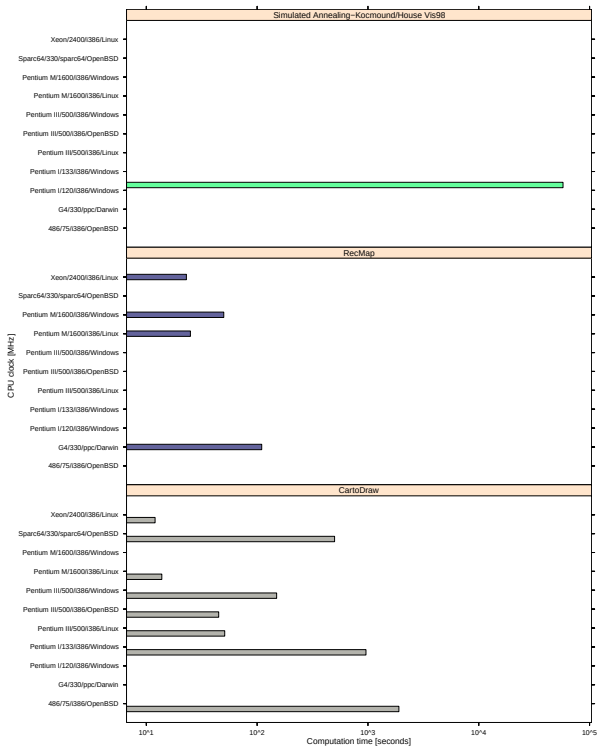


Figure 6.13: The graph plots the run time for several cartogram algorithms on various architectures. Even on a 486 CPU (75MHz clocked) *CartoDraw* it is two magnitudes faster the the best-known approach (upper panel) running on a i586 CPU (120MHz clocked).

7 Evaluation and Application

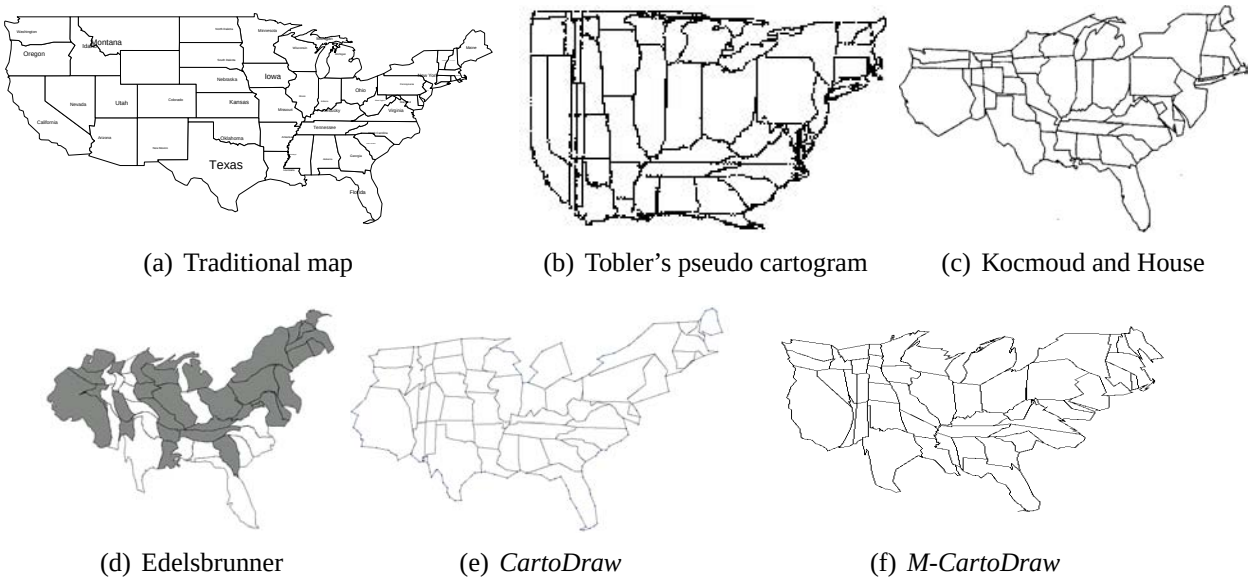
The algorithms described in the previous chapters have been implemented in *ANSI-C* [89] and *C++*using the LEDA library [97] and were run on a number of different example applications.

In this section, we report and discuss the results, the quality, the effectiveness and efficiency of the different approaches.

Although our focus is on efficiency, the examples show that our proposed algorithms *CartoDraw*, *M-CartoDraw*, *HistoScale*, and *RecMap* also provide results of very high quality.

In some cases checker board examples are used. We continue to use a state map of the continental U.S. and the population data for the comparisons.

7.1 Comparison of the Quality with Previous Methods



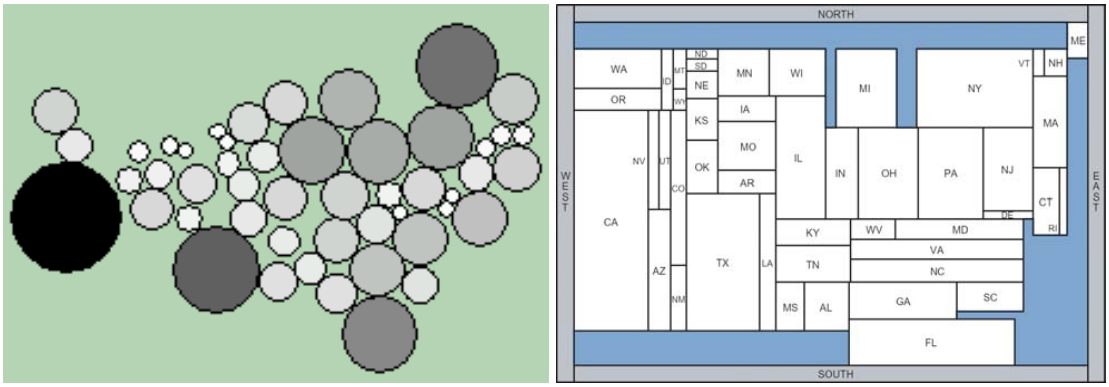
(a) Traditional map (b) Tobler's pseudo cartogram (c) Kocmoud and House

(d) Edelsbrunner (e) *CartoDraw* (f) *M-CartoDraw*

Figure 7.1: Comparison with related contiguous cartogram drawing methods.

Figure 7.1 and 7.2 show U.S. population cartograms generated by our algorithm and by the techniques proposed by Tobler [121], by Kocmoud and House [90], by Dorling [30], and van Kreveld and Speckmann [126]. A visual comparison shows that our approach offers comparable if not better visual results, with the geography of the United States being clearly perceivable.

For the *RecMap* approach we can use an additional quality measure. Since each polygon is represented by a rectangle the question arises how good the polygon region can be approximated. Two possible layouts are presented in figure 7.3.

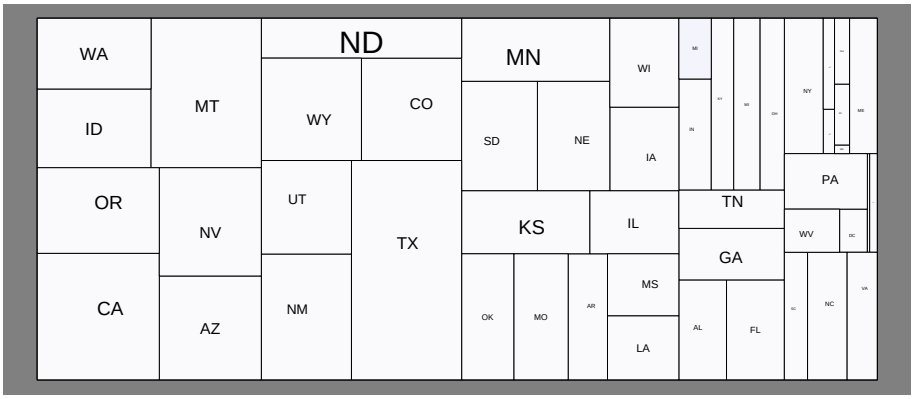


(a) Circle cartogram (b) RecCarto

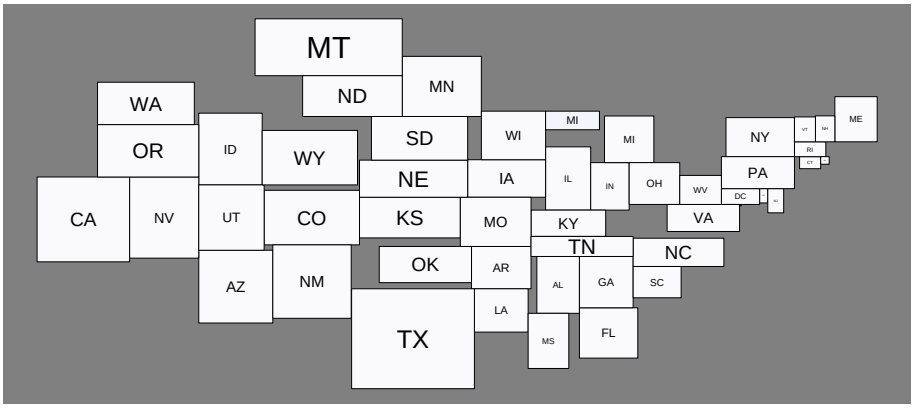
(c) *RecMap* (*MP*1) (d) *RecMap* (*MP*2)

Figure 7.2: Comparison with related cartogram drawing methods.

(a) U.S. map



(b) *MP*1



(c) *MP*2

Figure 7.3: *RecMap* the map regions were approximated by rectangles. The area of each rectangle complies to the area of the original polygon.

simulated annealing	$O(n^2)$
M-CartoDraw	$O(m \cdot \log m + m \cdot n)$
RecMap-MP1	$O(p^2)$
RecMap-MP2	$O(p^3)$
HistoScale	$O(n)$

Table 7.1: Time complexity of introduced cartogram methods with number of map nodes $n = |V|$, number of nodes of the global polygon $m = |GP(\mathcal{P})|$, number of polygons $p = |\mathcal{P}|$

7.2 Overall Effectiveness and Efficiency Comparison

Table 7.1 shows an overview of the time complexities of the introduced cartogram methods. Run time comparison for some input examples can bee seen in table 7.2. To evaluate the results analytically, figure 7.4 shows the total area error d_A for all approaches. Figure 7.4 shows that our proposal provides even better results than the complex optimization-based approach by Kocmoud and House [90].

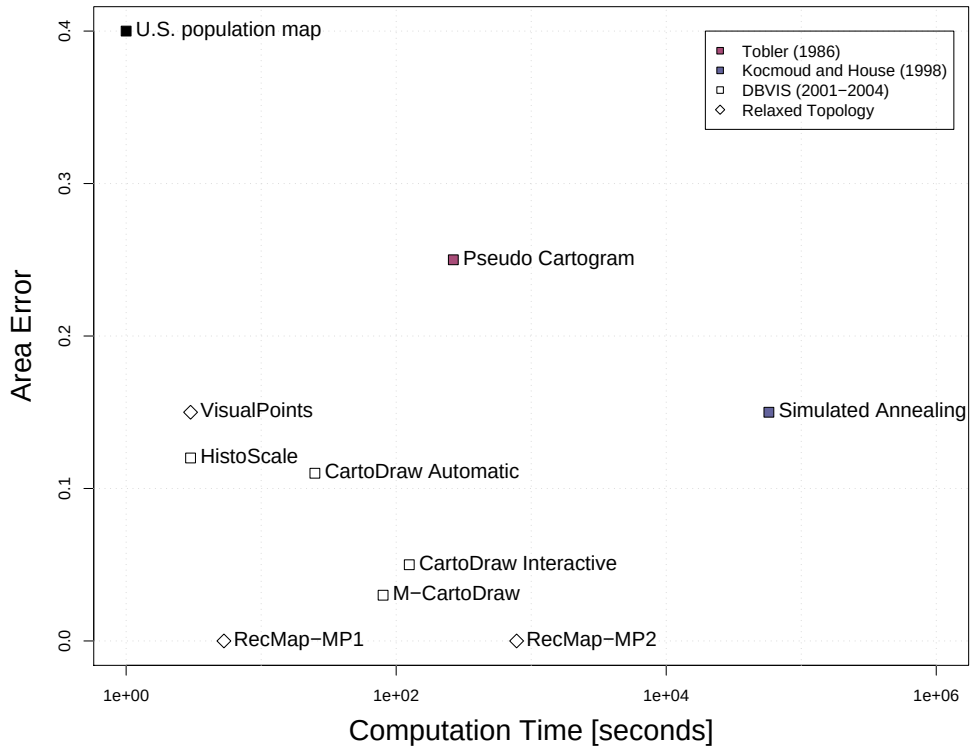


Figure 7.4: The comparison of all cartogram drawing methods assumes an U.S. state level map, the U.S. population data and an i386 120 MHz clocked CPU.

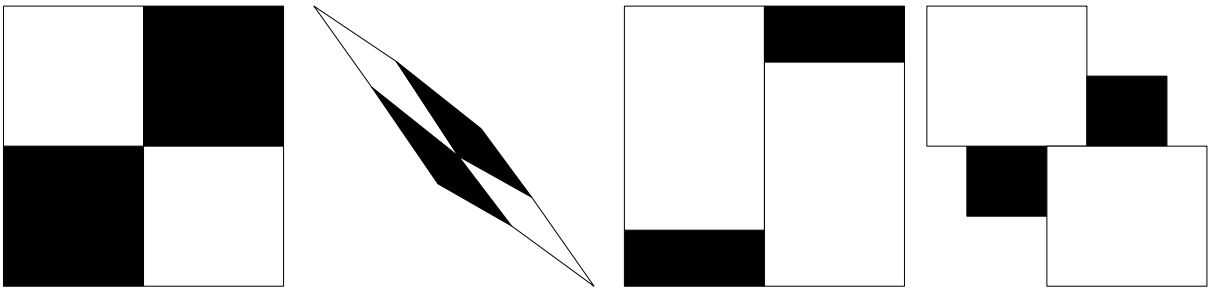
In terms of efficiency, our approaches are much faster than existing techniques. While previous approaches need hours or even days to compute a solution, our implementations runs in a matter of seconds. Figure 7.4 shows that our scanline-based heuristic needs about 25 seconds while the Kocmoud & House approach needs about 16 hours, making our approach about 2000 times faster. Please note that the time axis on the plot is in logarithmic scale.

Furthermore, figure 7.4 demonstrates the trade-off between the area constraint and the computation time.

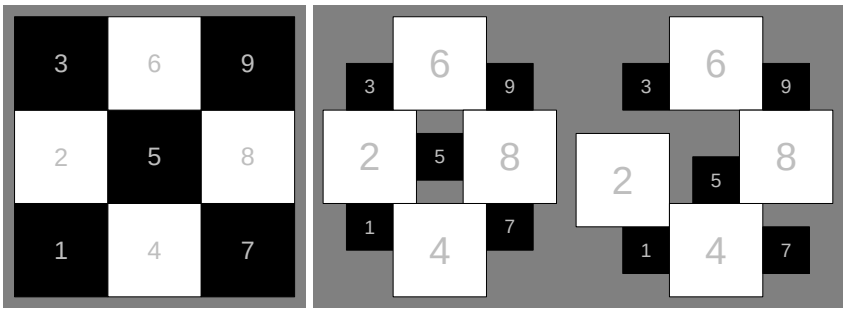
map	algorithm	$\|V\|$	$\|E\|$	$\|\mathcal{P}\|$	$\|GP(\mathcal{P})\|$	$d_A(\mathcal{X},\mathcal{P})$	$d_A(\mathcal{X},\overline{\mathcal{P}})$	time(secs)
US-state	*M-CartoDraw*	808	1286	49	429	0.38	0.05	16
US-county	*M-CartoDraw*	7609	20901	3085	474	0.63	0.33	223
NY-county	*M-CartoDraw*	204	443	62	88	0.67	0.05	3
Germany	*M-CartoDraw*	3510	6010	433	1877	0.55	0.23	466
US-state	*RecMap*-MP1	-	-	49	-	0.38	0	$\ll 1$
US-county	*RecMap*-MP1	-	-	3085	-	0.63	0	NA
CA-county	*RecMap*-MP1	-	-	59	-	0.63	0	$\ll 1$
TX-county	*RecMap*-MP1	-	-	255	-	0.66	0	10
7x7 board	*RecMap*-MP1	-	-	49	-	0.28	0	$\ll 1$
US-state	*RecMap*-MP2	-	-	49	-	0.38	0	50
US-county	*RecMap*-MP2	-	-	3085	-	0.63	0	$\approx 48 \cdot 3600$
CA-county	*RecMap*-MP2	-	-	59	-	0.63	0	180
TX-county	*RecMap*-MP2	-	-	255	-	0.66	0	240
7x7 board	*RecMap*-MP2	-	-	49	-	0.28	0	40

Table 7.2: Run time of computed cartograms

7.3 Application of Self-Generated Data



(a) 2×2 checkerboard (b) Preserved topology (*CartoDraw*) (c) No empty space (*RecMap MP*1) and preserved shape (*RecMap MP*2)



(d) 3×3 checkerboard (e) Preserved shape (*RecMap MP*2)

Figure 7.5: Checker board examples for different construction procedures.

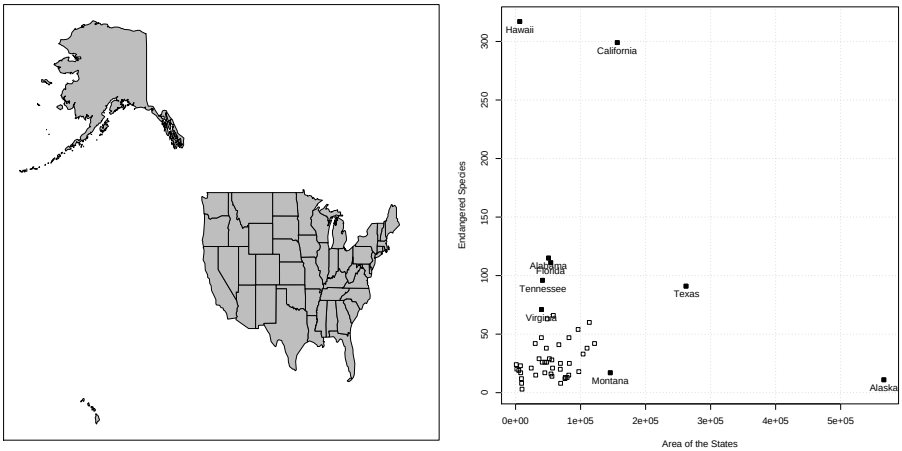
Figure 7.5 shows the result when we use self-generated checker boards as discussed in chapter 3. In this thesis we introduced three different solution procedures. Hence, we have a wide range of possible configurations. On all maps in figure 7.5 the size of the black regions has to be one area unit and the size of the white ones has to be four area units. On the input maps in sub figures 7.5(a) and 7.5(d) the areas are proportional to two units. Figure 7.5(b) shows a topology-preserving solution using our *M-CartoDraw* algorithm and the 2×2 checker board as input. On the map the medial axes of the global polygon are the

diagonals of the quadrat. Please note that the *CartoDraw* can not distort the mesh so that the area is equal to the desired area. This intention would be aimless since we do not allow to insert additional points on the polygon lines. *RecMap* approximates polygons by rectangles so that the *area error* of the map regions is avoided completely. Two different constraint configurations can be seen in figure 7.5(c). The first one avoids empty space where the second one preserve the shape of the regions. A regular 3 checker board is difficult to handle using *CartoDraw* since the scanline function takes the amount of the parameters of a given sized sector as factor for the transformation. In contrast, *RecMap* works well. Figure 7.5(e) shows two possible layouts for the 3×3 checker board map. Both layouts are shape-preserving. The different layouts can be explained by different parameter setting for the topology and empty space weights.

7.4 Application – Geographic Related Data

We ran the algorithms introduced in this thesis on a number of different maps and parameter vectors.

7.4.1 Environmental and Health Data



(a) Traditional map of the U.S. including Alaska and the Hawaii islands

(b) A X-Y-Plot shows the area of each state versus the number of endangered species

(c) The cartogram has been distorted using our *M-CartoDraw* algorithm. The area on that visualization corresponds to the number of endangered species in each state.

Figure 7.6: The graphic displays the U.S. proportion of endangered species using *M-CartoDraw* (data: courtesy of Environmental Defense Fund).

