First publ. in: Information visualization 2 (2003), 1, pp. 58-67

Visualizing geographic information: *VisualPoints vs CartoDraw*

Daniel A. Keim[1],
Stephen C. North[2],
Christian Panse[3],
Jörn Schneidewind[4]

[1]*University of Constance, Germany;* [2]*AT&T Shannon Laboratory, Florham Park, NJ, USA;* [3]*University of Constance, Germany;* [4]*University of Halle, Germany*

Correspondence:
Professor Dr. Daniel A. Keim, Computer Science Institute, Universität Konstanz Fach D78, Universitätsstr. 10, D-78457 Konstanz, Germany.
Tel: +49 7531 88 3161; Fax: +49 7531 88 3062;
E-mail: keim@informatik.uni-konstanz.de

Abstract
Cartograms are a well-known technique for showing geography-related statistical information, such as population demographics and epidemiological data. The basic 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 paper, we deal with the problem of making continuous cartograms that strictly retain the topology of the input mesh. We compare two algorithms that solve the continuous cartogram problem. The first one uses an iterative relocation of vertices based on scanlines. This algorithm explicitly accounts for induced shape error. The second one is based on the Gridfit technique, which uses pixel-based distortion based on a quadtree-like data structure. The basic idea is to insert pixels, the number of which corresponds to a statistical parameter, into the data structure and distort the pixels such that every pixel obtains a unique, nonoverlapping position. Relocation of vertices of the map are positioned using the same distortion. We discuss the results obtained from both methods, compare their shape and area trade-offs as well as their efficiency, and show results from different applications.

Keywords: Cartograms; visualization of geographic information

Introduction

Cartograms are a powerful way of visualizing geography-related information. A cartogram is a generalization of an ordinary map, which is distorted by resizing its regions by a geographically related input parameter. Example applications in the literature include population demographics,[1] election results,[2] and epidemiology.[3] Since cartograms are difficult to make by hand, the study of automated methods is of interest.[1,4–8]

Cartograms can also be seen as a general information visualization technique. They provide a means for trading shape against area to improve a visualization, by scaling polygonal elements according to an external parameter. In population cartograms, by allocating more area to densely populated areas, patterns that involve many people are highlighted, while those involving fewer people are emphasized less. Figure 9(a) shows a conventional map of the 2000 U.S. presidential elections along with two population-based cartograms representing the same information. The two cartograms were generated using the two methods compared in this study. In the cartogram, the area of the states is scaled to their population, and reveals in that way the close result of a presidential election more effectively than the professionally designed map in Figure 9(a). For a cartogram to be effective, a human being must be able to understand quickly the displayed data and relate it to the original map. Recognition 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 even just to retain the original map's topology.[9] Even allowing for errors in shape and area representations, we are left with a difficult simultaneous optimization problem for which currently available algorithms are very time-consuming.

The cartogram problem

Cartogram generation can be defined as a map deformation problem. The input is a planar polygon mesh (map) and a set of values, one for each region. The goal is to deform the map so that the area of each region matches the value assigned to it, and in such a way that the overall shape of the regions is preserved well enough for them to be recognizable.

Problem. The Cartogram Problem.

Input: A planar polygon mesh $\mathscr{P}$ consisting of polygons $p_1, \ldots, p_k$, values $\chi = x_1, \ldots, x_k$ with $x_i > 0$, $\sum x_i = 1$. 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 $\bar{\mathscr{P}}$ consisting of polygons $\bar{p}_1, \ldots, \overline{p_k}$ such that the function $f(\bar{S}, \bar{A})$ is minimized with

$$\bar{S} = \{s_1, \ldots, s_k\}, \text{ where } s_i = d_S(p_i, \bar{p}_i) \quad (\text{Shape error}),$$

$$\bar{A} = \{a_1, \ldots, a_k\}, \text{ where } a_i = d_A(x_i, A(\bar{p}_i)) \quad (\text{Area error}).$$

The function $f(\bar{S}, \bar{A})$ can be used as $f(\bar{S}, \bar{A}) = c_1 \sum_{i=1}^{k} s_i + c_2 \sum_{i=1}^{k} a_i$, where c_1 and c_2 are constant weights. Intuitively, topology preservation means that the faces of the input mesh must stay the same, that is, the cyclic order of adjacent edges in $\mathscr{P}$ must be the same as in $\bar{\mathscr{P}}$. This can be expressed formally by saying that the pseudo-duals (The *pseudo-dual* of a planar graph is a graph that has one vertex for each face and an edge connecting two vertices if the corresponding faces are adjacent.) of the planar graphs represented by $\mathscr{P}$ and $\bar{\mathscr{P}}$ should be isomorphic. It is likely that even simple variants of the cartogram problem that involve 2-D comparison is NP-complete.

Since it may be impossible to fulfill the area and shape constraints simultaneously, the functions $f(\cdot, \cdot)$, $d_S(\cdot, \cdot)$, and $d_A(\cdot, \cdot)$ model the error in an output cartogram. We discuss possible forms of these functions in more detail in the section The *CartoDraw* solution.

Previous work

Several families of cartogram generators are described in the literature. They range from trivial noncontiguous cartograms that merely scale and display disconnected polygons, to sophisticated solutions that apply non-linear transformations or techniques from computational geometry to distort a map without breaking its topology. Examples of the latter include the conformal maps proposed by Tobler,[1] the radial expansion method of Selvin *et al.*,[5] the rubber sheet method of Dougenik *et al.*,[6] the line integral method of Guseyn-Zade and Tikunov,[7] the 'piezopleth' method of Cauvin *et al.*,[11] and Dorling's[11] cellular automaton approach. Very similar drawings can also be achieved by non-linear magnification.[12–15]

In Figure 1, we provide four examples of population cartograms. The first is a non-continuous cartogram that scales each polygon to the desired size but does not retain the input map's topology (see Figure 1(a)). While non-continuous cartogram are easy to generate, they do not increase the size of small polygons and are therefore of limited use in understanding data. The second example is a pseudo-cartogram[8] that expands the map along lines of longitude and latitude to achieve a least root-mean-square area error (see Figure 1(b)). The third example applies a non-linear, topologypreserving mesh transformation technique[16] (see Figure 1(c)). None of the methods mentioned so far captures an explicit notion of shape-preservation. In contrast, the force-based approach in the fourth example alternately optimizes shape and area error[2] by a non-linear optimization process (see Figure 1(d)). Although its results are better than most other methods, the complex optimization algorithm has a prohibitively high execution time of about 18 h for a modest-sized map with 744 vertices.

Our contribution

This study compares two new methods for generating cartograms. The first one is a recently proposed scanline-based local repositioning of vertices.[9] In essence, this approach uses a line drawn through the map as a hint for the direction in which to extend or contract polygons, while explicitly computing shape error. A series of such local improvements using different scanlines leads to the final cartogram. The second approach is based on the *Gridfit* technique implemented in the *VisualPoints* system.[17] *Gridfit* applies pixel-based distortion to a quadtree-like data structure. In this study, we show how the *Gridfit* technique can be extended to cartogram construction. The basic idea is to insert some number of pixels, corresponding to the desired area parameter, into the data structure and distort the pixel placement such that every pixel is assigned to a unique non-overlapping position. The vertices of the map are then repositioned using the same distortion. We show how different pixel insertion algorithms can yield different cartograms.

Algorithm 1. Cartogram $(\mathscr{P}, \chi)$

```
SetOfSL=GetScanlines {automatically or interactively}
P' = P
repeat
  AreaErr=AreaError(P, P', χ);
  for all SL ∈ SetOfSL do
    {in order of area error reduction potential}
    P̄=ProcessSL(P', X̃, SL)
    if Topology(P̄) && (ShapeError(P', P̄) ∈ s then
until (AreaErr − AreaError (P, P', χ) ≤ ∈)
```

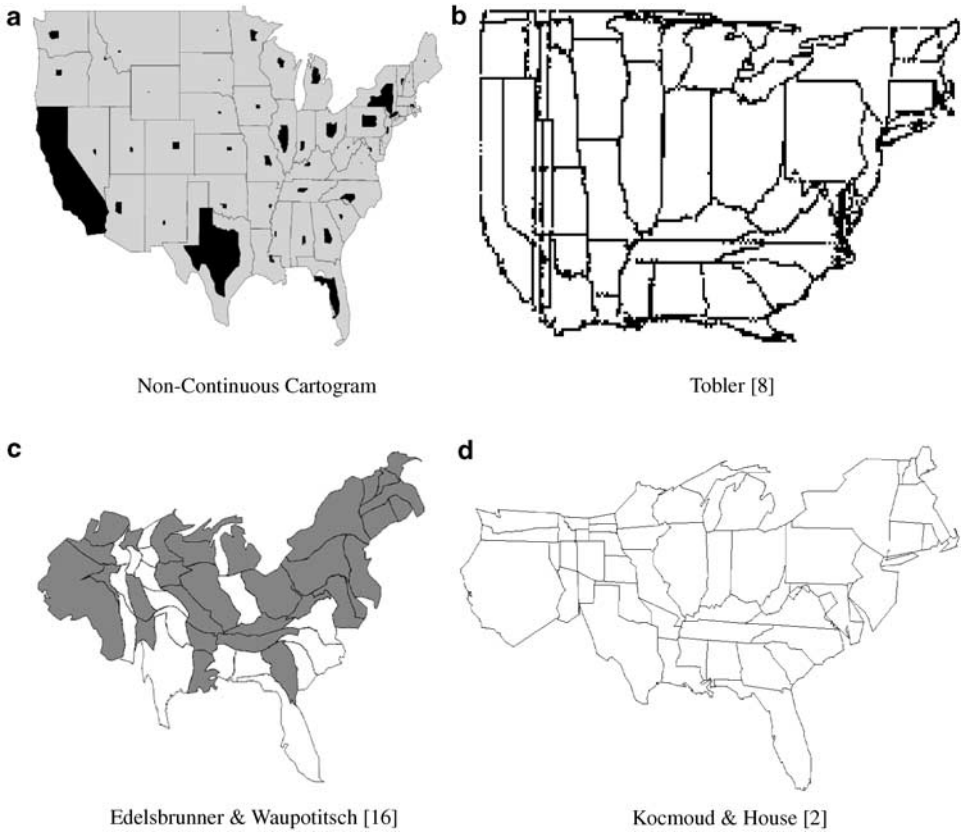


Figure 1 Cartogram drawing methods: (a) non-continuous cartogram, (b) Tobler,[8] (c) Edelsbrunner and Waupotitsch,[17] and (d) Kocmoud and House.[2]

Our main objective is to compare these methods.[18] We provide a detailed analysis of the area and shape error trade-offs of each and compare their efficiency. We also apply them to several different problems including population, election and telephony data.