Figure 7.7: World SARS pseudo-cartogram – On the figure the area corresponds to the number of cases. The gray color indicates countries with SARS cases. The pseudo-cartogram is generated using *HistoScale*.

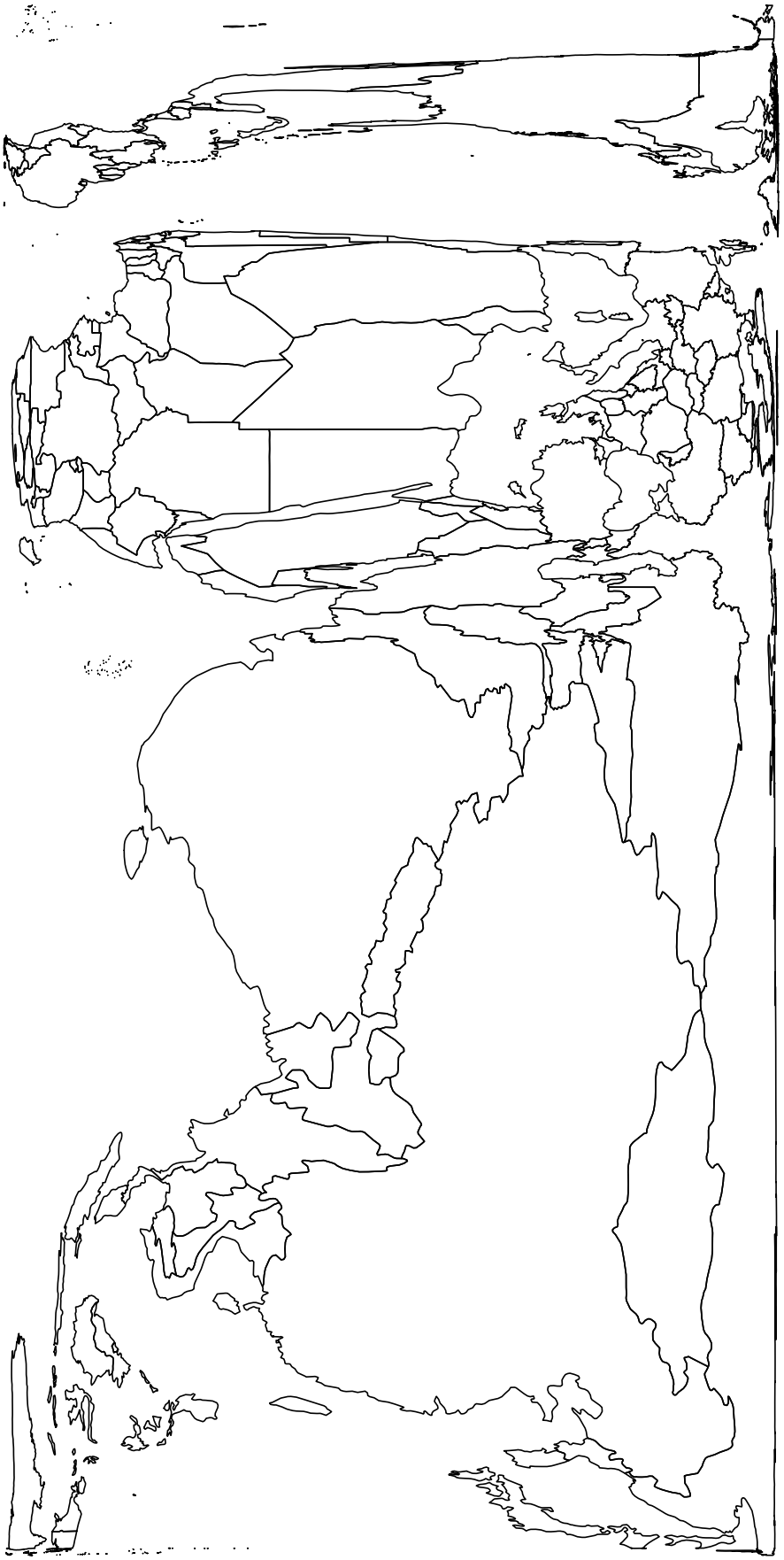


Figure 7.8: World population pseudo-cartogram (*HistoScale*)

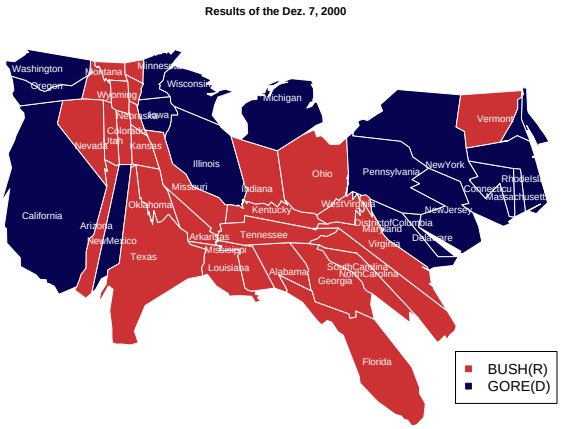
7.4.2 U.S. Election Cartograms

Election data are another application for cartograms.

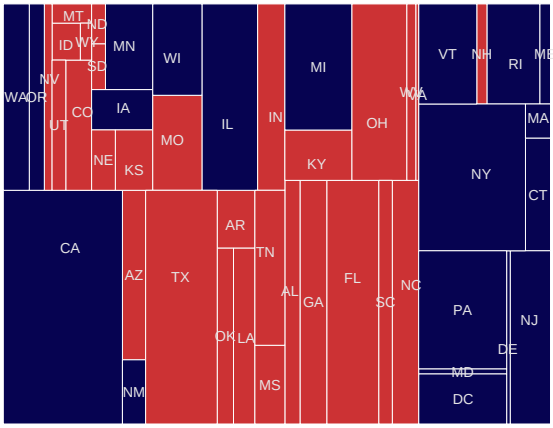
2000 – Bush versus Gore

The maps in figure 7.9 display visualizations of the presidential race in the year 2000. The areas in Figures 7.9(b), 7.9(c), and 7.9(d) correspond to the electoral voters. The red and blue color depict which candidate has won each state. The candidate who covers the most area of the map in figures 7.9(b), 7.9(c), and 7.9(d) has won the vote.

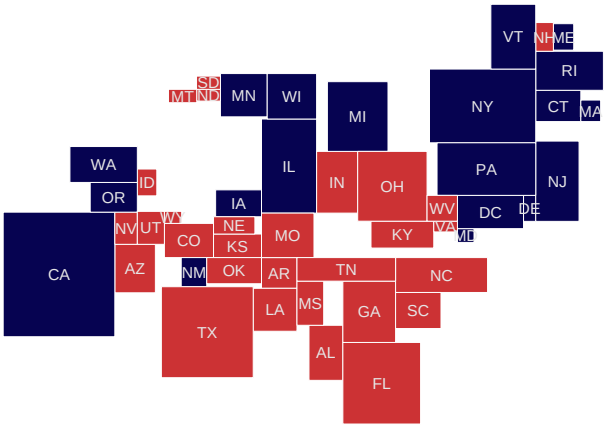
(a) Traditional map which can be archived from `http://www.nytimes.com/specials/election2000/results-pres.html`



(b) *M-CartoDraw*



(c) *RecMap MP*1



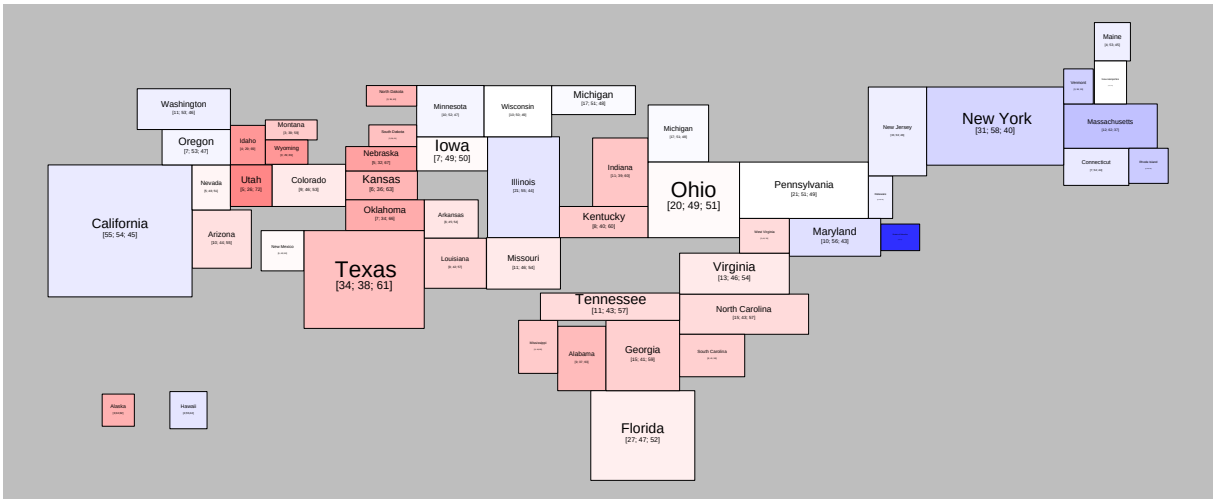
(d) *RecMap MP*2

Figure 7.9: U.S. election 2000 analysis (area: #electors; red: Bush; blue: Gore)

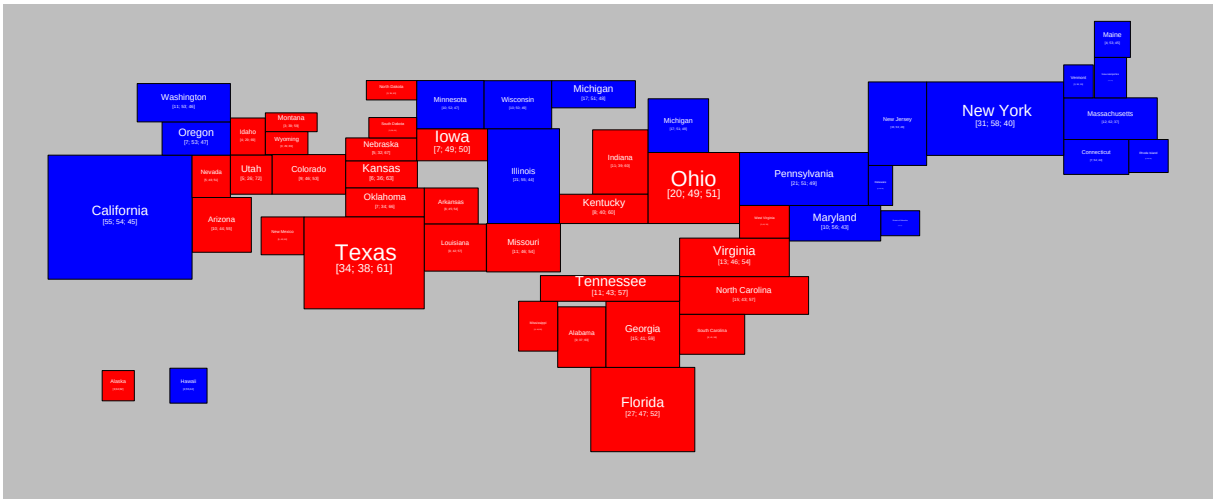
2004 – Bush versus Kerry

Figures 7.10, 7.11, 7.12, 7.13, 7.14, and 7.15 are visualizations of the 44th presidential election of the United States. On the value-by-area cartograms 50 states and 3085 continental counties, respectively, were approximated by rectangles. The aspect ratio of each county is the same as on the traditional map. The area of each rectangle corresponds to the number of population. The blue (Kerry) and red (Bush) colors show which candidate got the majority of each county. The brightness indicates the shortage of the vote. The visualization demonstrates clearly that the election was a head-to-head race, the county

level cartograms illustrate the election behavior between high populated cities and low populated area region. The source of the election data is `http://www.personal.psu.edu/users/a/c/acr181/2004_Election.zip`. The input U.S. maps are from `http://www.census.gov/`.



(a) *MP*1



(b) *MP*2

Figure 7.10: U.S. 2004 election analysis using *RecMap MP*2 on state level (area: #electors; red: Bush; blue: Kerry)

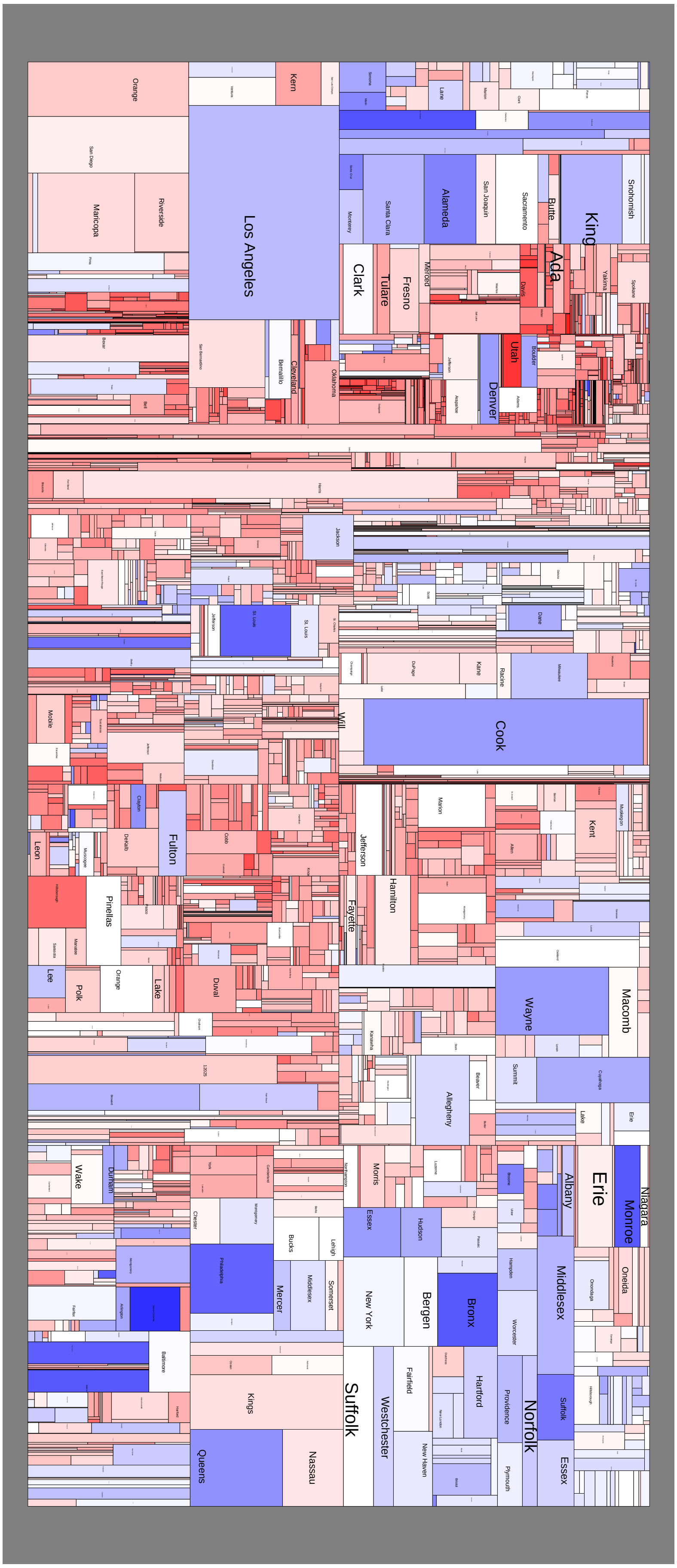


Figure 7.11: U.S. 2004 election analysis using *RecMap MP1* on county level (area: #electors; red: Bush; blue: Kerry)

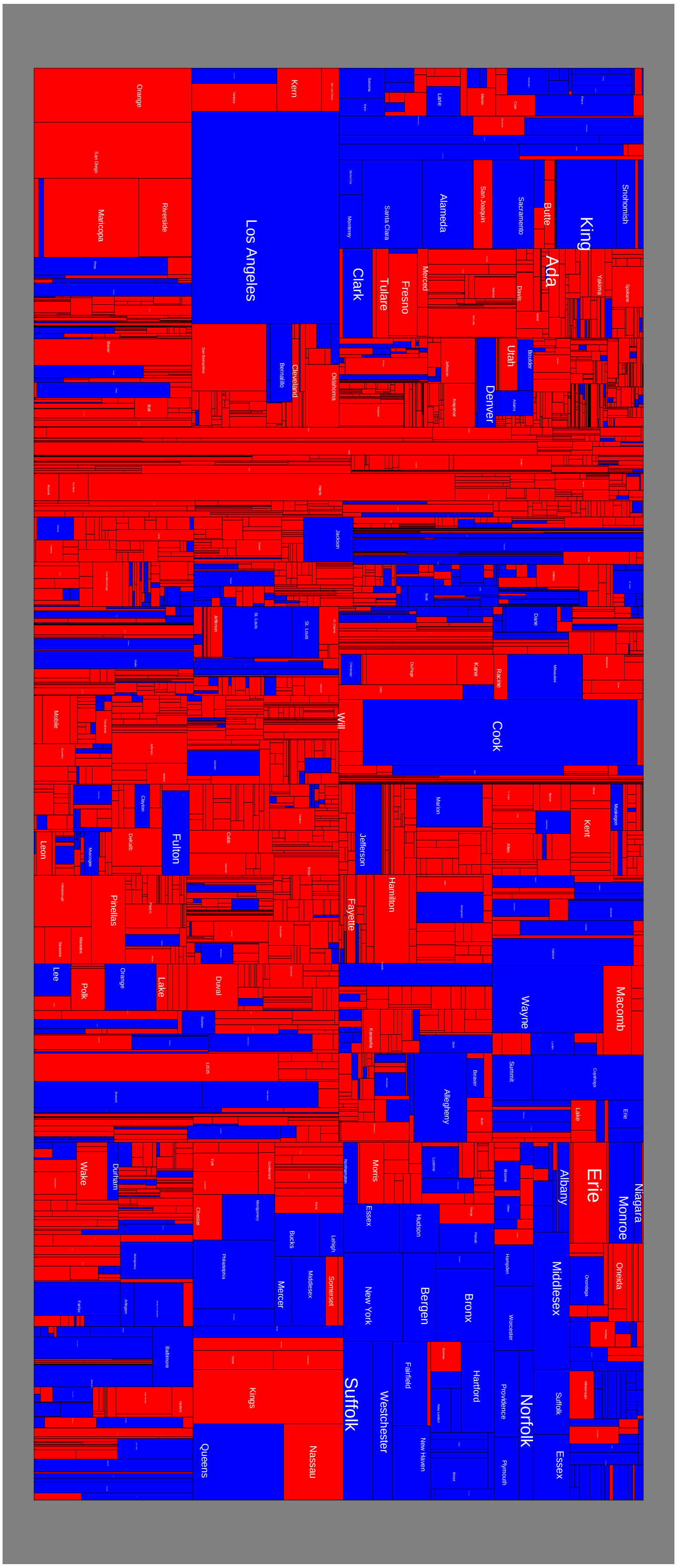


Figure 7.12: *RecMap MP1* – U.S. election 2004 analysis (area: #electors; red: Bush; blue: Kerry)

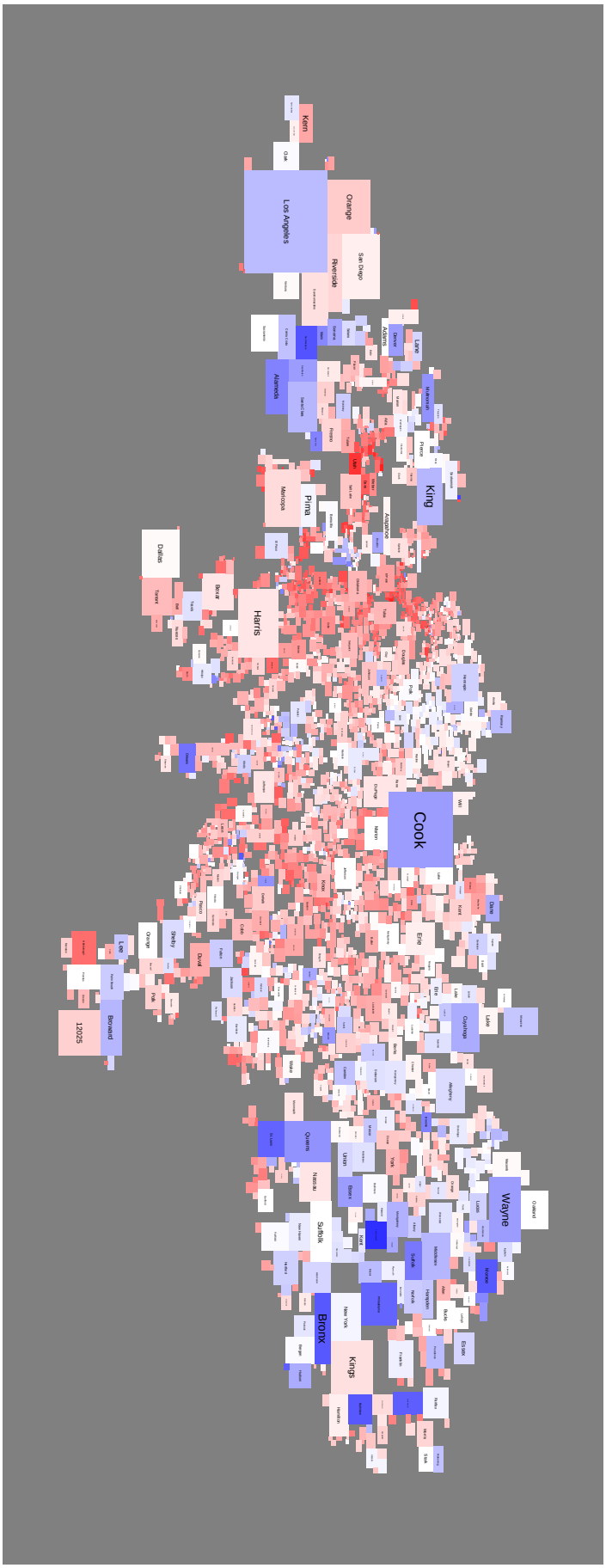


Figure 7.13: *RecMap MP2* – U.S. election 2004 analysis (area: #electors; red: Bush; blue: Kerry)

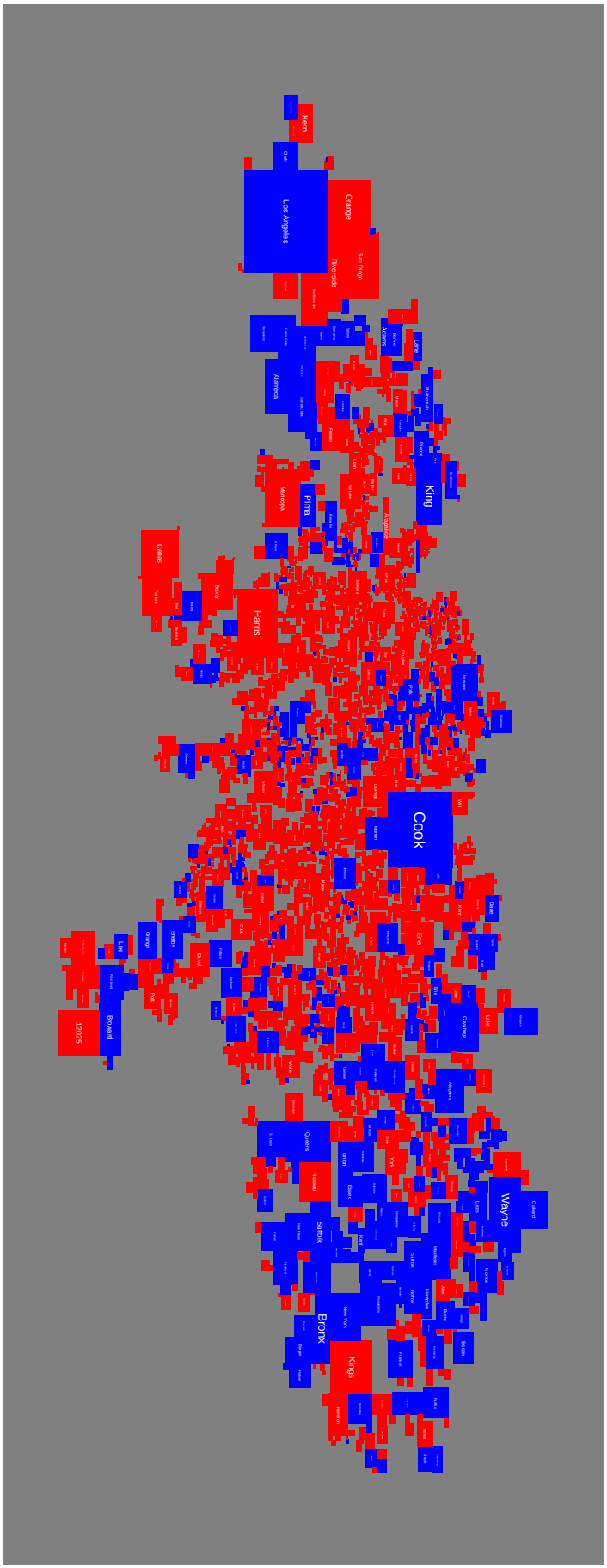


Figure 7.14: *RecMap MP2* – U.S. election 2004 analysis (area: #electors; red: Bush; blue: Kerry)

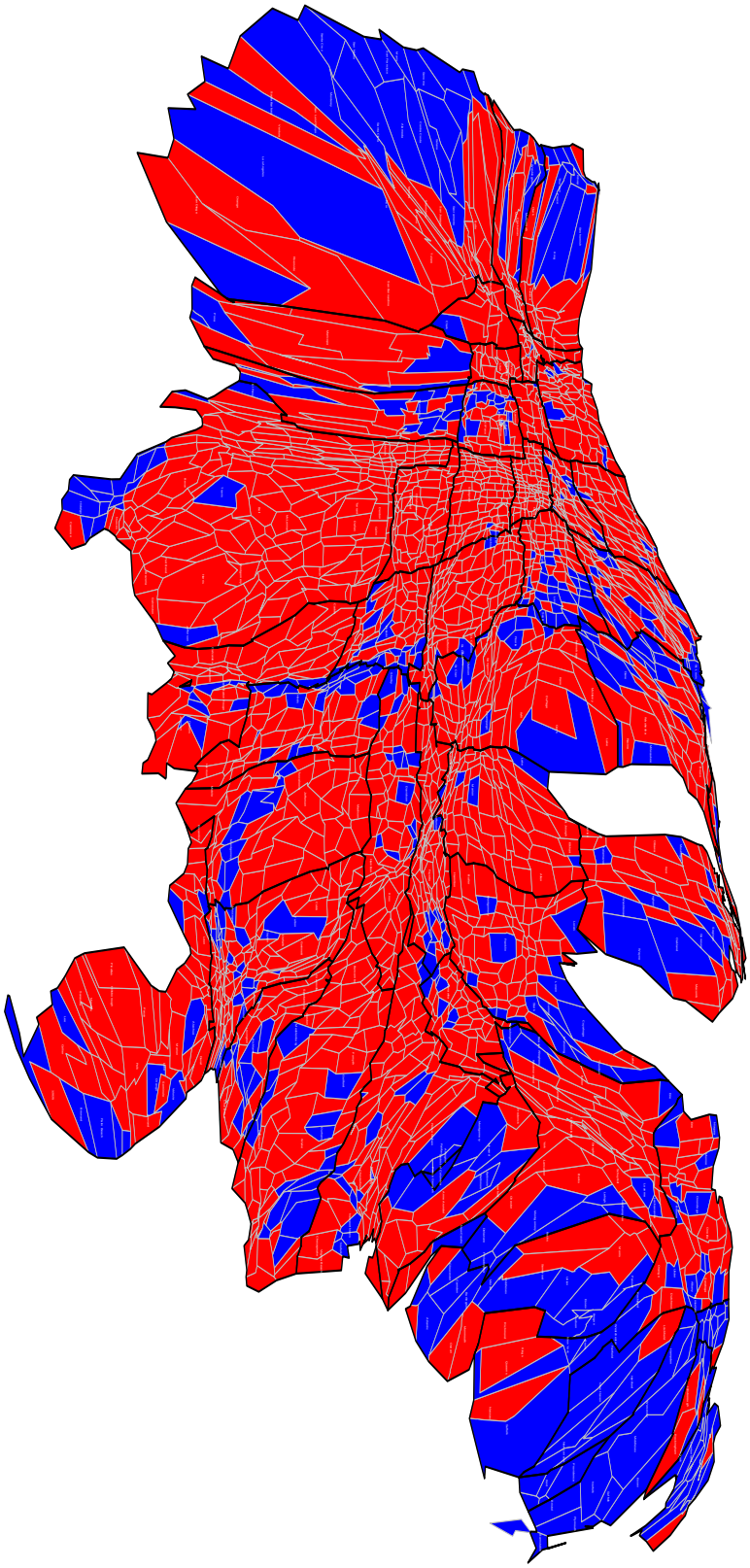


Figure 7.15: U.S. 2004 election analysis using *CartoDraw* on county level (area: #electors; red: Bush; blue: Kerry)

7.4.3 Visualizing Sequences – AT&T Call Volume Analysis

(a) 0:00 am (EST) (b) 6:00am (EST) (c) 12:00pm (EST) (d) 6:00pm (EST)

Figure 7.16: Analyzing long distance call volume data using *CartoDraw* ($d_A \leq 0.20$)

As mentioned at the outset, we developed the *CartoDraw* algorithm for displaying telephone call volume data, which needs to be continuously monitored and visualized. Since the underlying polygon data does not change, the vertex reduction needs to be run only once. This provides a significant speed-up, making interactive display with an update rate of about one second feasible. Figure 7.16 shows the results of the normalized telephone call volume at different times during one day. The resulting visualizations clearly reflect the different time zones of the U.S., and show interesting patterns of phone usage as it proceeds during the day.

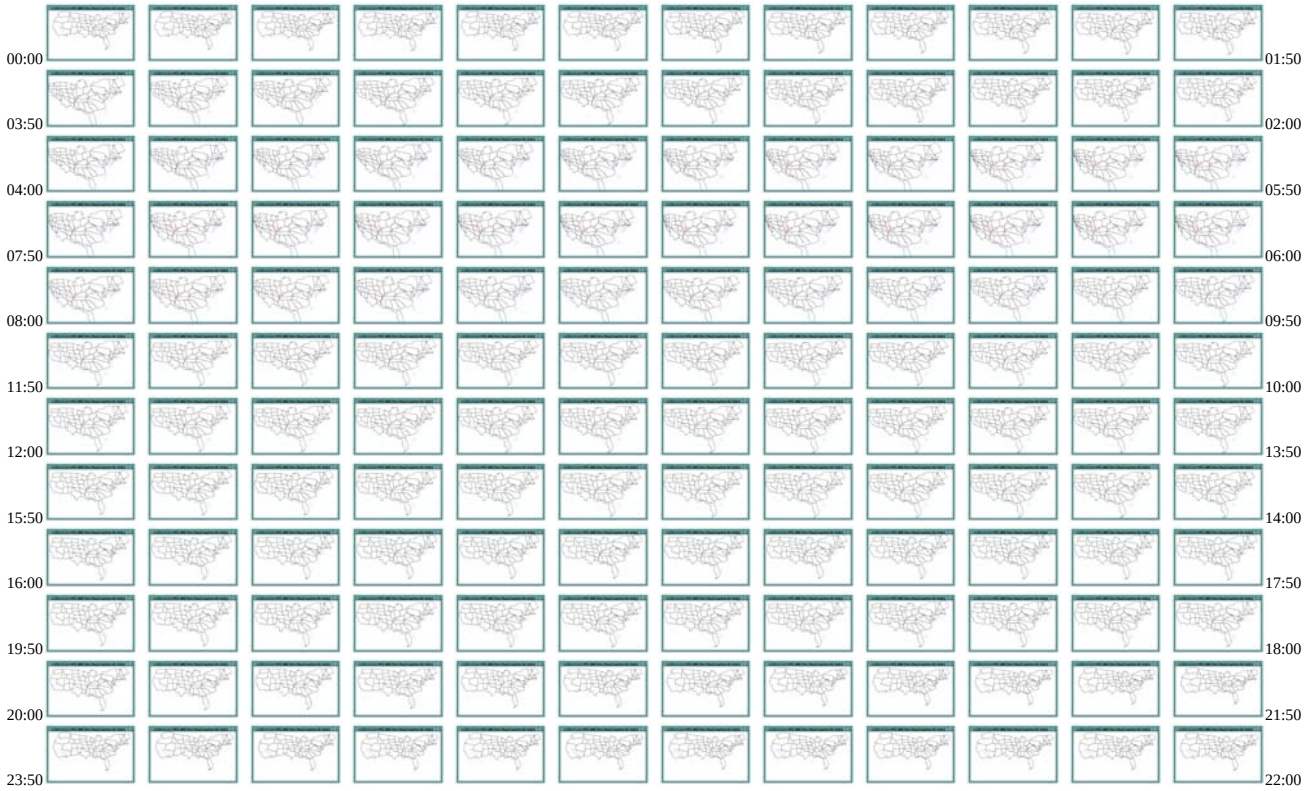


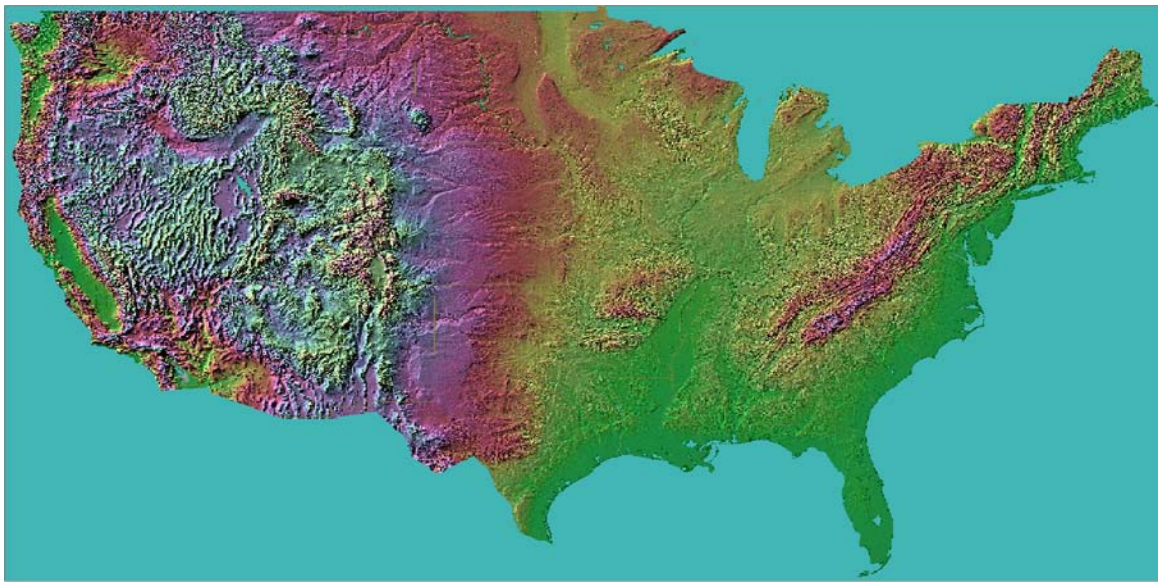
Figure 7.17: The graphics illustrate 24 hours call volume analysis in a mirrowed S-curve starting in the upper left corner.

On a dynamic map things that move probably attract more than things that do not [95, page 280]. Based on this concept figure 7.17 displays 144 time frames during one single day, one time frame every ten minutes. As already mentioned in chapter 4.4.2 on page 43, for this sequence we have used bilinear interpolation inside the *CartoDraw* algorithm to place the inner nodes of the mesh. As a result of this features, the transitions between two frames are very smooth. The video, the *dynamic map*, is accessible from the *CartoDraw* web site [76]. On the background of the video the medial axes of the global polygon

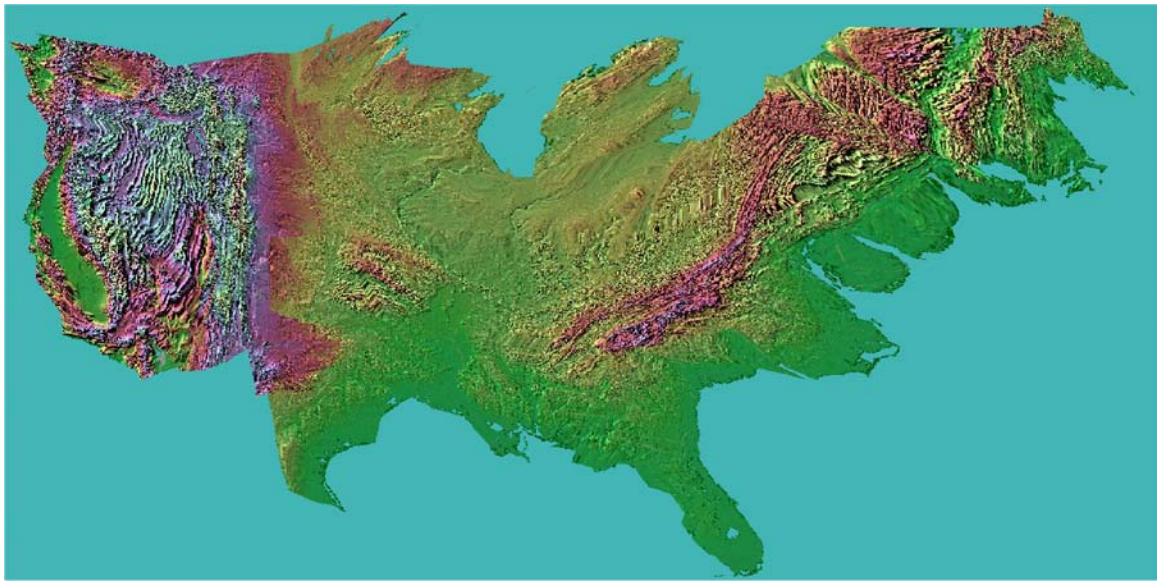
are drawn and colored according the *area error* of each region. The direction of the transformation is aligned to the color parameter of this skeleton to form the cartogram.

7.4.4 Texture Mapping Cartograms

Figure 7.18 is another example in which we apply texture maps to visualize geography-related data. Figure 7.18(a) shows an undistorted map of the U.S. relief. Figure 7.18(b) illustrates the mesh distortion from *M-CartoDraw*. On that map the relief is distorted in relation to proportion of the number of inhabitants.



(a) Original map

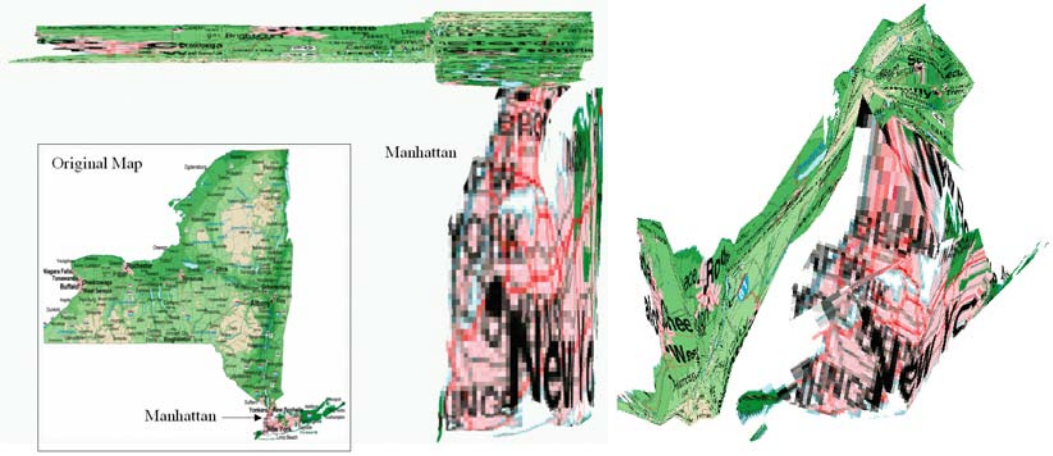


(b) From *M-CartoDraw* distorted map

Figure 7.18: U.S. population cartogram with texture (source of the texture: `http://fermi.jhuapl.edu`, January 9, 2003)

The relief of the continental U.S. is stretched where huge amounts of people live, e.g. coast lines, and it is shrunk e.g. on the mountain areas. The stretching and shrinking of the map regions can be more stressed on the construction movie[76].