The paper is organized as follows. In the following section we briefly review scanline-based cartogram generation. In the section thereafter we describe how the *Gridfit* technique can generate cartograms, and present several variants of pixel insertion. The penultimate section presents a comparison using the visualization of several real application data sets. We conclude with open questions and ideas for future work in the last section.

The *CartoDraw* solution

CartoDraw was recently proposed as a practical approach to cartogram generation.[9] In this section, we outline its main ideas and some useful variations.

The *CartoDraw* algorithm

The basic idea of *CartoDraw* is to incrementally reposition the vertices of the map's polygons by means of scanlines. Local changes are applied if they reduce total area error without introducing excessive shape error. Scanlines may be determined automatically, or entered interactively (see the section on manual *vs* automatic scanlines). The main search loop over the scanlines is presented in Algorithm 1. For each, it computes a candidate transformation of the polygons, and checks it for topology and shape preservation. If the candidate passes the tests, it is made persistent; otherwise it is discarded. The scanline processing order depends on their potential for reducing area error. The algorithm runs until the area error improvement over all scanlines falls below a threshold $\in$. Observe that in processing an individual scanline, the algorithm is allowed to increase the area error to escape local minima. However, in each iteration of the repeat-until loop, the area error decreases monotonically, so termination is guaranteed.

The area error *AreaErr* is the sum of the single polygon area errors. In the simplest case, the single polygon area error function d_A is the norm of the difference of desired and actual area. If the v_i and $A(p_i)$ (see The cartogram problem section) are normalized $(v_i > 0 \wedge \sum v_i = 1)$; $(\mathrm{A}(p_i) > 0 \wedge \Sigma A(p_i) = 1)$, the *AreaErr* can be determined as

$$AreaErr = \sum_{i=1}^{k} d_A(v_i, A(p_i)) = |v_i - A(p_i)|.$$

Depending on the application, we may also want to weigh the area error such that the error in regions of high interest (high values of v_i) contributes more to the overall error.

Algorithm 2. ProcessScanline$(\mathscr{P}, \chi, SL)$
  for all $cl \in CuttingLines(SL)$ **do**
    {cutting lines on Scanline SL}
    SF=*ScalingFactor*($\{p_i | p_i \cap cl \neq \varnothing\}, \chi$)
    {determines the aggregated scaling factor of all p_i}
  for all $v \in G\mathscr{P}(\mathscr{P})$ **do**
    $v = v + SF \cdot \text{side}(v, cl) \cdot \overrightarrow{SL} \,|\, \overrightarrow{SL}|\text{side}(cl, v) = 1$
    if v is on the left side of cl and -1 otherwise}

The shape of two polygons can be compared several ways. We can approximate the curvature of the polygons by a turning angle algorithm,[19] curvature plots such as geometric hashing[20] or Fourier approximations.[21] Our implementation incorporates a Fourier transformation of the polygons' curvatures. If $\mathcal{C}(p)$ denotes the curvature of a polygon p and $\mathscr{F}(\mathcal{C}(p))$ denotes its Fourier transformation, the shape error d_S can be determined as

$$d_S = \sum_{i=1}^{k} d_{Euclid}(\mathscr{F}(\mathcal{C}(p)), \mathscr{F}(\mathcal{C}(\bar{p}))).$$

Note that for polygons, the Fourier transformation of the curvature can be determined analytically.[21]

Scanline-based local repositioning

The input scanlines are arbitrary, and may be computed automatically or entered interactively (see the immediate subsection). The idea is to use line segments (called *cutting lines*) perpendicular to scanlines at regular intervals. Consider the two edges on the boundary of the polygon intersected by a cutting line on either side of the scanline. These edges divide the polygon boundary into two connected chains. Now, if the area constraints require the polygon to be expanded, the algorithm applies a translation *parallel* to the scanline to each vertex on the two connected pieces of the boundary (in opposite directions) to *stretch* the polygon at that point. Similarly, if a contraction is called for, the direction of translation is reversed. Figure 2 illustrates the approach.

In contrast to the algorithm described by Keim *et al.*,[9] in this study we apply translation only to points on the boundary of the polygon mesh. We use interpolation to remap points in the interior of the subdivision. For this purpose, each interior point has four associated *reference points* on the boundary, determined by the four closest points to the interior point in each of the four quadrants. The location of an interior point is updated with respect to its reference points. Since interior points are connected by straight line segments, a candidate update may create an intersection of segments, violating the topology preservation property. Consequently, the algorithm explicitly checks candidate transformations and rejects those causing such intersections.

The processing of a single scanline is presented in Algorithm 2. The function *ScalingFactor* determines if the global polygon is to be stretched or contracted, and how much. It computes the average of the area errors of the polygons intersected by the cutting line, weighted by their scale factors. The algorithm does not calculate the new positions of all global vertices for each cutting line. Rather, it aggregates the distortion vectors for each point and applies the aggregate vector after all cutting lines of a scanline have been considered.

Manual *vs* automatic scanlines

The central *CartoDraw* algorithm is independent of the particular way that scanlines are generated. The *automatic generation of scanlines* employs a fixed grid of horizontal and vertical scanlines (see Figure 9(b)). The grid's resolution can be varied, but within reason this has only a minor influence on the result. Since 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 and does not improve 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

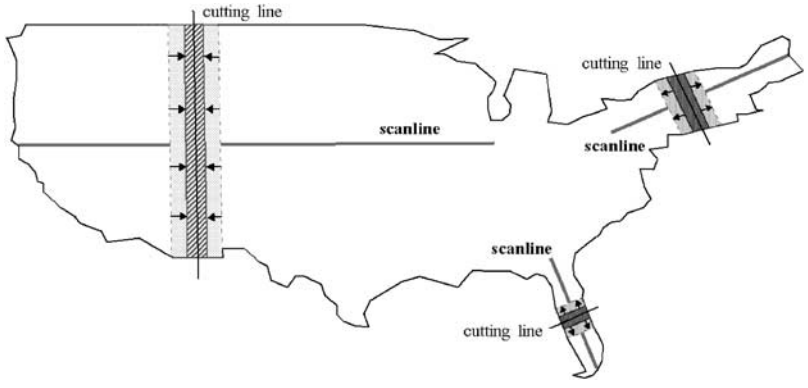


Figure 2 Scanline-based local repositioning.

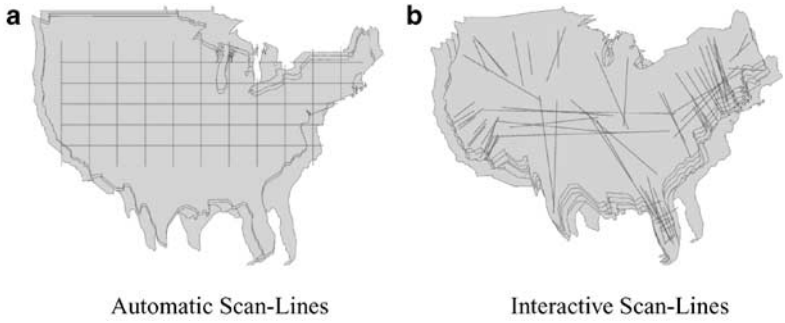


Figure 3 Population cartogram with automatically and interactively placed scanlines: (a) automatic scanlines and (b) interactive scanlines.

for improvement. Automatic placement based on these criteria so far has been difficult to achieve, therefore we allow the user to *interactively position the scanlines* on the current map (Figure 3). The scanlines seem to work best if they are positioned such that they are either parallel or orthogonal to the contour of the global polygon. Figure 9(b) shows an example of a set of manually placed scanlines. Note how parts of the map needing large changes have many scanlines of varying lengths, while other parts have hardly any.

The *VisualPoints* solution

The *VisualPoints* system[17] was developed to address the problem of overplotting spatially referenced data. It works by moving points that would be drawn on already occupied pixels to nearby unoccupied pixels, instead of overplotting them. *VisualPoints* 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 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.

The *VisualPoints* algorithm

In each step of the *VisualPoints* 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 sized subregions, we have to determine new extents 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 4).

Generating cartograms with *VisualPoints*

To adapt the *VisualPoints* technique to cartogram generation, a few changes need to be made to the original algorithm. The modified algorithm is shown in Algorithm 3. The most important changes will be explained in more detail.

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 *VisualPoints* 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 *VisualPoints* algorithm, there is no change in the first step of Figure 4(b), since there is enough space (18 pixels) in the left partition to accommodate all 12 data points. In the modified algorithm, the border shifts proportionately to the number of data points, that is, in a ratio of 12:2 resulting in the partition shown in Figure 5(a). Note that the result of the second step is also different (compare Figures 4(c) and 5(b)).

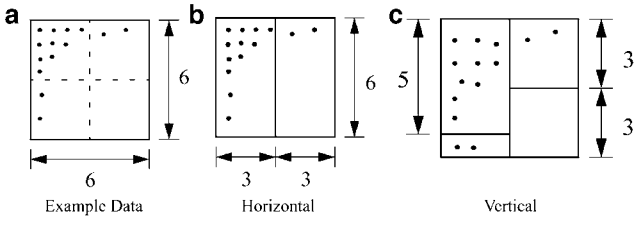


Figure 4 Original *VisualPoints* algorithm: (a) example data, (b) horizontal, and (c) vertical.

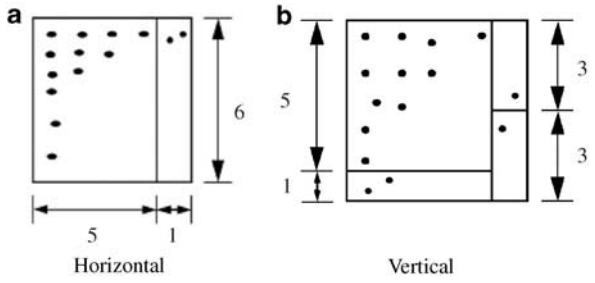


Figure 5 *VisualPoints* algorithm for cartograms: (a) horizontal and (b) vertical.

No pixel placement A second difference between the original and the modified *VisualPoints* algorithm is that pixels do not 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 do not 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,

Algorithm 3. VPcarto$(\mathscr{P}, \chi)$
Quadtree Q; {empty initialized Quadtree}
for all *polygons* $P \in \mathscr{P}$ **do**
point cur=FindStartPoint(P);
while *pc* < *P.DesiredArea*(χ) **do**
{area is represented as pixels}
now=P.ComputeNextPosition(cur);
{depends on insert strategy}
Q.InsertQuadtree(cur);
pc=pc+1;
Trans formQuadtree(Q);
{moves the borders of the quadtree}
for all *polygons* $P \in \mathscr{P}$ **do**
for all points $p \in P$ **do**
qnode node=Q.FindNode(p);
p=scale(node,p);
{depends on new height and width of node}

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(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(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(c)).