Figures 7.19(a) and 7.19(b) show a texture cartogram of New York state (U.S.), using an almost zero *area error* cartogram transformation. The cartogram emphasizes the proportion of New York City in the total population. Another candidate example would be a map transformation in which we distort the map according to a given route to highlight its most interesting features.[1] The graphics in figure 7.19(b) show two different kinds of distortions. In this case the basic continuous vertical and horizontal distortion of the *HistoScale* is of value in contrast to figure 7.19(b). The advantage is in the distribution of the data.



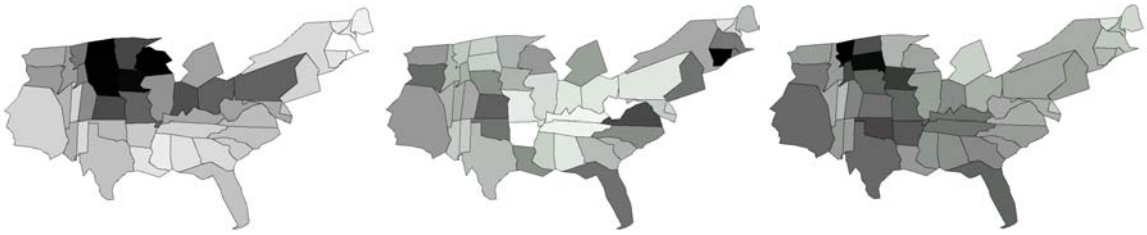
(a) *HistoScale* (b) *M-CartoDraw*

Figure 7.19: The graphics show two distorted map of NYC with almost zero area error.

[1]The quality of the *cartogram texture mapping* is restricted by the resolution of the given textures. Vector graphic data would overcome that problem. Since organizing vector data containing interesting material is very difficult or at least very expensive we are not able to apply *cartogram texture mapping* on them.

7.4.5 Population Cartograms

First, we used an U.S. population cartogram to show some statistic data obtained from the U.S. census database [124]. In figures 7.20(a) - 7.20(c), the color shows different statistical parameters on top of the population cartogram: Figure 7.20(a) shows the number of persons with German ancestry, figure 7.20(b) the median household income in 1989, and figure 7.20(c) the percentage of unpaid family workers. Since



(a) Persons with German inheritance (b) Median household income in 1989 (c) Unpaid Family Workers

Figure 7.20: The population cartogram shows census data statistics ($d_A = 0.10$) using the *CartoDraw* algorithm.

the area of the states corresponds to their population, the shaded areas in the figures directly correspond to the number of people with those properties. It is interesting that the highest percentage of people with German inheritance is in the northern Midwest (with some higher numbers also in Philadelphia and East). In the median income cartogram it is interesting that the areas with small income are basically all states with a rather small population (middle of the non-coastal east of the U.S.), and in the unpaid family workers cartogram, the numbers are high in the northern Midwest with some additional high numbers in Florida, Texas, and California. To show that our algorithm works with arbitrary polygon meshes, we also

Figure 7.21: Population cartogram of middle Europe (*CartoDraw*)

applied the algorithm to the population data of Europe, as shown in figure 7.21.

Figure 7.22 illustrates trends in U.S. population during the ten decades of the 20th century. Each decade is colored according to population increase or decrease. The colormap (figure 7.22(a)) can be interpreted as follows: dark blue - extremely large immigration, light blue - low immigration, white - no changes in population, and light red - low migration. For example, we can see the rapid decrease of the fraction of the population of the western states. Note that the area of the U.S. does not reflect the absolute number of inhabitants each year. Figure 7.22(l) accounts for this by scaling the whole polygon mesh according to the number of U.S. inhabitants. The global polygon of the U.S. in 2000 with 270 millions inhabitants is colored in black, and the global polygon of the population in 1900 with 76 millions is colored in grey.

A key goal for *M-CartoDraw* is to handle large maps. Now we want to use it for maps with a high number of polygons.

(a) Used color map

(b) 1910 (c) 1920 (d) 1930 (e) 1940 (f) 1950

(g) 1960 (h) 1970 (i) 1980 (j) 1990 (k) 2000

(l) Compare19002000

Figure 7.22: Population trends over the last 100 years. Each decade is colored according to population increasing or decreasing. The colormap (figure7.22(a)) can be interpreted as follow: dark blue-high rate of immigration, light blue-low immigration, white-no change, and light red-low emigration (*M-CartoDraw*)

Figure 7.23 shows the quantile plot of the U.S. census county population data set. The y-axis was scaled logarithmically. Furthermore, we used a color to link the statistical value to the maps. On the map we labeled the three highest populated counties and linked them to their data items. The resulting cartograms for the *M-CartoDraw* algorithm can be seen in figures 7.24 and 7.25. Figures 7.28 and 7.29 graph the results for the rectangular approach.

Often geo-related data are hierarchically structured e.g. on the U.S. map we have a state level, county level, track etc. Both county cartograms in figures 7.24 and 7.25 demonstrate two of three possible hierarchy layouts for contiguous cartograms.

On the U.S. county map all polygon lines are assigned with a five digit number. The first two digits indicate the U.S. states and the rest three digits indicate the counties inside each U.S. state. For example the county `08013` can be interpreted as follows: The first two digits stand for the U.S. state *Colorado* and the other tree digits stand for the county called *Boulder*. This description is generic and therefore it works for all hierarchies. The state boundaries can be achieved if we draw all segments of the map thicker which are adjacent to edges where the two first digits are different. Notices that each polygon is a closed line loop: Therefore, each to be drawn edges occurs twice and can be eliminated

Often the hierarchy order of the map can not be out-formed as clear as when we use the census map as input data. Therefore on the second approach the higher level map (in this case the U.S. state map)

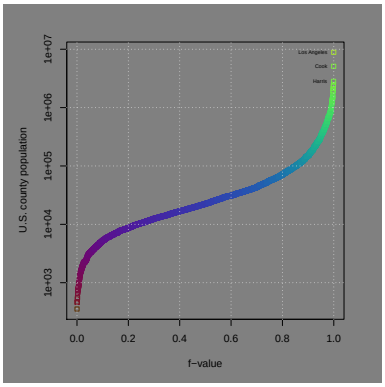


Figure 7.23: The graph shows a quantile plot of the U.S. census county level population data set. Each box on the graph represents one part of the quantile plot.

is pinned on the county mesh and we used the same cartogram transformation as for the county level map. The result can be seen in figure 7.25. The previously described technique was used for the Germany example in figure 7.27.

If both variants are not applicable, the *cartogram texture* technique described in 6.5 on 81 can be used.

Figure 7.26(c) shows a U.S. county level population cartogram. On that transformation only the state level borders were drawn. It can be clearly seen that finer granular statistical values have formed the shape of the new mesh (e.g. NYC region).

On all maps in figures 7.25, 7.24, 7.27, 7.29, 7.30, 7.31, 7.32, and 7.33 a logarithmic color mapping is used which can be computed as follows:

$$c_i = \frac{\log(\tilde{x}_i + 1) - \log(\min_i(\tilde{x}_i + 1))}{\log(\max_i(\tilde{x}_i + 1)) - \log(\min_i(\tilde{x}_i + 1))} \tag{7.1}$$

Figures 7.27(a) and 7.27(b) show the population of Germany on a map with about 400 polygons. We used the unipolar colormap shown in figure 7.27(a) to visualizes the population data. The cartogram helps to focus attention on highly populated regions. The area error of the three largest polygons as well as the area error of the yellow polygons in the northwest are almost zero; cities and other highly populated regions are clearly visible. Since the underlying polygons do not change dramatically in each step, especially when the algorithm is close to termination (see figure 4.19 on page 47), the medial axis is not recalculated on each iteration to save time. Note that *M-CartoDraw* took less than five minutes to compute the cartogram in figure 7.27(b) and less then ten minutes to compute the cartogram in figure 7.24, respectively. The computation time for this is only one magnitude higher than that needed to compute the U.S. state cartogram.

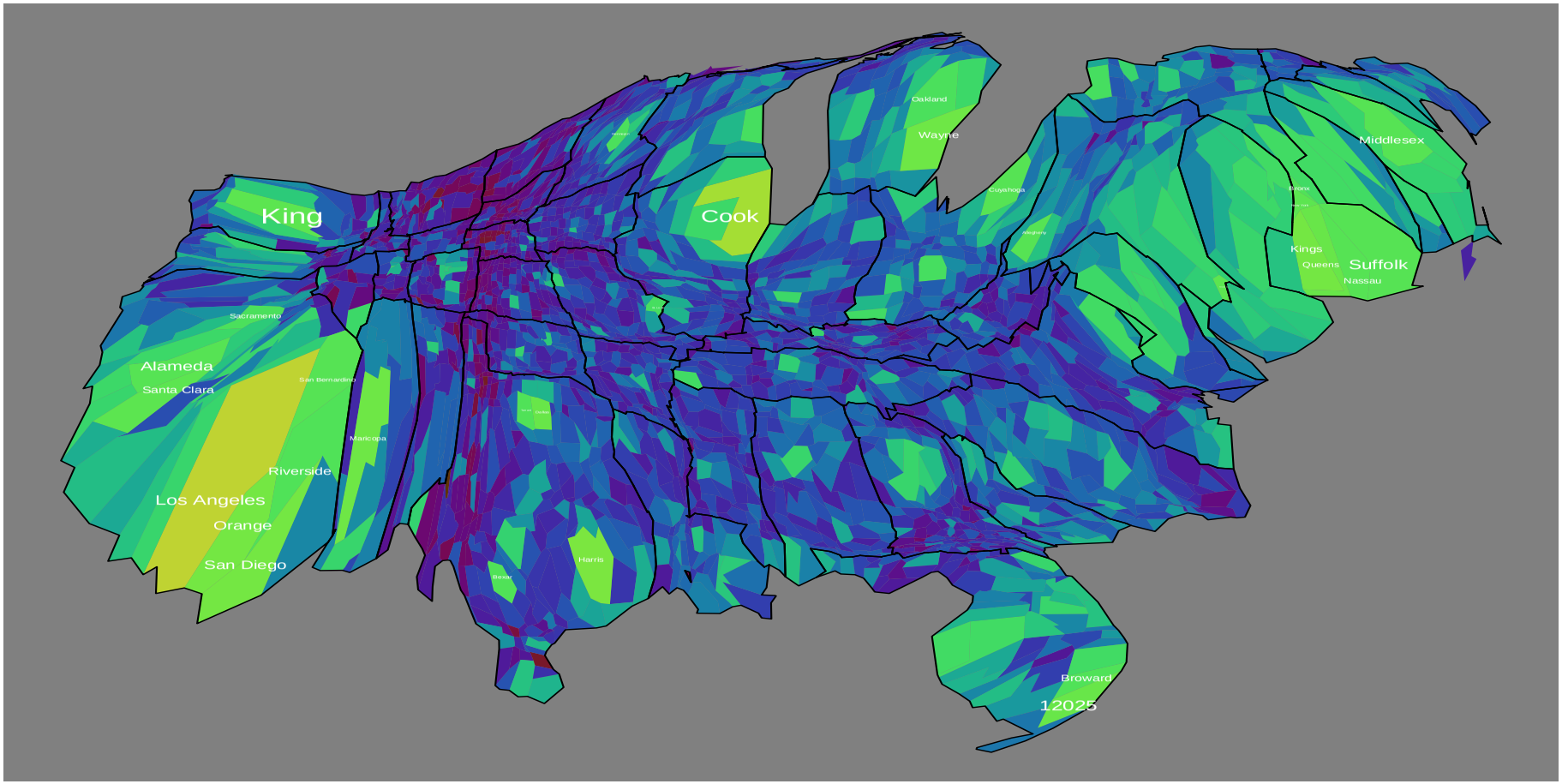


Figure 7.24: The picture is a visualization of the U.S. census 2000 county population cartogram with drawn state borders using the five digit representation.

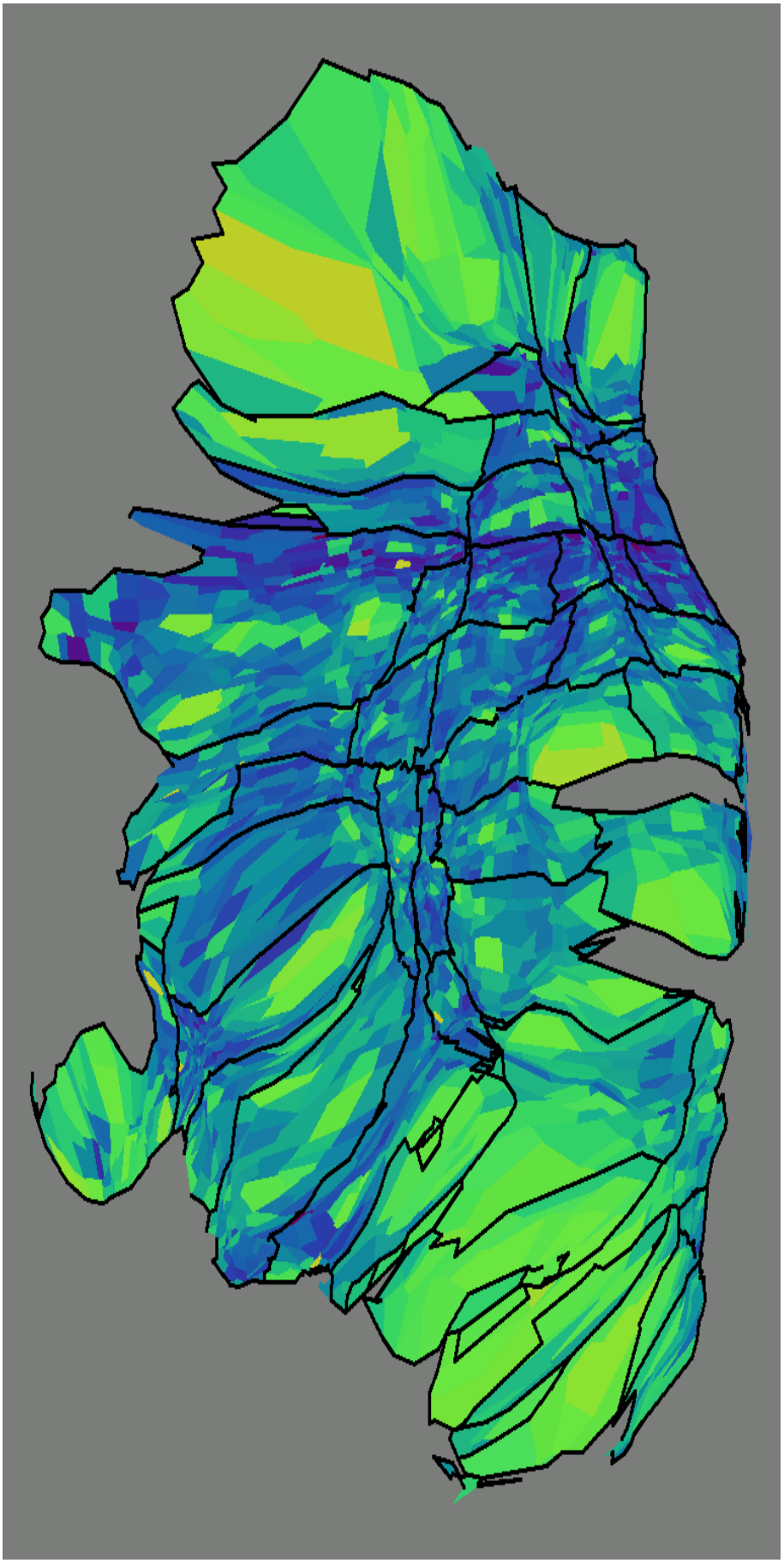
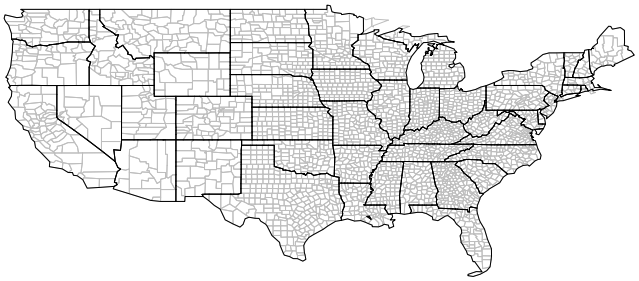
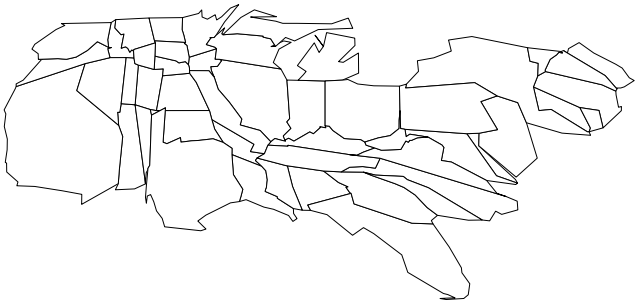


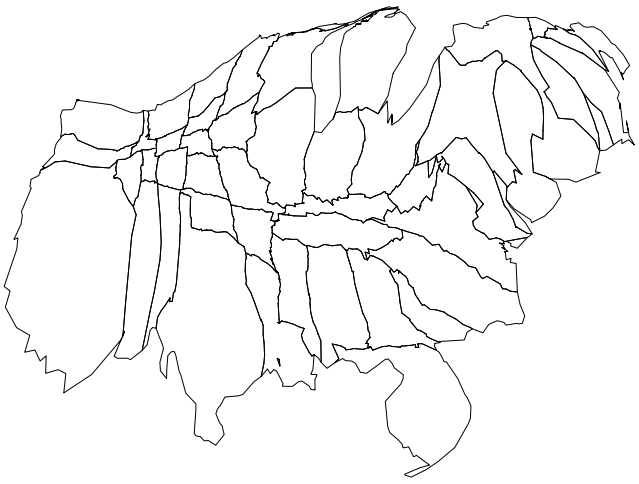
Figure 7.25: The figure shows another possibility of an U.S. county population cartogram with state borders. On that cartogram the state borders were pinned to the county map during the cartogram transformation.