Observe that pixels are only used to construct the quadtree-like data structure but are not actually positioned as in case of the *VisualPoints* system, so the exact position of each pixel is not that important. As Figure 6(a–c) shows, the pixel insertion strategy is of great importance for the quality of the resulting cartograms,

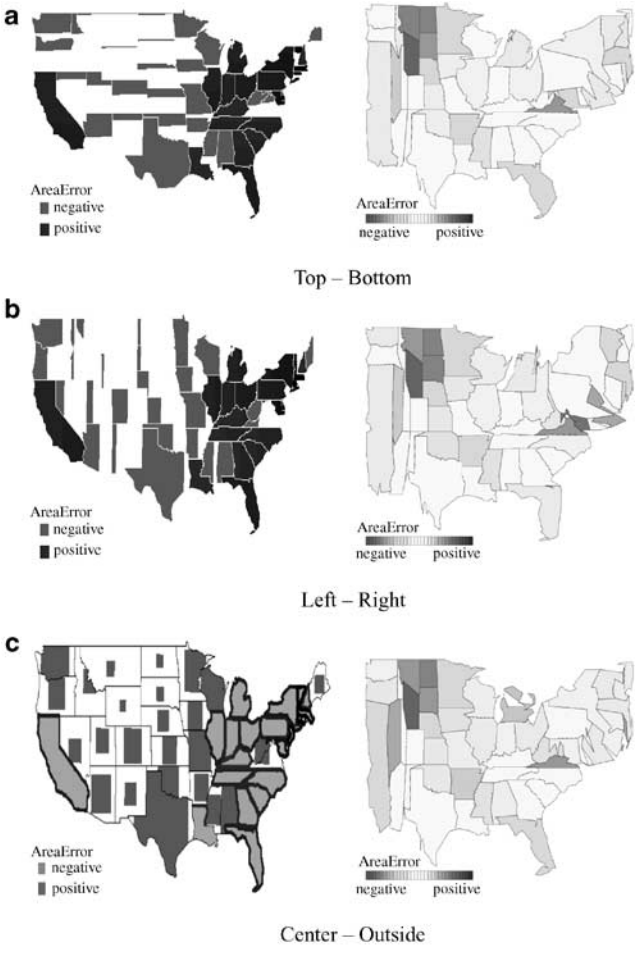


Figure 6 Insertion strategies: (a) top–bottom, (b) left–right, (c) center–outside.

especially with respect to the shape of the polygons and the overlap of 13 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.

Comparison and evaluation

The *CartoDraw* algorithm described in the section on The *CartoDraw* solution was implemented in C++ using the LEDA library[22] and the *VPCarto* algorithm described in the previous section 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.

Efficiency and effectiveness

Figure 7 shows the measured efficiency and effectiveness results. The total run time was 3 s for the new *VisualPoints* approach, 25 s for the automatic scanline approach, and 16 h for the non-linear optimization approach by Kocmoud and House.[2] (The comparison assumes that all algorithms run on a 120 MHz computer.) Note that the scale on the *y*-axes of Figure 7 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 7(b) compares shape *vs* area error for population cartograms made with *VPCarto* and interactive scanlines, measured on the four call volume cartograms of Figure 8. 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 statewise area error, Figure 7(c) shows the shape error by state, sorted by shape error. Figure 7(c) reveals that the *CartoDraw* algorithm consistently provides a lower shape error than the *VPCarto* algorithm.

Application examples

We applied both algorithms to several example data sets. In all figures, the area of the states in the cartograms

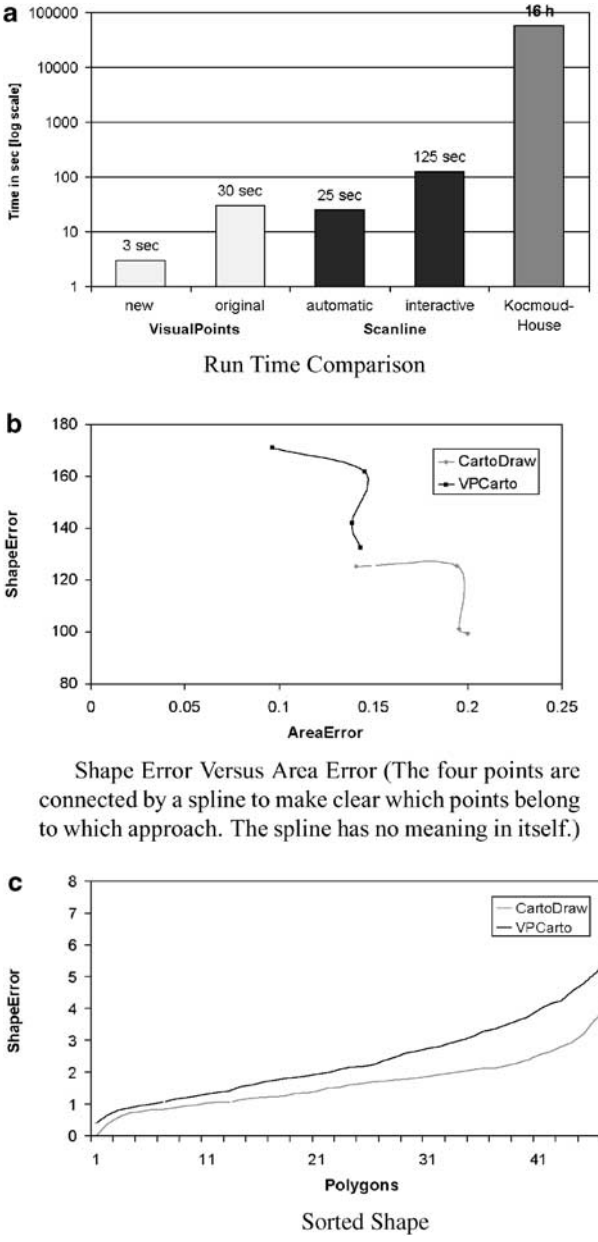


Figure 7 Efficiency and effectiveness results: (a) run-time comparison, (b) shape error *vs* area error (The four points are connected by a spline to make clear which points belong to which approach. The spline has no meaning in itself.), and (c) sorted shape.

corresponds to population and the colors represent the different values. Figure 9(c) shows the U.S. population cartogram with the percentage of the tax paid per capita.[23] Since the area of a state corresponds to the number of inhabitants, the cartograms show how many people and which part of the country has to pay low, medium, or high capita tax rates.

Figure 9(d) visualizes the percentage of uninsured drivers. An average about 14% of the drivers are

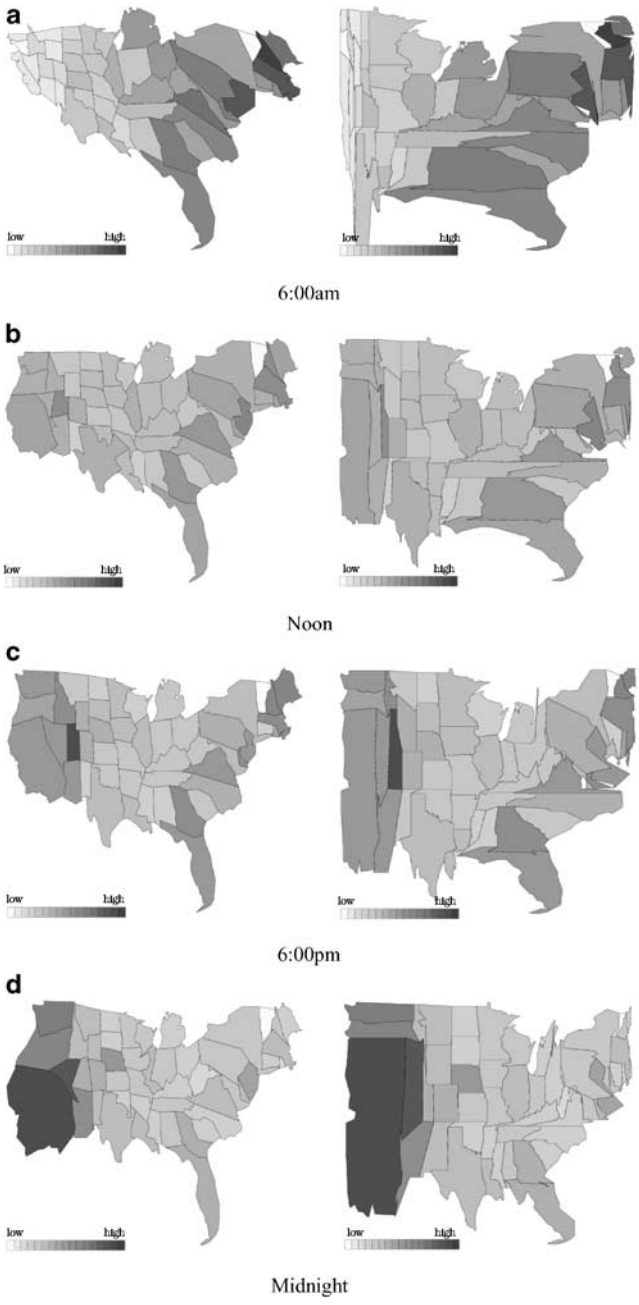


Figure 8 Long-distance call volume data computed with *CartoDraw* (left) and *VisualPoints* (right). The unipolar colormap shows the normalized call volume: (a) 6 a.m., (b) noon, (c) 6 p.m., and (d) midnight.

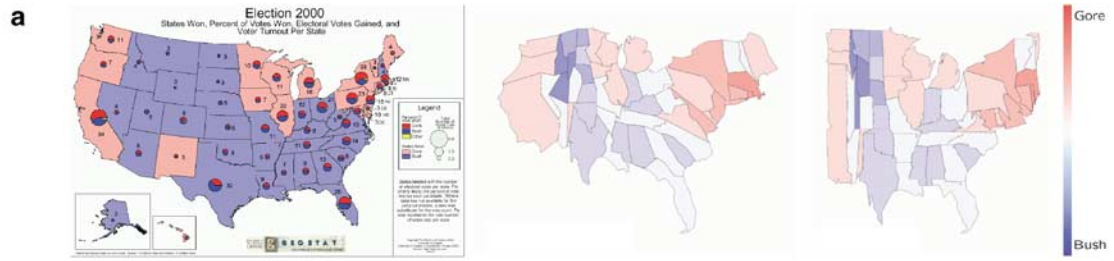
uninsured, but in some states, the number is much higher. The cartogram shows the states with high rates of uninsured drivers and provides an impression of how the uninsured rate depends on the geographical location. In New Mexico and Colorado, for example, the number of uninsured drivers, represented by red color, is very high in contrast to New England states, where the blue color reflects a low percentage of uninsured drivers. Note that the *VPCarto* algorithm provides slightly lower area error, while the *CartoDraw* algorithm provides superior shape preservation. (Notice, for example, shape distortion in New England.)