(a) U.S. map

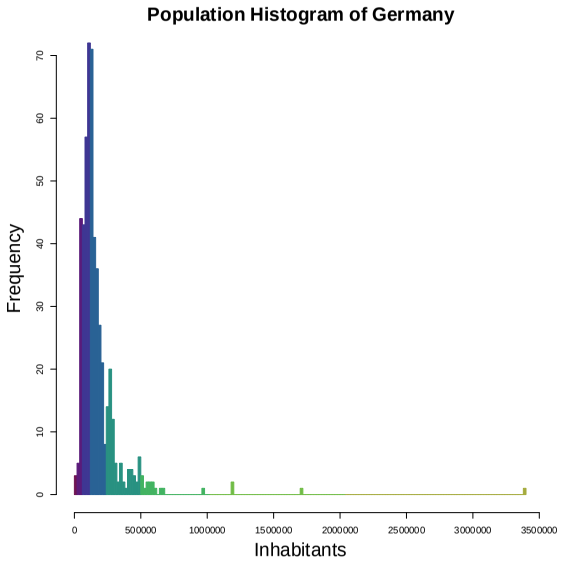


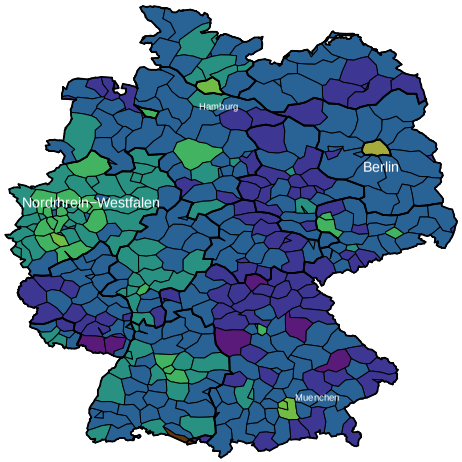
(b) State level cartogram



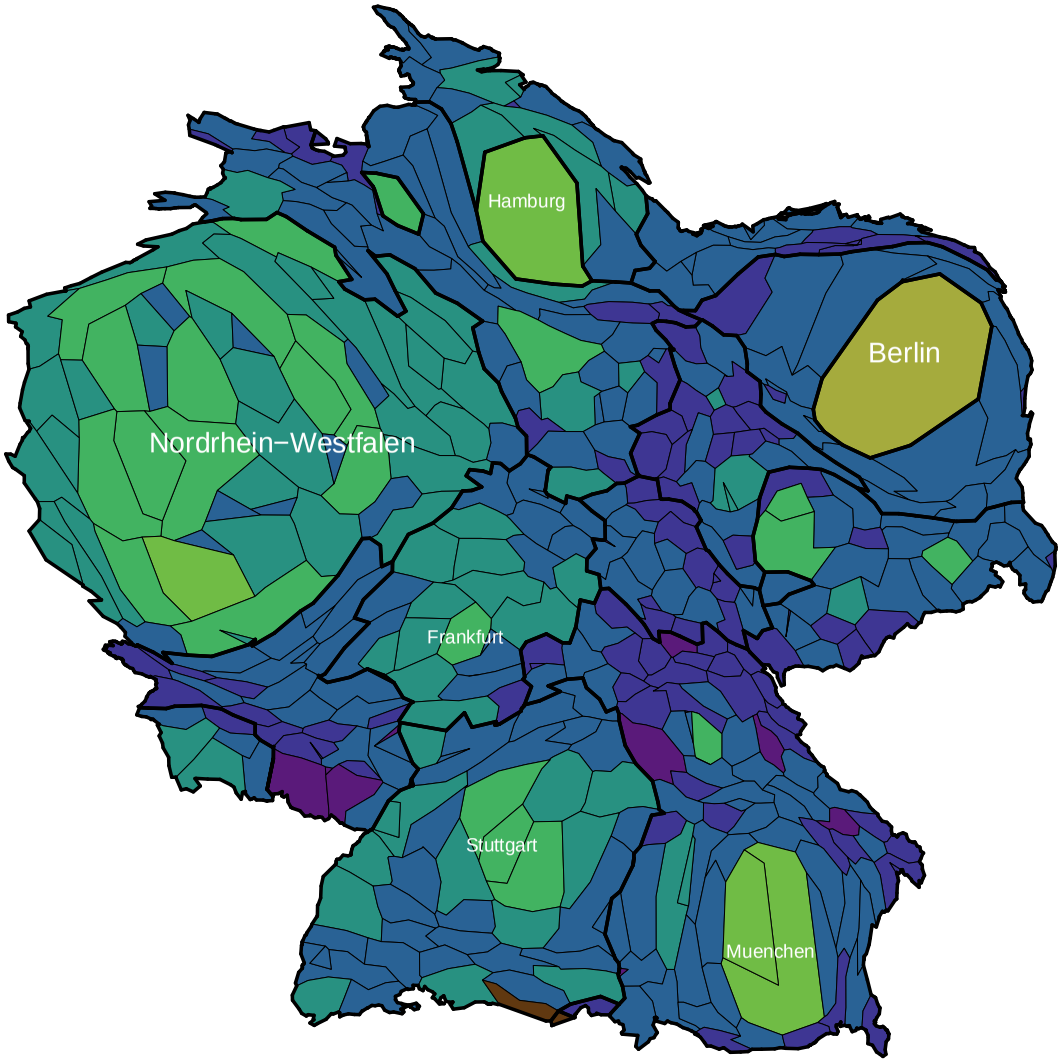
(c) County level cartogram

Figure 7.26: Cartogram on state and county level; only state poly lines were drawn.





(a) Traditional German map



(b) German population cartogram was drawn by *M-CartoDraw*

Figure 7.27: Figure 7.27(a) displays the traditional map of the German county level as well as a population histogram. A second layer (thicker black lines) indicates the German states. Figure 7.27(b) visualizes the same data as a Germany year 2000 population cartogram. Color linking is used to bridge all three pictures. On Figure 7.27(b) all high-populated regions (e. g. Berlin, Hamburg, Munich) are clearly visible. Data source: Verwaltungsgrenzen BRD, Bundesamt für Kartographie und Geodäsie.

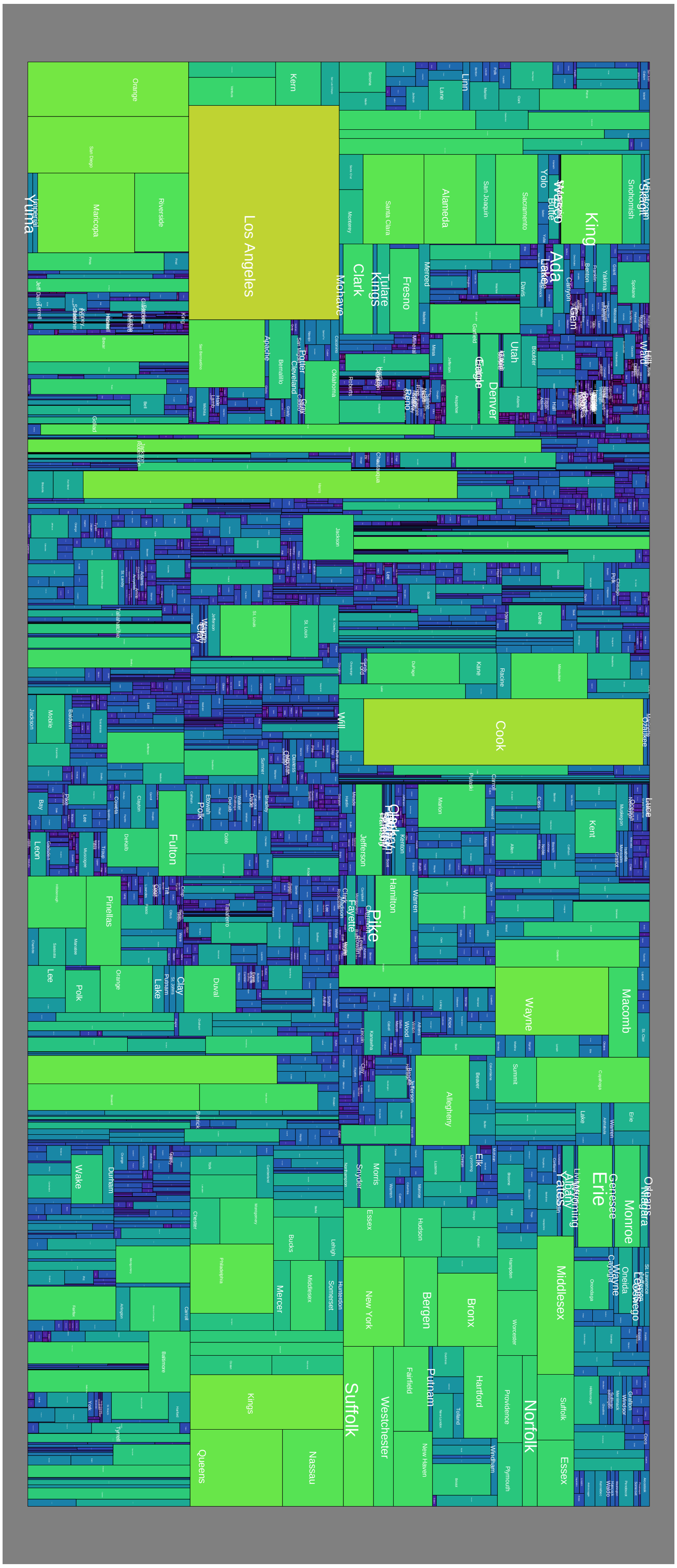


Figure 7.28: U.S. county population cartogram using *RecMap* (*MP1*)

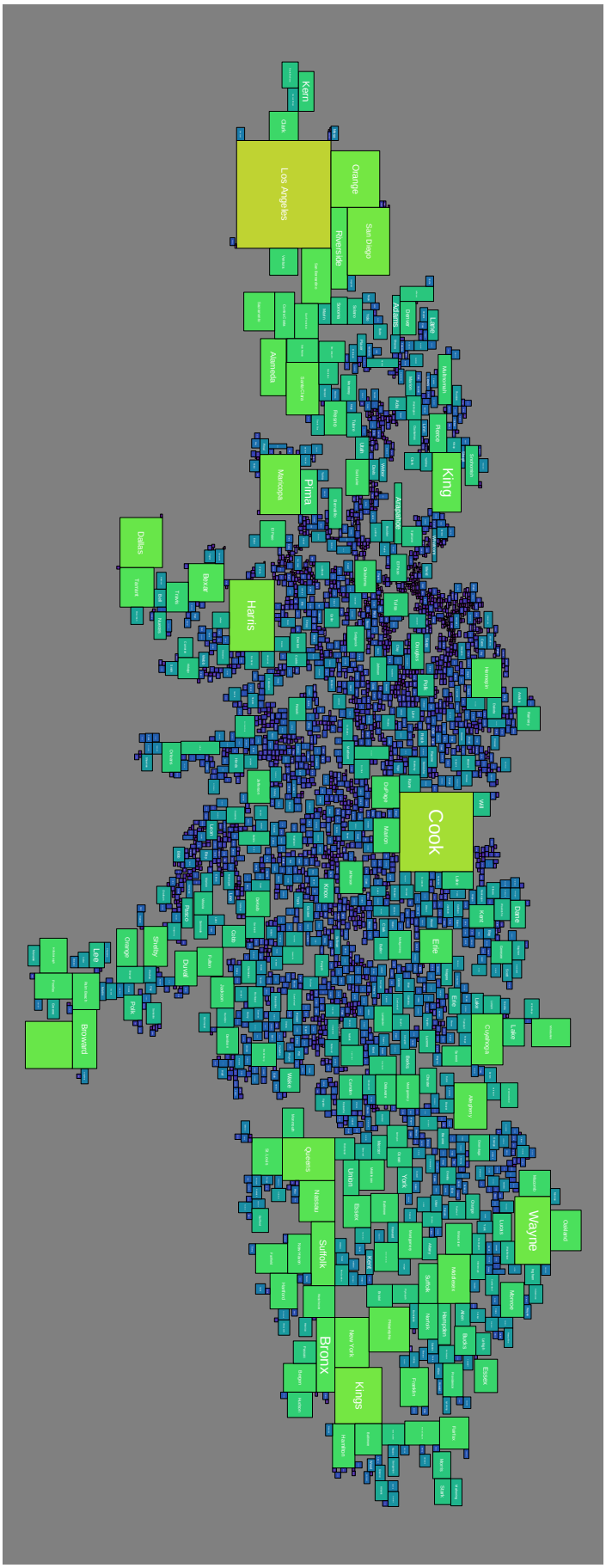
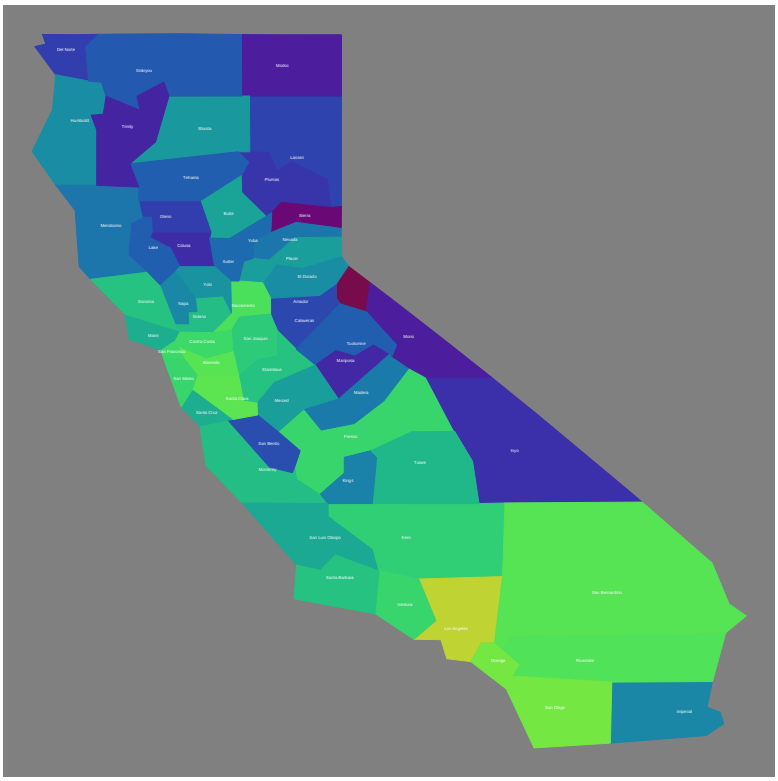
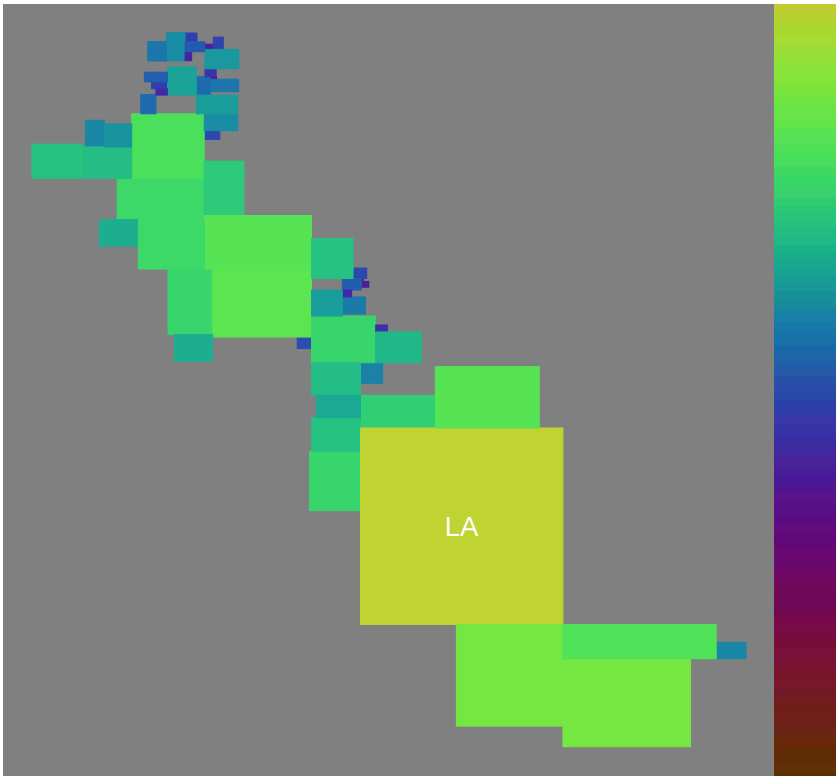


Figure 7.29: U.S. county population cartogram using *RecMap* (*MP2*)