We also have applied cartograms to monitoring a continuous stream of telephone call volume data. Figure 8 shows the results of the telephone call volume (by origination, normalized by population) at four different times during 1 day. Color is redundantly mapped to the normalized call volume, with brighter colors corresponding to smaller call volumes. The resulting visualizations clearly reflect the different time zones of the US, and show interesting patterns of phone usage as it changes during the day. For example, we see the western part of the country shrink in size in the early part of the day (6 a.m. EST) and slowly increase in size as the day goes on, reflecting increasing traffic originated in that part of the country. It is interesting that the call volume is especially high in the morning and in the evening (see Figures 8(a–e) and (d–h): 6 a.m. on the east coast and midnight on the west coast), while it is slightly lower during the day. Again, the *VPCarto* algorithm has a slightly lower area error, while the *CartoDraw* algorithm (interactive scanlines) provides a better shape preservation. The evaluation and comparison shows that both approaches have their advantages and disadvantages: while *CartoDraw* is superior in shape preservation, it needs significantly more run time and yields somewhat higher area error. In contrast, *VPCarto* runs in interactive time 18 independent of the number of polygons involved, but does not deal with the shape of the polygons and therefore problems with local edge crossings can occur.

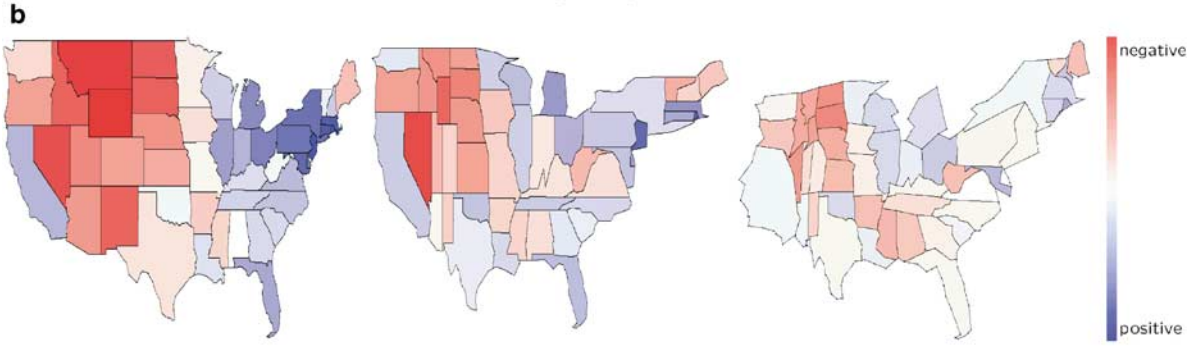
Conclusions and future work

We analyzed and discussed the problem of efficient cartogram drawing, and proposed a new cartogram drawing heuristic based on the *VisualPoints* algorithm. The new algorithm offers improved area error results and significantly less running time, at some cost in the final shape error. Experiments show that both algorithms offer good results for a variety of applications, and the speed of the new algorithm allows interactive animation of online data for maps of many dozens of polygons.

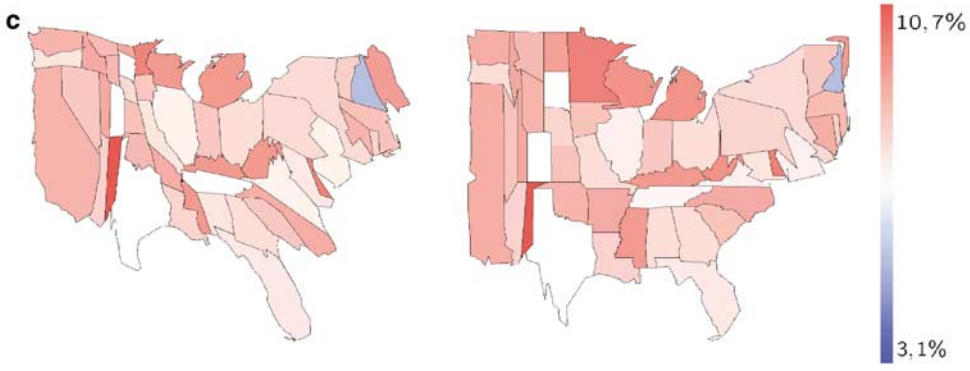
While the proposed algorithms are a significant step in fast, reliable, and effective cartogram generation, there are many potential directions for further improvements of this work. One promising area is improve shape preservation in the *VPCarto* approach, possibly by starting with a *CartoDraw* map generated by automatic scanlines and further refining it using the *VPCarto* approach. An important related question is how to better determine the placement of scanlines automatically. It is possible that this could open the way to achieve performance comparable to that of the best aspects of both heuristics.