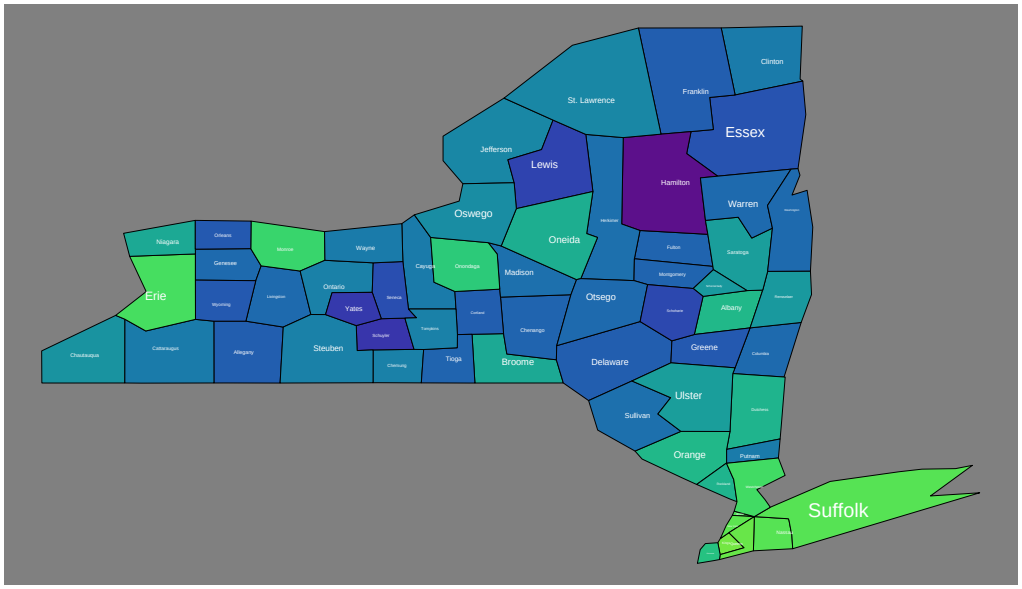


(a) Traditional California state

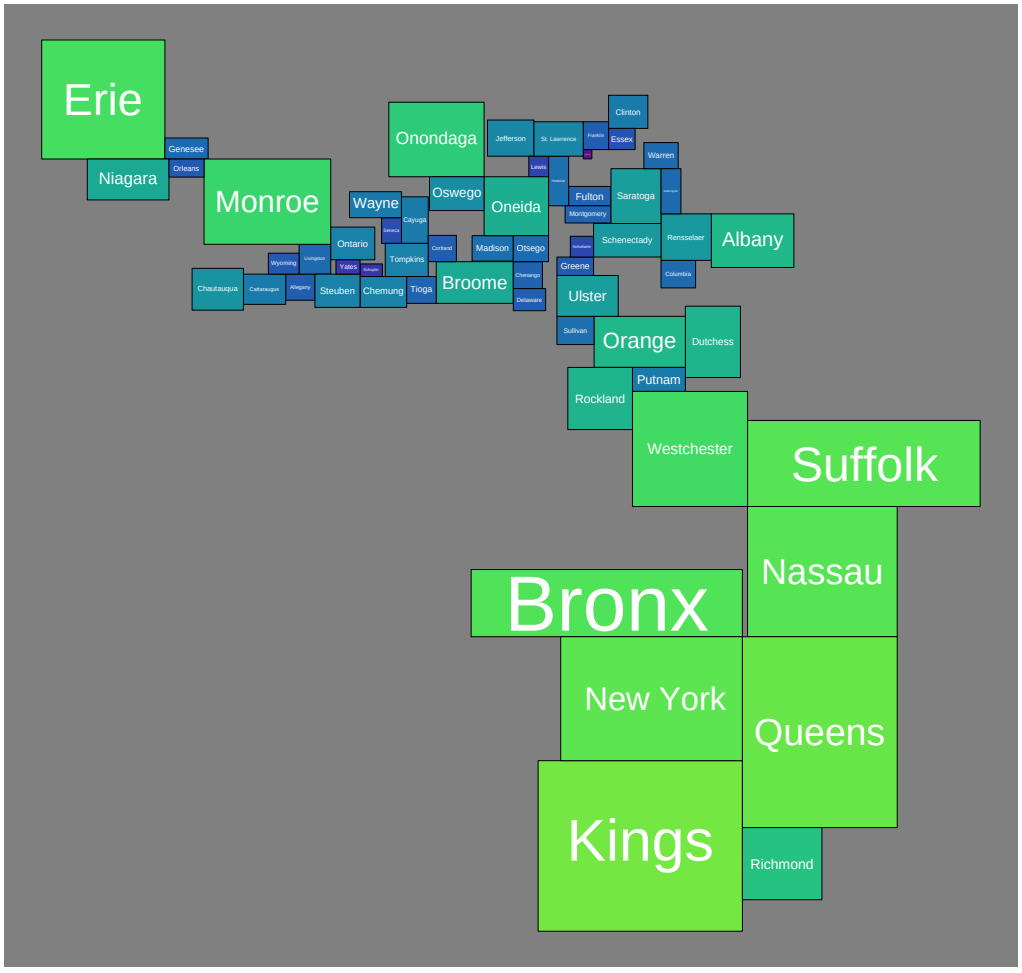


(b) *RecMap MP*2

Figure 7.30: California county population cartogram using *RecMap*

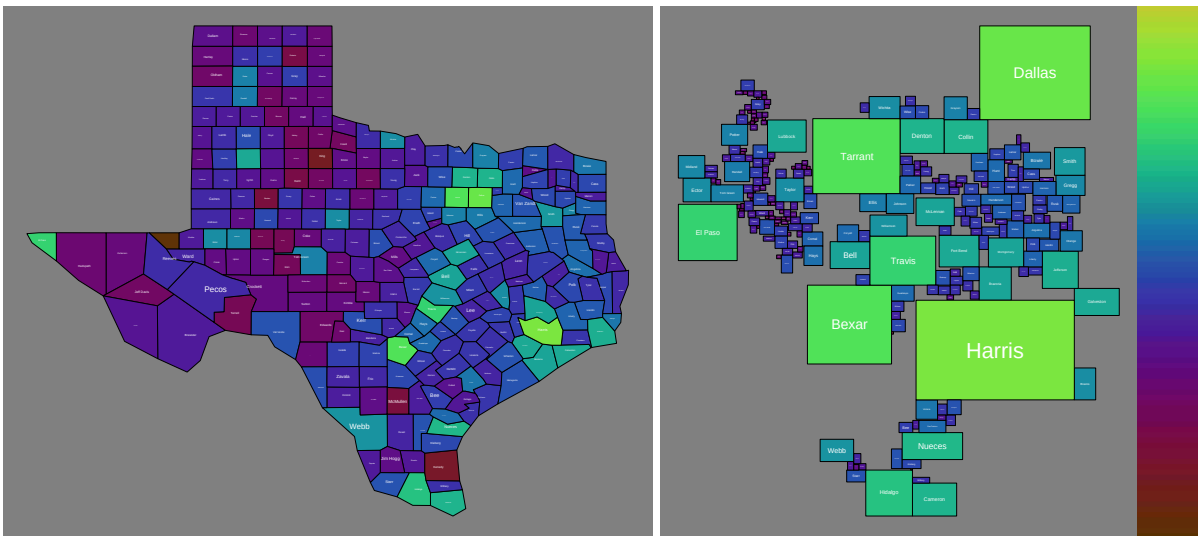


(a) Traditional map New York state



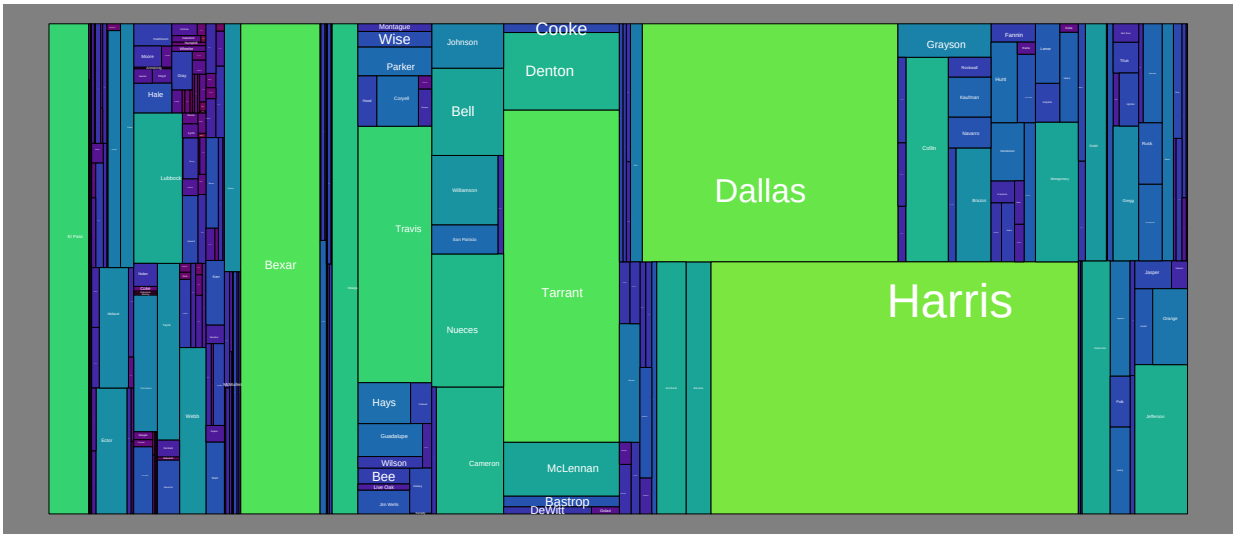
(b) *RecMap MP2*

Figure 7.31: New York U.S. census 2000 county population cartogram using *RecMap*



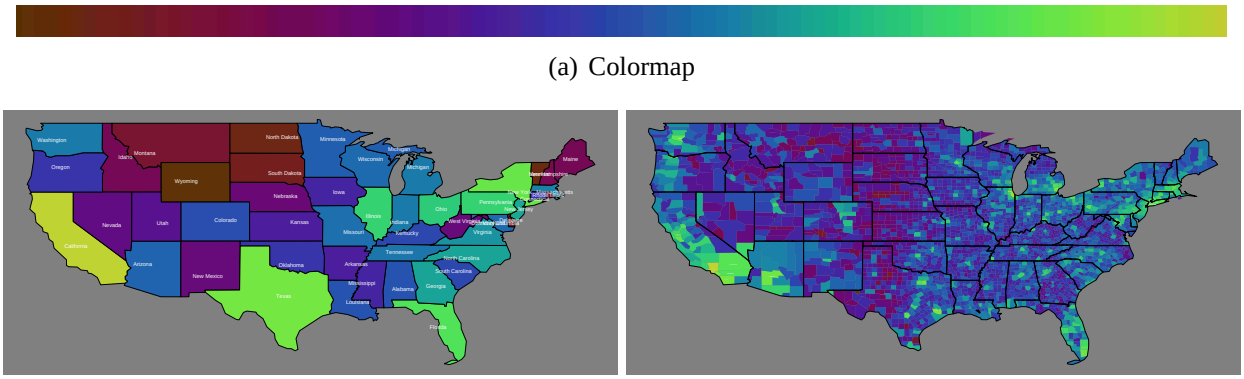
(a) Traditional map Texas state

(b) *RecMap MP*2



(c) *RecMap MP*1

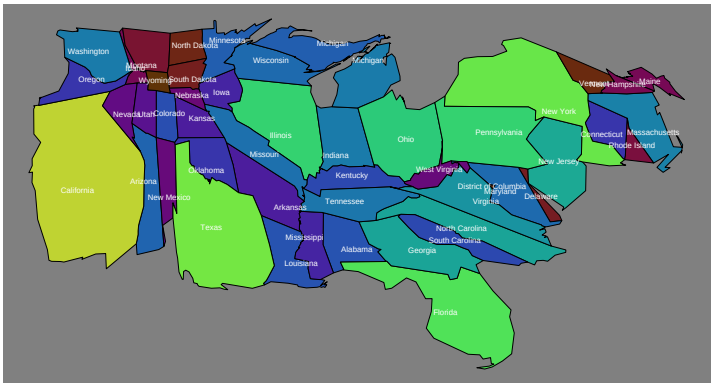
Figure 7.32: Texas county population cartogram using *RecMap*



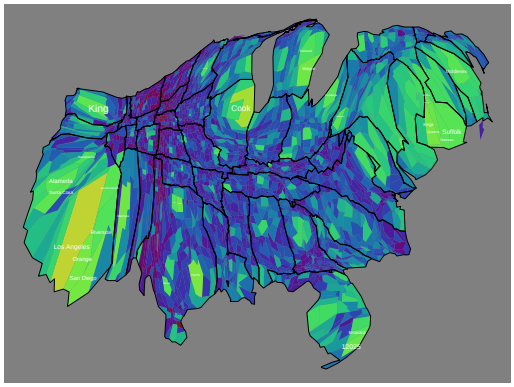
(a) Colormap

(b) Traditional U.S. map on state level

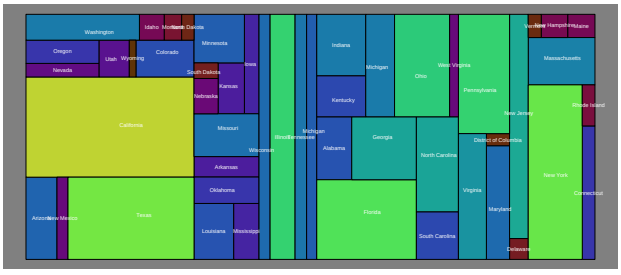
(c) Traditional U.S. map on county level



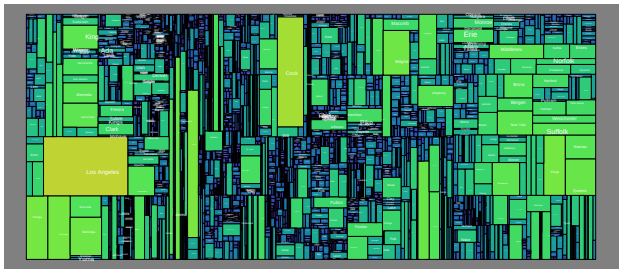
(d) *M-CartoDraw*



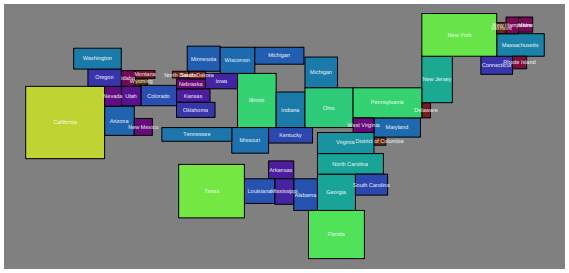
(e) *M- CartoDraw*



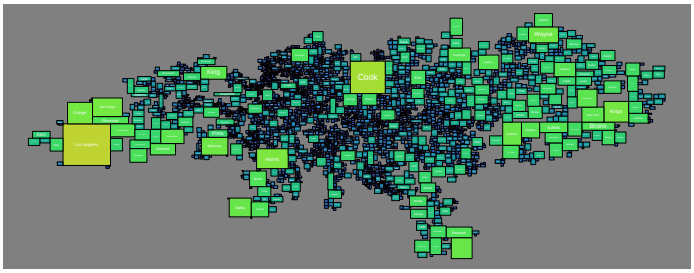
(f) *RecMap MP*1



(g) *RecMap MP*1



(h) *RecMap MP*2



(i) *RecMap MP*2

Figure 7.33: Population data from U.S. Census Bureau on various levels for the year 2000 – The area of each map partition corresponds to the number of people living there. The colormap of Figure 7.33(a) indicates the number of people living in each region (yellow: high population; brown: low population) and is a link from the traditional maps 7.33(b),7.33(c) to the corresponding cartogram 7.33(d)–7.33(i).

8 Conclusions

The work presented in this dissertation introduces two completely new methods for generating value-by-area cartograms called *CartoDraw* and *RecMap*. Both algorithms cover a wide range of user requirements (area, topology, shape, empty space, and computational time) which can be directly maintained by the user. The first algorithm strictly retains the topology and shape of the map regions while minimizing the *area error*. The second algorithm approximates the map regions by rectangles focussing on an exact area approach.

Both algorithms are fast enough to be used in interactive systems which is important to be used as information visualization technique.

The thesis is attended by many extensions such as *texture mapping cartograms*, pseudo value-by-area cartograms (*HistoScale*), and pixel based cartograms (*Visual Points*) as well as useful combinations with other visualization techniques such as the *HistoMap* approach.

All algorithms have been implemented and compared (visual, effectiveness, and efficiency) to state-of-the-art existing methods. The results of our novel algorithms are comparable if not better than existing ones. The algorithm were bundled into one application called *CartoDraw*-System.

Furthermore, the *CartoDraw* algorithm has been designed to generate cartograms for maps with a high number of polygons in an adequate computational time. Both algorithms, *RecMap* and *CartoDraw*, are scalable enough to make cartograms for the 3000 counties of the United States map.

The application of our techniques is shown by many cartograms using different input maps and data. We used real world maps as well as self-generated data.

The work is based on a theoretical analysis which shows that the cartogram problem is unsolvable in the general case. To achieve feasible solutions we have to relax some constraints.

This work opens a new area in the flexibility of cartogram generation. Never before users have had such a wide flexibility of user constraints.

Also, this thesis provides some new research directions.

- Value-by-area cartograms are just one of many cartogram types. We can formally define many cartogram variants. For some cartogram types there exist polynomial time algorithms (e.g choropleth maps). For other types of cartograms there exist heuristics because no polynomial time algorithm are known. A complexity classification of cartogram types does not exist and neither does the $\mathcal{NPC}$ proof. To the best of our knowledge for some types there exist just hand-made visualizations (e.g. isochrones).

- Behind the theoretical goals there are some practical issues. A further promising application of the scanline-technique used in *CartoDraw* are the generation of route cartograms as it can be seen in figure 8.1. Existing work does not take the relative positions and angle between the edges into account. For the problem of route cartograms similar constraints can be defined as for the cartogram drawing problem presented in this thesis.

- Using *CartoDraw* and *HistoScale* we can compute world cartograms. As mentioned earlier *HistoScale* computes only a pseudo-cartogram transformation. *CartoDraw* can minimize the *area error* of the map regions. A possible problem with *CartoDraw* is that there exist several non-connected map components of the world map. During the scanline transformation the particular map components move over the “map space”. Therefore, it is not guaranteed that the transformation is

free of overlapping. A pre-processing algorithm for generating an acceptable starting position is needed. This can be done similar to *RecMap* by formulating an optimization problem that computes an overlap-free *multi component cartogram (MCC)* where the area of the component regions are proportional to the cumulated statistical map regions of each component. The *MCC* algorithm minimize the relative position error as well as the aspect ratios of the map components.

- An algorithm is just approved if it is a steady component of widely-distributed and often-used software. The complete integration of the described methods into existing GIS software is of interest. Difficulties are the existence of several different geographic formats.

- One possible future application of *CartoDraw* could be an application as *"route guide"* similar to the famous *LineDrive* [1]. Using *CartoDraw*, the map transformation can be done by the *scanlines*. The *scanlines* are guided by the route. The advantage to the methode of Agrawala and Stolte [1] is that the maps topology and relative position are preserved and the "driver" has additional landmarks for orientation. However, to realize this project we need complex new cost functions to evaluate the results. Figure 8.1 gives an impression of this idea.

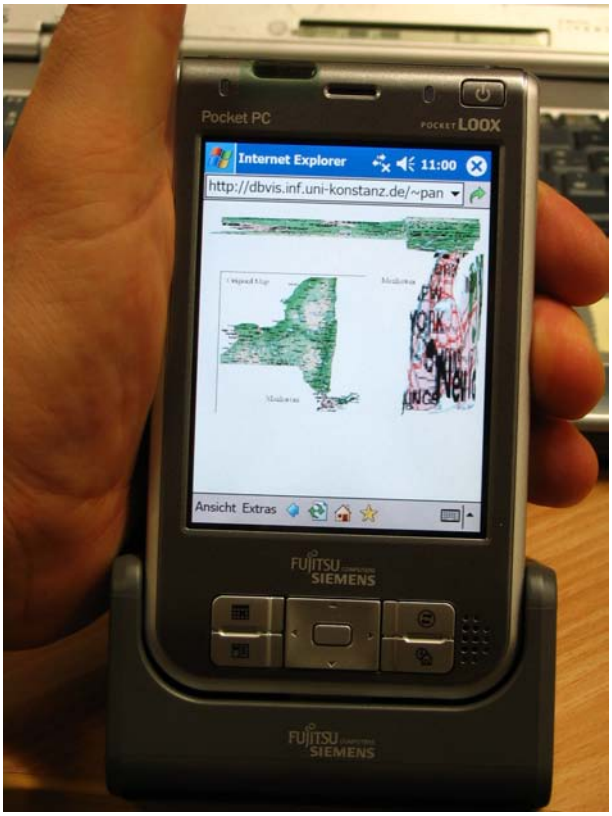


Figure 8.1: Maps and cartograms have always been used by humans to encode valuable and important information. These visualizations were often drawn on light materials to have a handy device to carry. The picture shows a possible future application of *CartoDraw* as *"route guide"* of NYC on a PDA.

A Measuring the Shape Error by Fourier Transformation

A shape similarity function compares the new shape of a polygon with its original one. Beside measuring the shape error by summing up angle differences of the input and output mesh (see chapter 4), we can also use a more general Fourier transformation based approach. The advantage here is we do not need a graph isomorphism between the two meshes. Defining a useful shape similarity function is in itself a difficult problem, since the similarity measure should be translation-invariant, scale-invariant, and at least partially rotation-invariant. From CAD research it is known that the Euclidean distance in Fourier space is useful for measuring shape similarity [67, 8]. To gain invariance against translation, rotation, and scaling, we use the Fourier transformation of the differential geometric curvature of the polygons, instead of the polygons themselves, and normalize the arc length of the polygons to 2π. Using the curvature guarantees translation- and rotation-invariance, and normalizing the arc length guarantees scale-invariance.

In the following, we assume that the polygons are transformed into a normalized parameterized polygon contour function $p : [0, 2\pi] \rightarrow \mathbb{R}^2$. Then, we can define the curvature C of the polygons as

$$C : (\mathbb{R} \rightarrow \mathbb{R}^2) \longrightarrow (\mathbb{R} \rightarrow \mathbb{R}^2). \tag{A.1}$$

The Fourier transformation F is a transformation

$$F : (\mathbb{R} \rightarrow \mathbb{R}^2) \longrightarrow \mathbb{R}^d, \tag{A.2}$$

determining the Fourier coefficients for a given curvature function in d-dimensional Fourier space. The shape similarity of two polygons p and $\overline{p}$ can then be defined as

$$\tilde{d_S}(S(p), S(\overline{p})) = d_{Euclid}(F(C(p)), F(C(\overline{p}))). \tag{A.3}$$

In the following, we describe the curvature transformation C and the Fourier transformation F in more detail.

Determining the Curvature of a Polygon

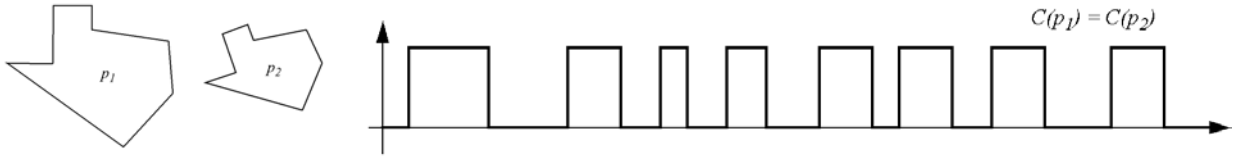
In general, the curvature of a polygon defined as a parameterized function is mathematically undefined because the second derivative is not continuous. We can avoid this problem by approximating the polygon by replacing each vertex with very small circular arcs as shown in Figure A.1. This yields a new geometric object of which the first derivative is continuous. The curvature of this structure is defined in sections; concatenating these sections we obtain the curvature as a square wave function.

To describe the curvature transformation in more detail, let us focus on two subsequent edges e_{i-1} and e_i. These edges coincide in vertex v_i with an angle α_i. For the polygon containing v_i, we may easily compute the curvature function $c_i(t)$, describing the differential geometric curvature of the approximated polygon because the curvature of a circle segment with radius r is a constant function $\frac{1}{r}$ and the curvature of a straight line is a constant zero function. We may calculate the arc length of the circle segment substituting vertex v_i by $b_i = |\alpha_i| \cdot r$. For $c_i(t)$, we therefore obtain

$$c_i(t) = \begin{cases} \frac{1}{r} & \text{if } (t_{v_i} - b_i/2 \ > \ t \ > \ t_{v_i} + b_i/2) \\ 0 & \text{otherwise} \end{cases} \tag{A.4}$$

(a) Original polygon (b) Approximated polygon (c) Curvature

Figure A.1: Approximation of a polygon



(a) Two polygons (b) Curvature of the two polygons

Figure A.2: Curvature transformation

The curvature of an arbitrary polygon p is $c(t) = \sum_{k=0}^{|p|-1} c_k(t)$. Figure A.1c shows the graph of the curvature function $c(t)$ for the approximation of the polygon section in figure A.1a. Figure A.2 shows the curvature functions for two polygons which are identical under translation-invariance, rotation-invariance, and scale-invariance.

The approximation of the original polygon, and in particular the choice of r, influences the curvature function. If we reduce the radius r of the circle segment, $\frac{1}{r}$ will be increased while b_i will be decreased. This causes $c(t)$ to become more narrow and the amplitude of square waves to become higher, while the approximation of the polygon converges against the polygon itself. On the other hand, $c(t)$ becomes difficult to handle numerically. An adequate value for r which has proven useful for our application is $\frac{\pi}{50}$ for polygons with a normalized length of 2π. As our experiments show, the similarity function is quite robust against a suboptimal choice of r, as long as r is smaller than half of the length of the shortest edge since otherwise individual square wave functions may overlap.