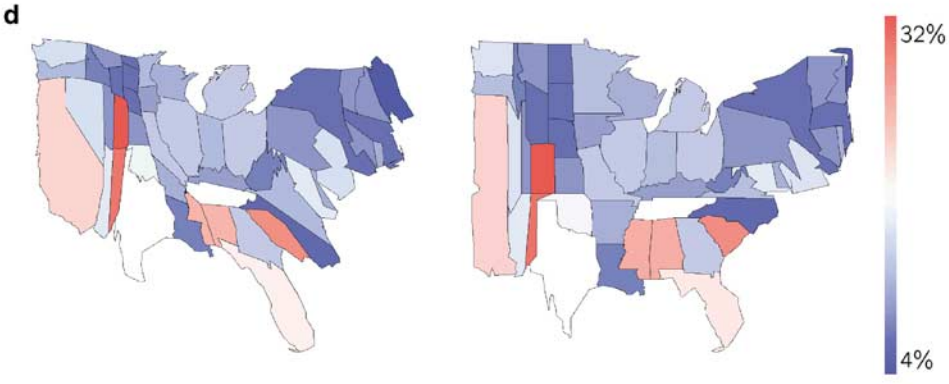
Election 2000: Traditional Map Design [24], *CartoDraw*, *VisualPoints*



Orig. Area Error US map ($E_{rel} = 0.4$), Automatic Scanlines ($E_{rel} = 0.21$), Int. Scanlines ($E_{rel} = 0.1$)



Percentage Tax Paid Per Capita [23]



Percentage Uninsured Drivers [25]

Figure 9 Population 2000 cartograms computed with *CartoDraw* (left) and *VisualPoints* (right) helps to demonstrate several data.

References

1 Tobler WR. *Cartograms and cartosplines*. Proceedings of the 1976 Workshop on Automated Cartography and Epidemiology (Washington, DC, 1976), 53–58.
2 Kocmoud CJ, House DH. *Continuous cartogram construction*. Proceedings of the IEEE Visualization (Research Triangle Park, NC, 1998), 197–204.
3 Gusein-Zade S, Tikunov V. Map transformations. *Geography Review*, 1995; **9**: 19–23.
4 Dent BD. *Cartography: Thematic Map Design*, 4th edn, Chapter 10, William C. Brown: Dubuque, IA, 1996.
5 Selvin S, Merrill D, Schulman J, Sacks S, Bedell L, Wong L. Transformations of maps to investigate clusters of disease. *Social Science and Medicine* 1988; **26**: 215–221.
6 Dougenik JA, Chrisman N, Niemeyer DR. An algorithm to construct continuous area cartograms. *The Professional Geographer* 1985; **37**: 75–81.
7 Gusein-Zade S, Tikunov V. A new technique for constructing continuous cartograms. *Cartography and Geographic Information Systems* 1993; **20**: 66–85.
8 Tobler WR. Pseudo-cartograms. *The American Cartographer* 1986; **13**: 43–40.
9 Keim DA, North SC, Panse C. Cartodraw: a fast algorithm for generating contiguous cartograms. IEEE TVCG 2003, to appear.
10 Cauvin C, Schneider C, Cherrier G. Cartographic transformations and the piezopleth method. *The Cartographic Journal* 1989; **26**: 96–104.
11 Dorling D, *Area Cartograms: Their Use and Creation*, 1st edn. Department of Geography, University of Bristol: England, 1996.
12 Keahey T, Robertson E. *Nonlinear magnification fields*. Proceedings of the IEEE Symposium on Information Visualization (Phoenix, AZ, 1997), 51–58.
13 Munzner T. Exploring large graphs in 3D hyperbolic space. *IEEE Computer Graphics & Applications* 1998; **18**: 18–23.
14 Carpendale MST, Cowperthwaite DJ, Tigges M, Fall A, Fracchia FD. *The TARDIS: a visual exploration environment for landscape dynamics*. Visual Data Exploration and Analysis VI, Proceedings of the SPIE, Vol. 3643 (San Jose, CA, January 1999), 110–119.
15 Alan Keahey T. *Area-normalized thematic views*. Proceedings of the International Cartography Assembly (Ottawa, Canada, August 1999).
16 Edelsbrunner H, Waupotitsch R. A combinatorial approach to cartograms. *Computational Geometry* 1997; **7**: 343–360.
17 Keim DA, Herrmann A. *The gridfit algorithm: an efficient and effective approach to visualizing large amounts of spatial data*. Proceedings of the IEEE Visualization (Research Triangle Park, NC, 1998), 181–188.
18 Keim DA, North SC, Panse C, Schneidewind J. *Efficient cartogram generation: a comparison*. Proceedings of the IEEE Information Visualization 2002 (Boston, Massachussets, 2002), 33–36.
19 Horn P, Berthold K. *Robot Vision*. MIT Press: Cambridge, MA, 1986.
20 Rigoutsos I, Hummel R. Massively parallel model matching: geometric hashing on the connection machine. *IEEE Computer* 1992; **25**: 33–42.
21 Berchtold S, Keim DA, Kriegel H-P. Using extended feature objects for partial similarity retrieval. *VLDB Journal* 1997; **6**: 333–348.
22 Mehlhorn K, Näher S. *The LEDA Platform of Combinatorial and Geometric Computing*, 1st edn. Cambridge University Press: Cambridge, 1999, http://www.mpi-sb.mpg.de/~mehlhorn/LEDAbook.html.
23 HSH Home Plans. http://homeplans.hsh.com/articles/taxes/state-tax-percapita.asp, December 2002.
24 History Central, www.multied.com/elections, March 2002.
25 CNBC. http://www.moneycentral.msn.com/articles/insure/basics/6292.asp, December 2002.