Fourier Transformation

The next step is computing the Fourier transformation F of the curvature. The principle of the Fourier transformation is to approximate a function by summing up sine and cosine functions with certain parameters. The quality of the approximation is improved by increasing the degree d of the Fourier approximation, which means to successively sum up $\cos(x), \sin(x), \cos(2x), \sin(2x), \ldots, \cos(x), \sin(x)$. More formally, the Fourier approximation of a function f with a period of 2π is defined as

$$F(x) = \frac{a_0}{2} \sum_{k=1}^{n} (a_k \cos(kx) + b_k \sin(kx)) \tag{A.5}$$

where the coefficients a_k and b_k are defined as

$$a_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(kx) dx \quad \text{and} \tag{A.6}$$

$$b_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(kx) dx \tag{A.7}$$

In general, integrals of the form $\int f(x) \sin(x) dx$ are difficult to solve analytically. For the special case where $f(x)$ is a square wave function, however, the integral can be easily determined. Let us assume that

$f(x)$ has a value of $\frac{1}{r}$ in the interval $[u,v]$ and is zero elsewhere. Since the value of the integral is zero outside of $[u,v]$, we just have to integrate from u to v. Therefore, we are able to calculate a_k and b_k as:

$$a_k = \frac{1}{\pi kr}(\sin(kv) - \sin(ku)) \quad \text{and} \tag{A.8}$$

$$b_k = \frac{1}{\pi kr}(\cos(kv) - \cos(ku)) \tag{A.9}$$

To determine the Fourier coefficients of the curvature function $c(t)$ of the whole polygon p, we only have to sum up the above formula $c_i(t)$ for all vertices v_i of the polygon. We obtain the following formulas for the Fourier coefficients:

$$a_k = \frac{1}{\pi kr}\sum_{i=0}^{|p|-1}\frac{\alpha_i}{|\alpha_i|}\left[\sin\left\{k\cdot\left(t_i+\frac{|\alpha_i r|}{2}\right)\right\} - \sin\left\{k\cdot\left(t_i-\frac{|\alpha_i r|}{2}\right)\right\}\right] \tag{A.10}$$

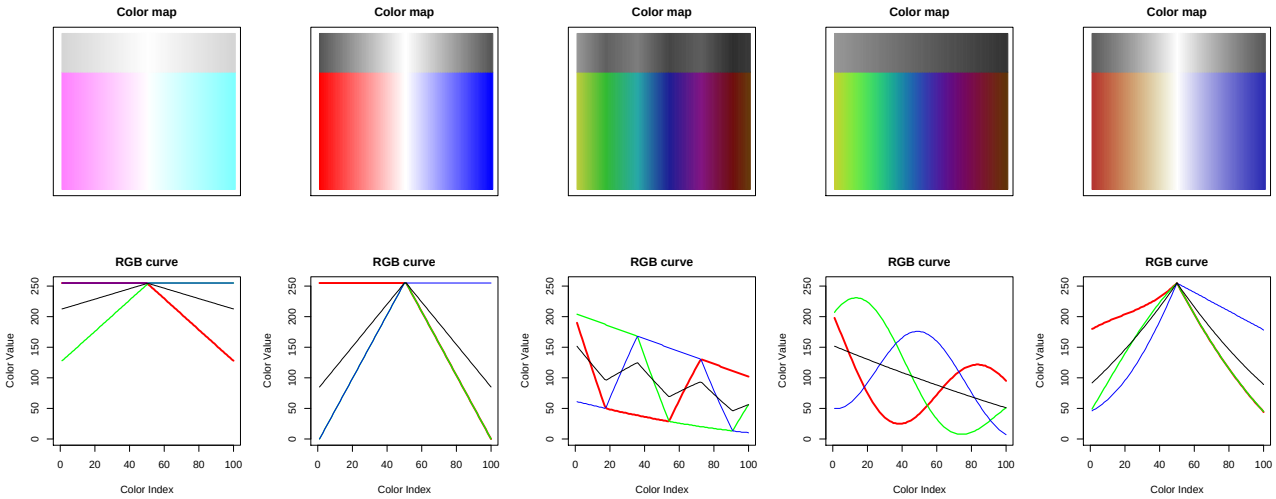
$$b_k = -\frac{1}{\pi kr}\sum_{i=0}^{|p|-1}\frac{\alpha_i}{|\alpha_i|}\left[\cos\left\{k\cdot\left(t_i+\frac{|\alpha_i r|}{2}\right)\right\} - \cos\left\{k\cdot\left(t_i-\frac{|\alpha_i r|}{2}\right)\right\}\right] \tag{A.11}$$

The calculation of a_k and b_k can be done in $O(|p|)$ time, and the calculation of all coefficients can be done in $O(|p|\cdot d)$, where d is the degree of the Fourier sum. Note that we are able to compute the coefficients of the Fourier sum analytically, and therefore do not run into numerical problems such as finding the right sample rate. Experimental results show that the Fourier transformation provides a good approximation of the polygons and their curvature function even for rather small d. For a detailed discussion of Fourier theory see [133].

B Color Maps Employed

Color is a powerful tool for decoding information in visualization. Instead of using color as assemble for categories, we use it regarding to our cartograms as perception of the order of quantities to the statistical values linked to the map regions. The quantities in our cartograms are often population data, election results, or area error. For that issue we need a sequence of contiguous color values, called color map, which can be obtained by going in a smooth curve through a color model. A good color map has a continuously monotonic increasing or decreasing of the brightness. As a second criteria a perfect color map should use a wide range of brightness.

Since all computer graphic displays use the additive RGB (R: Red, G:Green, B:Blue) color model, all color values have to be convert into the RGB color model (see [44]). In the following we will shortly explain the idea behind some color maps used in this thesis.



(a) Trellis-RGB (b) Bipolar-RGB (c) Uni-polar-HSV (d) Uni-polar-HSI (e) Uni-bipolar-HSI

Figure B.1: The plot displays the color maps and their corresponding color values used in this thesis. The color maps obtained from various color models have been converted into RGB color values. The black curves and the grey color maps indicate the brightness on a non-color device.

Trellis – color map This bipolar color map (see figure B.1(a)) has been introduced by [22]. This color map has been created for color ink printers. Since color ink printers use cyan, magenta and yellow as basic colors, all other colors can be obtained by mixing the 3 basic color. It becomes clear, if we use the cyan – magenta color map, we do not have to mix the colors and we will get a very clear result. This color map is the standard color map in the *S-Plus* and *R* software package (see [29, 105]).

Bipolar – color map We used that bipolar color map to indicate the area error of the map regions. The red color indicates if the area error is negative and the area of that region has to be smaller. The opposite is true for the the blue color. The area error can be obtained by the function

$$f_{AreaError} : \mathbb{R} \times \mathbb{R} \rightarrow [-1, 1] \in \mathbb{R}. \tag{B.1}$$

A single color value can be determined by

$$\gamma_{AreaError} : \mathbb{R} \times \mathbb{R} \rightarrow ([0,255],[0,255],[0,255]) \in (\mathbb{N} \times \mathbb{N} \times \mathbb{N}) \tag{B.2}$$

as follows:

$$c_{value} = (1 - f_{AreaError}) \cdot 255 \tag{B.3}$$

$$\gamma_{AreaError} = \begin{cases} (c_{value}, c_{value}, 255) & f_{AreaError} > 0 \\ (255, c_{value}, c_{value}) & \text{otherwise} \end{cases} \tag{B.4}$$

Figure B.1(b) shows the whole color map, where we used a sequence of reals in the interval $[-1,1]$ instead of f_{color}.

Uni-polar – HSI color maps The HSI (H: Hue, S: Saturation, I: Intensity) color model has been introduced by [68] and is a variation of the HSV (H: Hue, S: Saturation, V: Value) color model where a circular cone is used instead of a hex cone.

The advantage of HSI model is that a monotonically increasing/decreasing color map can be constructed (see the black curves on figure B.1(d) of the HSI color maps).

In contrast to a color map using the HSV model, the brightness ranges continuously from light to dark as it can be seen in figure B.1(d) This „HSV"-phenomena can be observed in figure B.1(c). It can be seen that the HSV color map has at least 3 local minima and maxima in contrast to the HSI color map which is monotonically decreasing. For both color maps we used similar start and end settings for hue, saturation, and intensity. To map the HSI and HSV color values to RGB color values we have used the HSI to RGB converter operator described in [68, page 99].
Furthermore, using the HSI/HSV model three perceptional attributes can be expressed. In practice, only two are useful.

Uni-bipolar color maps Both uni-polar and bipolar color maps can be combined. A possible result can be seen in figure B.1(e).

For generating useful color maps, we have developed a color map construction tool, where the user can walk through different color models with several start and end configurations. Using this tool, the user can chose between RGB, HSV, and the HSI color models. The tool was implemented in *Java* and it can be run as *Java*-applet [2] and can be used from all *Java* supporting web-browsers.

C Scripts and Tools for Generating Cartograms

Beside the graphical user interface GUI of the CartoDraw-System on page 85 we have developed several scripts and command line programs for generating cartograms as well as input data. The drawback of the GUI version is that it is difficult to run the algorithm as remote job. This is necessary whenever we want to compute cartograms at the same time e.g. if we have to compute cartogram sequences or if we have massive input mesh where the number of polygons is larger than 500. Additionally, it is useful for benchmarks. In the following appendix we describe some of the tools.

Generating a Checker Board Mesh First of all we need data to feed the algorithms. Often data are copy righted, difficult to achieve, or the data often contain digitizing errors. These problems are neither motivating nor gratifyingly.

Based on the idea of our beginning checker board examples in chapter 3 we wrote a script solving this difficulty. `createChecker.pl` is a small perl script, that will create arbitrary sized checker board meshes in the LEDA graph-file format as well as a parameter vector file. This script is especially useful for our *RecMap* algorithm introduced in 5.

```
 1  #!/usr/bin/perl -w
 2  use strict;
 3  sub createCheckerBoard{
 4     my $ncount=1; my $ecount=1;
 5     my $n = shift (@_);
 6     if (!defined ($n))
 7     { $n = 2;}
 8     open(LOG, ">/tmp/checker.log");
 9     my $nn = ($n-1) * ($n-1);
10     print LOG "$nn\n";
11     my %map;
12     open(FILE, ">/tmp/checker.gw");
13     print FILE "LEDA.GRAPH\npoint\nint\n";
14     # compute nodes
15     print FILE $n*$n;
16     print FILE "\n";
17     for (my $i = 0 ; $i < $n; $i++)
18     {
19        for (my $j = 0 ; $j < $n; $j++)
20        {
21           print FILE "|{($i,$j)}|\n";
22           $map{$i."|".$j} = $ncount++;
23        }
24     }
25     # compute edges
26     print FILE 4*($n-1)*($n-1);
```

```
27      print FILE "\n";
28      for (my $i = 0 ; $i < $n-1; $i++)
29      {
30         for (my $j = 0 ; $j < $n-1; $j++)
31         {
32            print FILE $map{$i."|".$j}; print FILE " ";
33            print FILE $map{($i+1)."|".$j}; print FILE " ";
34            print FILE "|{$ecount}|\n";
35
36            print FILE $map{($i+1)."|".$j}; print FILE " ";
37            print FILE $map{($i+1)."|".($j+1)}; print FILE " ";
38            print FILE "|{$ecount}|\n";
39
40            print FILE $map{($i+1)."|".($j+1)}; print FILE " ";
41            print FILE $map{$i."|".($j+1)}; print FILE " ";
42            print FILE "|{$ecount}|\n";
43
44            print FILE $map{$i."|".($j+1)}; print FILE " ";
45            print FILE $map{$i."|".$j}; print FILE " ";
46            print FILE "|{$ecount}|\n";
47            $ecount++;
48         }
49      }
50      close (FILE);
51
52      open(FILE, ">/tmp/checker.dat");
53      for (my $k = 1; $k < $ecount; $k++)
54      {
55         if ($k%2 == 0)
56         { print FILE "$k 4\n"   }
57         else
58         { print FILE "$k 1\n"   }
59      }
60      close (FILE);
61   }
62   #main
63   if ($#ARGV == 0)
64   {
65      my $n = $ARGV[0];
66      print "creating a $n x $n checkerboard in /tmp/ ... \n";
67      createCheckerBoard($n+1);
68   } else { print "error\n"; }
```

If, for example, we need a 2×2 checker board as it can be seen in figure 3.2(a) on page 20, we could run the perl script as follows:

```
% ./createChecker.pl 2
creating a 2 x 2 checkerboard in /tmp/ ...
```

The contents of the files is this:

```
% cat /tmp/checker.gw
LEDA.GRAPH
point
int
9
|{(0,0)}|
|{(0,1)}|
|{(0,2)}|
|{(1,0)}|
|{(1,1)}|
|{(1,2)}|
|{(2,0)}|
|{(2,1)}|
|{(2,2)}|
16
1 4 |{1}|
4 5 |{1}|
5 2 |{1}|
2 1 |{1}|
2 5 |{2}|
5 6 |{2}|
6 3 |{2}|
3 2 |{2}|
4 7 |{3}|
7 8 |{3}|
8 5 |{3}|
5 4 |{3}|
5 8 |{4}|
8 9 |{4}|
9 6 |{4}|
6 5 |{4}|
% cat /tmp/checker.dat
1 1
2 4
3 1
4 4
```

`createChecker.pl` generates two files in `/tmp`. `checker.gw` contains the LEDA graph-file consisting of nine nodes and sixteen edges. Each edge is assigned with the polygon id its belongs to. The other file, named `checker.dat`, stores the parameter value for each polygon in `checker.gw`.

So the idea of the parameter setting is that every second polygon has too be half sized and the other ones have to be doubled of the area of the original polygons.

Computing a Rectangular Cartogram For generating a rectangular cartogram as described in chapter 5 we can now use the following command.

```
% recmap
usage: ./recmap <gw-file-path>\
   <gw-file>\
   <gw-file-suffix>\
```

```
    <dat-file-path>\
    <dat-file>\
    <dat-file-suffix>\
    <output dir>\
    <number of iterations>
usage: ./recmap --help
usage: ./recmap -?
```

If we run `recmap` without arguments we will get the above as result. To gain a cartogram of our self generated mesh we have to run `recmap` as follows:

```
% recmap /tmp/ checker gw /tmp/ checker dat /tmp/ 10
```

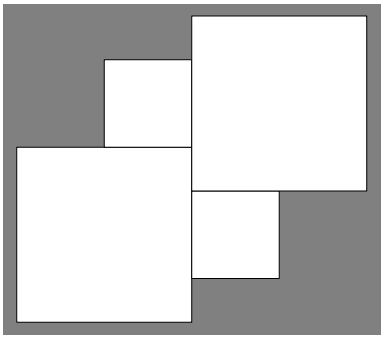
`recmap` will save the result in various formats under the `/tmp` path.

Plotting the Resulting Map To plot our result we use the `R` program, see [105] for more details. `R` has attractive graphic devices such as postscript and pdf which are very useful especially for studying extremely large maps. A possible `R`-script for plotting our rectangular cartogram could look like this:

```
 1  #R
 2
 3  # read the polygon
 4  polys<-read.table("/tmp/recmap2_checker.polygon",sep="|");
 5  dx<-max(polys$V1,na.rm=T)-min(polys$V1,na.rm=T)
 6  dy<-max(polys$V2,na.rm=T)-min(polys$V2,na.rm=T)
 7
 8  myheight<-20
 9  mywidth<-dx/dy*myheight
10
11  # keeps aspect ratio of the map
12  pdf("/tmp/carto.pdf",width = mywidth, height = myheight)
13
14  my.grey<-rgb(0.5,0.5,0.5)
15
16  op<-par(mar=c(0,0,0,0),bg=my.grey);
17
18  plot(polys,type="n", axes=F, xlab="", ylab="");
19  polygon(polys, border="black",col="white");
20
21  dev.off()
```

Finally, to run the script and to view the resulting pdf file we type the following command in a shell:

```
% cat plotCartogram.R | R --no-save && gv /tmp/carto.pdf
```



Because our map contains polygonal information we can easily extent the `R` script to fill the polygons with color using a color mapping described in the previous chapter to indicate a second statistical value[1]. As it can be seen in the application chapter beginning on page 89 there is also the possibility to label the cartogram regions.

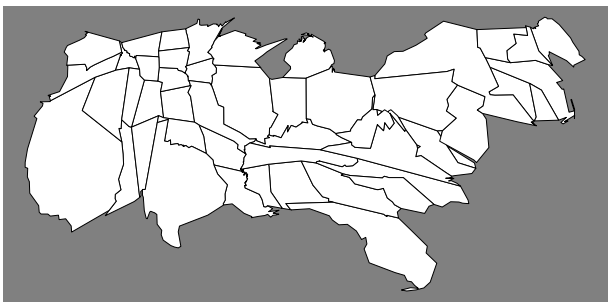
Computing a Continuous Cartograms Finnaly, we want to demonstrate the `cartodraw` command line program. The algorithm described behind the command has been introduced in chapter 4. The usage is similar to the `recmap` command. The only difference is that we will get a continuous cartogram as result.

```
% cartodraw --graph-file us.gw\
     --stat-file us-pop.psv\
     --iterations 8\
     --output-directory /tmp
```

The current version of the program does not provide us with an easy to plotting polygon file. So we must use an other program, called `gw2polygon`, to map the resulting LEDA graph-file into a polygon file which is easily understood by `R`.

```
% cat /tmp/cartodraw.gw | gw2polygon > /tmp/cartodraw.polygon
```

For the plotting we can use again our `R`-script listed on page 132.



[1] Usually the first statistical value is represented by the area of each cartogram region.

D Symbols

Symbol	Description	Chapter
$\mathcal{P}$	input set of connected simple polygons	3
$\|\mathcal{P}\|$	number of polygons in $\mathcal{P}$	3
p	input polygone $p \in \mathcal{P}$	3
$\|p\|$	number of point in p	3
$\mathcal{X}$	input parameter vector	3
x_i	element of $\mathcal{X}$	3
$\tilde{\mathcal{X}}$	a vector of desired area of a region	3
$\tilde{x_i}$	element of $\tilde{\mathcal{X}}$	3
$\overline{\mathcal{P}}$	output set of connected simple polygons	3
$\overline{p}$	output polygone $\overline{p} \in \overline{\mathcal{P}}$	3
$\widetilde{\mathcal{P}}$	partial cartograms	3
$\|\overline{p}\|$	number of point in $\overline{p}$	3
$GP(\mathcal{P})$	global polygon of $\mathcal{P}$	3
$I(v,\sigma)$	vertex' importance	3
$Sig(\alpha^v,\sigma)$	significance of the angle	3
$A(p)$	area of a polygon p	3
$S(p)$	shape of a polygon p	3
$T(\mathcal{P})$	topology of a set of polygons	3
v^i_j	i-th vertex of a polygon p_j	3
e^i_j	i-th edge of a polygon p_j	3
$\|e^i_j\|$	the length of edge e^i_j	3
$CE(v)$	cyclic order of edges at vertex v	3
d_T	topology distance function	4, 5
d_A	area distance function	4, 5
d_S	shape distance function	4, 5
d_R	relative position distance function	5
d_E	empty space distance function	5
$\mathcal{M}$	set of feasible solutions	4
f	objective function	4, 5
$\widehat{f}$	weighted objective function	5
f_C	cartogram mapping function	6
$c(p)$	centers of gravity of p	5
ψ	mesh to mesh mapping function	4
I_λ	construction sequence	5
λ	split position	5
$\mathcal{N}(p_r)$	neighbors of p_r in $\mathcal{P}$	5
$w_t, w_a, w_s \ldots$	weights	3, 4, 5

Bibliography

[1] Maneesh Agrawala and Chris Stolte. Rendering Effective Route Maps: Improving Usability Through Generalization. ACM, 2001.

[2] Anke Altintop. Color Map Tool, Nov 2004. `http://dbvis.inf.uni-konstanz.de/tools/colormap`, Sat Dec 4 13:44:27 CET 2004.

[3] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Ken Perlin, David Ratajczak, and Kathy Ryall. Human-guided greedy search: Combining information visualization and heuristic search. In *Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM '99), Kansas City, Missouri, USA, November 6, 1999, Proceedings*, pages 21–25. ACM, 1999.

[4] D. Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM Journal of Science & Stat. Comp.*, 6:128–143, 1985.

[5] Barco Reality Sim 4, Nov 2003. Ref.no. R599656.

[6] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis Tollis. *Graph Drawing*. Prentice Hall, 1st edition, 1999.

[7] Richard A. Becker and Allan R. Wilks. Maps in s. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 07974, February 1993.

[8] S. Berchtold, Daniel A. Keim, and H.-P. Kriegel. Using extended feature objects for partial similarity retrieval. *VLDB Journal*, 6(4):333–348, 1997.

[9] Jacques Bertin. *Graphische Darstellungen*. de Gruyter, 1st edition, 1982. `http://www.infovis.net/E-zine/2003/num_116.htm`, March, 2003.

[10] T. Biedl and B. Genc. Complexity of Octagonal and Rectangular Cartograms. Technical report, School of Computer Science; University of Waterloo, 2005.

[11] H. Blum and R. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, (10):167–180, 1978.

[12] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling*. Springer, 2nd edition, 2005.

[13] J. W. Brand and V. R. Algazi. Continues skeleton computation by voronoi diagram. *CVGIP: Image understanding*, (55):329–338, 1992.

[14] CAIDA (Cooperative Association for Internet Data Analysis). Visualizing internet topology at a macroscopic scale, March 2003. `http://www.caida.org/analysis/topology/as_core_network/`.

[15] Stewart Scott Cairns. *Introductory Topology*. The Ronald Press Company – New York, 1st edition, 1961.

[16] Stuart K. Card, Jock D. MacKinlay, and Ben Shneiderman. *Readings in Information Visualization Using Vision to Think*. Morgan Kaufmann, 1st edition, 1999.

[17] M.S.T. Carpendale, D.J. Cowperthwaite, M. Tigges, A. Fall, and F.D. Fracchia. The TARDIS: A visual exploration environment for landscape dynamics. *Visual Data Exploration and Analysis VI, Proc. SPIE*, 3643:110–119, January 1999.

[18] C. Cauvin, C. Schneider, and G. Cherrier. Cartographic transformations and the piezopleth method. *The Cartographic Journal*, 26(2):96–104, December 1989.

[19] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal Amer. Statistical Association*, 68:361–368, 1973.

[20] F. Chin, J. Snoeyink, and Wang. Finding the medial axis of a simple polygon in linear time. *Discrete & Computational Geometry*, (21):405–420, 1999.

[21] William S. Cleveland. *Visualizing Data*. Hobart Press, Summit, New Jersey, U.S.A, 1st edition, 1993. `http://cm.bell-labs.com/cm/ms/departments/sia/wsc/`.

[22] William S. Cleveland. *The Elements of Graphing Data*. Hobart Press, Summit, New Jersey, U.S.A, 1st edition, 1994.

[23] William S. Cleveland and Marylyn E. McGill. *Dynamic Graphics for Statistics*. Wadsworth, 1st edition, 1988.

[24] M. de Berg, M. van Kreveld, , M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications*. Springer, Berlin, 2 edition, 2000.

[25] Borden Dent. A note on the importance of shape in cartogram communication. *The Journal of Geography*, 71(7):393–401, October 1972.

[26] Borden Dent. Communication aspects of value-by-area cartograms. *The American Cartogapher*, 2(2):154–168, October 1975.

[27] Borden D. Dent. *Cartography: Thematic Map Design, 5th Ed., Chapter 11*. WCB/McGraw-Hill, Dubuque, IA, 1999.

[28] T. K. Dey and W. Zhao. Approximating the medial axis from the voronoi diagram with a convergence guarantee. *LNCS 2461*, pages 87–398, 2002.

[29] Data Analysis Division. *S-Plus 6.0 for UNIX Programmer's Guide*. MathSoft, Seattle, WA, 2000.

[30] Daniel Dorling. *Area Cartograms: Their Use and Creation*. Department of Geography, University of Bristol, England, 1st edition, 1996.

[31] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[32] J. Dykes. Cartographic visualization: Exploratory spatial data analysis with local indicators of spatial association using tcl/tk and cdv. *The Statistician*, 47(3):485–497, 1998.

[33] J. Dykes, A. MacEachren, and M.-J. Kraak, editors. *Exploring Geovisualization*. Oxford: Elsevier, 2004.

[34] Herbert Edelsbrunner. *Algorithms in Computational Geometry*. Springe-Verlag Heidelberg Berlin, 1st edition, 1987.

[35] Herbert Edelsbrunner and Roman Waupotitsch. A combinatorial approach to cartograms. *Computational Geometry*, pages 343–360, 1997.

[36] Environmental System Research Institute. ESRI Data & Maps. Technical report, April 2003. `http://www.esri.com/library/whitepapers/pdfs/datamaps2003.pdf`.

[37] ESRI. An ESRI White Paper: Customizing ArcInfo8. Technical report, Environmental System Research Institute, June 1999. `http://www.geoweb.dnv.org/Education/whitepapers/ArcINFO8Cust.pdf`.

[38] ESRI. An ESRI White Paper: Geographic Information Systems for the Java Platform. Technical report, Environmental System Research Institute, November 2002. `http://www.geoweb.dnv.org/Education/whitepapers/ArcINFO8Cust.pdf`.

[39] ESRI Web Site. , Dez 2004. http://www.esri.com/.

[40] B. S. Everitt and G. Dunn. *Applied Multivariate Data Analysis*. Arnold, 1991.

[41] fermi. Relief, Jan 2003. http://fermi.jhuapl.edu.

[42] U. Finke and J. L. Bentley. Quad trees: a data structur for retrival on composite keys. *Acta Inform*, 4:1–9, 1974.

[43] Clif Flynt. *Tcl/Tk: A Developer's Guide*. Morgan Kaufmann, 2nd edition, 2003.

[44] James. D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphic Principles and Practice*. Addison-Wesley, 2nd edition, 1996.

[45] Mark Foskey, Ming C. Lin, and Dinesh Manocha. Efficient computation of a simplified medial axis. In *Proceedings of the ACM Symposium on Solid Modeling*, pages 96–107, June 2003.

[46] L.R. Foulds. *Graph Theory Applications*. Springer, Berlin, 1992.

[47] H. Gray Funkhouser. Historical development of the geographical representation of statistical data. *Osiris*, 3:269–403, 1937.

[48] M. Gahegan. Geovista studio: A geocomputational workbench. In *5th International Conference on GeoComputation*, August 2000.

[49] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. In *Software: Practice and Experience*, pages 1203–1233, 2000. http://www.graphviz.org/.

[50] Michael T. Gastner and M. E. J. Newmann. Diffusion-based method for producing density-equalizing maps. *PNAS*, 101(20):7499–7504, 2004. http://www.pnas.org/cgi/reprint/101/20/7499.pdf.

[51] M. Gen and R. Cheng. *Genetic Algorithm and Engineering Optimization*. Wiley, New York, 2000.

[52] GRASS Development Team. GRASS GIS Homepage, Dez 2004. http://grass.baylor.edu//.

[53] Sabir Gusein-Zade and Vladimir Tikunov. A new technique for constructing continuous cartograms. *Cartography and Geographic Information Systems*, 20(3):66–85, 1993.

[54] Sabir Gusein-Zade and Vladimir Tikunov. Map transformations. *Geography Review*, 9(1):19–23, 1995.

[55] Roland Heilmann, Daniel A. Keim, Christian Panse, and Mike Sips. RecMap: Rectangular Map Approximations. In *InfoVis 2004, IEEE Symposium on Information Visualization, Austin, Texas*, pages 33–40, October 2004. http://infovis.stanford.edu/infovis/2004/slides/christian.panse.pdf, Tue Nov 23 10:31:51 CET 2004.

[56] P. J. Huber. The annals of statistics. *Projection Pursuit*, 13(2):435–474, 1985.

[57] Greg Humphreys and Pat Hanrahan. A distributed graphics system for large tiled displays. In *Proc. Visualization 1999, San Francisco, CA*, pages 215–223, 1999.

[58] John Hunter and Jonathan C. Young. A technique for the construction of quantitative cartograms by physical accretion models. *The Professional Geographer*, 20:402–406, 1968.

[59] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proc. Visualization 90, San Francisco, CA*, pages 361–370, 1990.

[60] Charles B. Jackel. Using arcview to create contiguous and noncontiguous area cartograms. *Cartography and Geographic Information Systems*, 24(2):101–109, 1997.

[61] Peter James and Nick Thorpe. *Keilschrift, Kompass, Kaugummi; Eine Enzyklopädie der frühen Erfindungen*. dtv, 2nd edition, 2002.

[62] B. Johnson and B. Shneiderman. Treemaps: A space-filling approach to the visualization of hierarchical information. In *Proc. Visualization '91 Conf*, pages 284–291, 1991.

[63] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theor. Comp. Sci.*, (172):175–193, 1997.

[64] T. Keahey and E. Robertson. Nonlinear magnification fields. *Proceedings of the IEEE Symposium on Information Visualization*, pages 51–58, 1997.

[65] T. Alan Keahey. The generalized detail-in-context problem. In *Proceedings IEEE Visualization*, 1998.

[66] T. Alan Keahey. Area-normalized thematic views. *Proceedings of International Cartography Assembly*, August 1999.

[67] L. Kehrer and C. Meinecke. *Perceptual Organization of Visual Patterns: The Segmentation of Textures*, chapter 2. Academic Press, London, 1995.

[68] Daniel A. Keim. *Visual Support for Query Specification and Data Minig*. PhD thesis, Universität München, 1995.

[69] Daniel A. Keim. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, January–March 2000.

[70] Daniel A. Keim. Visual exploration of large data sets. *Communications of the ACM (CACM)*, 44(8):38–44, 2001.

[71] Daniel A. Keim and Annemarie Herrmann. The gridfit algorithm: An efficient and effective approach to visualizing large amounts of spatial data. In *IEEE Visualization, Research Triangle Park, NC*, pages 181–188, 1998.

[72] Daniel A. Keim, Florian Mansmann, Christian Panse, Jörn Schneidewind, and Mike Sips. Mail Explorer - Spatial and Temporal Exploration of Electronic Mail. In *EuroVis 2005: Eurographics/IEEE-VGTC Symposium on Visualization, Leeds, United Kingdom*, June 2005.

[73] Daniel A. Keim, Stephen C. North, and Christian Panse. CartoDraw: A fast algorithm for generating contiguous cartograms. Technical report, Information Visualization Research Group, AT&T Laboratories, Florham Park, 2001. `http://www.research.att.com/~north/papers/01/KNP01.pdf`, Sat Dec 4 13:44:27 CET 2004.

[74] Daniel A. Keim, Stephen C. North, and Christian Panse. Method for generating contiguous cartograms, Feb 2003. United States Patent #:6853386, Grant Date:02/08/2005, Patent Application: No. 10/371714, Filed 02/21/2003.

[75] Daniel A. Keim, Stephen C. North, and Christian Panse. CartoDraw: A fast algorithm for generating contiguous cartograms. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):95–110, January/February 2004. `http://www.research.att.com/areas/visualization/papers_videos/papers/2004knp.pdf`, Tue Nov 23 12:37:25 CET 2004.

[76] Daniel A. Keim, Stephen C. North, and Christian Panse. CartoDraw Home, Nov 2004. `http://dbvis.inf.uni-konstanz.de/~panse/CartoDraw`, Sat Dec 4 13:44:27 CET 2004.

[77] Daniel A. Keim, Stephen C. North, and Christian Panse. Medial-Axis-based Cartograms. *IEEE Computer Graphics and Applications*, 25(3):60–68, May/June 2005.

[78] Daniel A. Keim, Stephen C. North, Christian Panse, Matthias Schäfer, and Mike Sips. HistoScale: An efficient approach for computing pseudo-cartograms. In *IEEE Visualization 2003 DVD-ROM, Seattle, Washington, USA*, pages 28–29, October 2003. IEEE Catalog Number 03CG37496D, ISBN 0-7803-8121-1.

[79] Daniel A. Keim, Stephen C. North, Christian Panse, and Jörn Schneidewind. Efficient Cartogram Generation: A Comparison. In *InfoVis 2002, IEEE Symposium on Information Visualization, Boston, Massachusetts*, pages 33–36, October 2002.

[80] Daniel A. Keim, Stephen C. North, Christian Panse, and Jörn Schneidewind. Visualizing geographic information: VisualPoints vs CartoDraw. *Palgrave Macmillan – Information Visualization*, 2(1):58–67, March 2003.

[81] Daniel A. Keim, Stephen C. North, Christian Panse, and Mike Sips. PixelMaps: A New Visual Data Mining Approach for Analyzing Large Spatial Data Sets. In *The Third IEEE International Conference on Data Mining (ICDM03), Melbourne, Florida, USA*, November 2003.

[82] Daniel A. Keim, Stephen C. North, Christian Panse, and Mike Sips. Pixel based visual data mining of geo-spatial data. *ELSEVIER Computer & Graphics*, 28(3):327–344, June 2004.

[83] Daniel A. Keim, Stephen C. North, Christian Panse, and Mike Sips. Visual Data Mining in Large Geospatial Point Sets. *IEEE Computer Graphics and Applications*, pages 36–44, September 2004.

[84] Daniel A. Keim, Christian Panse, Jörn Schneidewind, and Mike Sips. Analyzing Large Collections of Email. In *Analyzing Large Collections of Email, Las Vegas, Nevada*, pages 275–281, 2004.

[85] Daniel A. Keim, Christian Panse, Jörn Schneidewind, and Mike Sips. Geo-Spatial Data Viewer: From Familiar Land-covering to Arbitrary Distorted Geo-Spatial Quadtree Maps. In *WSCG 2004, The 12-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, February 2004.

[86] Daniel A. Keim, Christian Panse, Jörn Schneidewind, Mike Sips, Ming C. Hao, and Umesh Dayal. Pushing the Limit in Visual Data Exploration: Techniques and Applications. In *Advances in Artificial Intelligence, 26th Annual German Conference on AI, KI 2003, Hamburg, Germany, September 15-18, Lecture Notes in Artificial Intelligence, Vol. 2821*, September 2003.

[87] Daniel A. Keim, Christian Panse, and Mike Sips. Visual Data Mining of Large Spatial Data Sets. In *Databases in Networked Information Systems – Third International Workshop, DNIS 2003, Aizu, Japan*, pages 33–36, September 2003. ISBN: 3-540-20111-4, `http://www.springerlink.com/index/PKUCAP147A4EN0MT.pdf`, Sat Dec 4 14:01:12 CET 2004.

[88] Daniel A. Keim, Christian Panse, and Mike Sips. *Information Visualization: Scope, Techniques and Opportunities for Geovisualization*, chapter 1. In Dykes et al. [33], 2004.

[89] Brian W. Kernighan and Dennis M. Ritchie. *Programmieren in C*. Hanse, München, 2nd edition, 1990.

[90] Christopher J. Kocmoud and Donald H. House. Continuous cartogram construction. In *IEEE Visualization, Research Triangle Park, NC*, pages 197–204, 1998.

[91] T. Kohonen. *Self organising maps*. Springer, 1995.

[92] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring n-dimensional databases. In *Proc. Visualization '90, San Francisco, CA*, pages 230–239, 1990.

[93] Y. Leung and M. Apperley. A review and taxonomy of distortion-oriented presentation techniques. In *Proc. Human Factors in Computing Systems CHI '94 Conf., Boston, MA*, pages 126–160, 1994.

[94] A. MacEachren and M. Kraak. Research challenges in geovisualization. *Cartography and Geographic Information Science*, 28:3–12, 2001.

[95] Alan M. MacEachren. *How Maps Work: Representation, Visualization, and Design*. The Guilford Press, New York, 1995.

[96] MaxMind. , Dez 2004. `http://www.maxmind.com/`.

[97] Kurt Mehlhorn and Stefan Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1st edition, 1999. http://www.mpi-sb.mpg.de/~mehlhorn/LEDAbook.html.

[98] Harald Mueller. *Der geschmiedete Himmel*. Theiss, 1st edition, 2004. Begleitband zur Sonderausstellung Landesmuseum für Vorgeschichte, Halle (Saale), Germany, ISBN: 3-8062-1907-9.

[99] Tamara Munzner. Exploring large graphs in 3D hyperbolic space. *IEEE Computer Graphics and Applications*, 18(4):18–23, 1998.

[100] NED. NASA/IPAC Extragalactic Database, Dez 2004. http://nedwww.ipac.caltech.edu/.

[101] K. Neumann. *Produktions– und Operations–Management*. Springer, Berlin, 1996.

[102] Kent Norman, Ben Shneiderman, Catherine Plaisant, Evan Golub, Chris North, Gunjan Dang, Egemen Tanin, and Haixia Zhao. User interfaces for the u.s. bureau of census online survey interfaces and data visualization, Mar, 30th 2004. http://www.cs.umd.edu/projects/hcil/census/.

[103] Joseph O'Rourke. *Computational geometry in C*. Cambridge Univ. Press, 1st edition, 1994. http://cs.smith.edu/~orourke/.

[104] Brian Paul. Chromium for Cluster Rendering, Oct 2004. Vis2004 Workshop, http://graphics.stanford.edu/~mhouston/VisWorkshop04/ChromiumVis2004pt1.pdf.

[105] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0, http://www.R-project.org.

[106] Erwin Raisz. *General Cartography*. McGraw-Hill, New York, 1948.

[107] Erwin Raisz. *Principles of Cartography*. McGraw-Hill, New York, 1962.

[108] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[109] S. Selvin, D. Merrill, J. Schulman, S. Sacks, L. Bedell, and L. Wong. Transformations of maps to investigate clusters of disease. *Social Science and Medicine*, 26(2):215–221, 1988.

[110] B. Shneiderman. The eye have it: A task by data type taxonomy for information visualizations. In *Visual Languages*, 1996.

[111] Mike Sips. *Pixel Based Geo-Related Visualization*. PhD thesis, Universität Konstanz, 2005.

[112] Terry A. Slocum. *Thematic cartography and visualization*. Prentice Hall, Upper Saddle River, NJ, 1999.

[113] SpamAssassin. The Apache SpamAssassin Project, Dez 2004. http://spamassassin.apache.org/.

[114] Robert Spence. *Information Visualization*. Addison-Wesley, 1st edition, 2001. http://www.ee.ic.ac.uk/research/information/www/Bobs.html.

[115] Jonathan Stott and Peter Rodgers. Metro Map Layout Using Multicriteria Optimization. In *Proceedings 8th International Conference on Information Visualisation (IV04)*, pages 355–362. IEEE, July 2004.

[116] M. Takatsuka and M. Gahegan. GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis and Visualization. *The Journal of Computers & Geosciences*, 2002. http://www.geovistastudio.psu.edu/jsp/publications.jsp.

[117] The GDB Human Genome Database. The Official World-Wide Database for the Annotation of the Human Genome, Dez 2004. http://www.gdb.org/.

[118] The Xentera GT Series. colorgraphics, 2004. http://www.colorgraphic.net/newsite/downloads/pdf/xentera_gt_spec.pdf.

[119] The XFree86 Project, Inc. TM. Xfree86 server 4.x design (draft), Dez 2004. http://www.xfree86.org/current/DESIGN2.html.

[120] Waldo R. Tobler. Cartograms and cartosplines. *Proceedings of the 1976 Workshop on Automated Cartography and Epidemiology*, pages 53–58, 1976.

[121] Waldo R. Tobler. Pseudo-cartograms. *The American Cartographer*, 13(1):43–40, 1986.

[122] Waldo R. Tobler. Thirty five years of computer cartograms. *Annals, Assoc. Am. Geographers*, pages 43–40, March 2004. http://www.geog.ucsb.edu/~tobler/publications/pdf_docs/inprog/Thirtyfiveyears.pdf.

[123] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

[124] US. census, Oct, 30th 2004. http://www.census.org.

[125] Marc van Kreveld and Bettina Speckmann. On rectangular cartograms. In *Proc. 12th European Symposium on Algorithms (ESA), Lecture Notes in Computer Science 3221, Springer Verlag*, pages 724–735, September 2004.

[126] Marc van Kreveld and Bettina Speckmann. On rectangular cartograms. Technical report, institute of information and computing sciences, utrecht university, December 2004. UU-CS-2004-040, http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-2004/2004-040.pdf.

[127] Marc van Kreveld and Bettina Speckmann. On rectangular cartograms. *Computational Geometry: Theory and Applications*, to appear.

[128] W. N. Venables and B. D. Ripley. *Modern Applied Statisitc with S*. Springer, 2002.

[129] Matthew O. Ward. XmdvTool: Integrating Multiple Methods for Visualizing Multivariate Data. In *IEEE Conf. on Visualization '94*, pages 215–223, 1994. http://davis.wpi.edu/~xmdv.

[130] Colin Ware. *Information Visualization - Perception for Design*. Morgan Kaufmann, 1st edition, 2000.

[131] Alan Watt. *Computergrafik*. Addison-Wesley, 3rd edition, 2002.

[132] Bin Wei, Claudio Silva, Eleftherios Koutsofios, Shankar Krishnan, and Stephen North. Visualization research with large displays. *IEEE Computer Graphics and Applications*, pages 50–54, July/August 2000.

[133] N. Weisstein. *The Joy of Fourier Analysis*. Erlbaum, Hillsdale, NJ, 1980.

[134] Ben White, Ian Gregory, and Humphrey Southall. Analysing and visualising long-term change. *GIS Research UK, 6th National Conference*, 1998.

[135] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley, 3rd edition, 1999. http://www.opengl.org/developers/documentation/.

Index