Algorithms for Cartogram Animation*

Min Ouyang Peter Revesz
Computer Science and Engineering Department
University of Nebraska-Lincoln
Lincoln, NE68588, USA

Abstract

We describe several value-by-area cartogram animation algorithms that can be used to visualize geographically distributed continuous spatiotemporal data that often occur in GIS systems. We implemented the algorithms as part of the graphical user interface of the MLPQ/GIS database system.

1. Introduction

Many GIS databases contain spatiotemporal data such that the geographically distributed value change continuously by time. Population and precipitation distributions are two such examples. Value-by-area cartograms provide a highly expressive visualization for this kind of data. A value-by-area cartogram is created by dividing the original map into small areas (cells) and then enlarge or shrink each cell to make its area proportional to its given "value". For example, a value-by-area cartogram where the cells are the continental U.S. states and the values are their populations in 1990 is shown in Figure 1. In this value-by-area cartogram, each state's area is proportional to its population. By looking at the map, we can easily see the population distribution in the continental United States.

For continuously changing data such as population or precipitation data the value-by-area cartogram animation successively displays value-by-area cartograms at different time instances. Such animations can reveal more information than is revealed by only a few selected value-by-area cartogram snapshots. For example, we did an animation for the monthly precipitation in the continental U.S. states between 1948 and 1998. The animation revealed that the precipitation has more regular cycles in the New England states than in the western states. Such animations allow everyone to make similar observations without even knowing anything about statistics.

*This research was supported in part by NSF grant IRI-9625055 and a Gallup Research Professorship.

Figure 1. A Value-by-Area Cartogram for the U.S. Population in 1990

In this paper, we propose several methods for value-by-area cartogram animation. Each animation method is based on a new algorithm for creating single value-by-area cartograms. Computer experiments show that our cartogram algorithm works much faster than previous cartogram algorithms [2]. We implemented the animation algorithms and integrated them into the MLPQ/GIS constraint database system[5].

The rest of the paper is structured as follows. Section 2 discusses three different methods for creating sequences of value-by-area cartograms as well as a new fast value-by-area cartogram algorithm. Section 3 gives implementation results. Section 4 discusses related works. Finally, Section 5 concludes with possible directions for future work.

2. Animation with Large Number of Snapshots

Some cartogram animation models [6, 8] pre-compute and save the snapshots for display. Each of these models contains only a few cartogram snapshots, thus they can be pre-computed and saved for display. However, if the animation contains a large number of snapshots, then the pre-computation strategy is not practical. A better way is to

compute the snapshots during animation. But this strategy requires fast algorithms for snapshot computation.

2.1. Value-by-Area Cartogram Animation Methods

We describe three basic methods for value-by-area cartogram animation, which provides a visualization of geographically distributed continuous spatiotemporal data.

Each animation method uses as a basic procedure some value-by-area cartogram snapshot algorithm. Each snapshot algorithm takes as input a map with some cell division and values for each cell and gives as output a distorted map in which each cell has the same cell density (the "density" is the result of cell value divided by cell area). Independent of the basic snapshot procedure used, the three animation methods work in different ways.

Parallel Method: For each time instance t to be displayed this method takes the original map with the cell values at time t and gives an output map or snapshot. This is called the parallel method because the snapshots can be computed in parallel (see Figure 2).

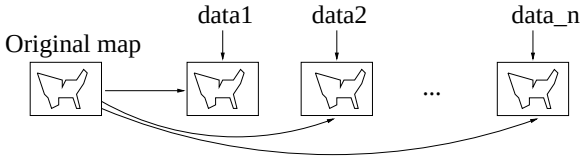

Figure 2. Constructing cartogram snapshots using Parallel Method

Serial Method: In this method each cartogram snapshot is constructed from the previous cartogram snapshot (except for the first cartogram snapshot, which is constructed from the original map) as shown in Figure 3.

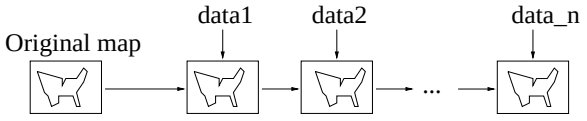

Figure 3. Constructing cartogram snapshots using Serial Method

Hybrid Method: This method combines the previous two methods. It generates each $k \times n$th snapshot from the original map, and all the other snapshots from the previous snapshots (see Figure 4).

Comparison: The Parallel Method is easily parallelized because each snapshot can be computed independently, while the Serial Method is not immediately parallelizable.

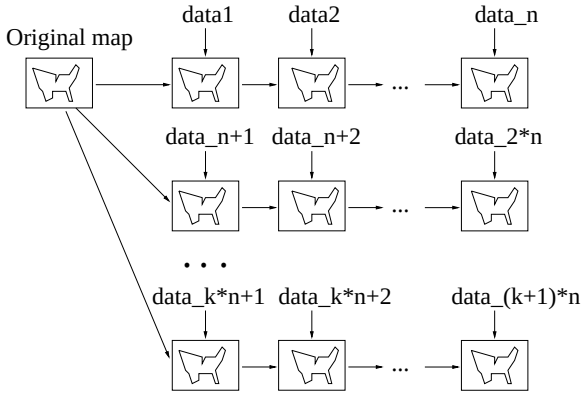

Figure 4. Constructing cartogram snapshots using Hybrid Method

However, the Parallel Method is slower than the Serial Method in a single processor computer because there is usually a bigger difference between the original map and a snapshot than between two consecutive snapshots, hence calculating the snapshot from the original map requires generally more time than calculating it from the previous snapshot.

For the Serial Method the cartogram snapshot quality is not as good as for the Parallel Method, because in the Serial Method the previous snapshot will transmit any possible the cell distortion error to its successor and hence the cell shape distortion may accumulate.

The Hybrid method overcomes the cell shape distortion accumulation problem inherent in the Serial Method, while being almost as fast as the Serial Method. Table 1 summarizes the discussions on three methods.

Category	Parallel Method	Hybrid Method	Serial Method
Speed	Slow	Fast	Fastest
Easily Parallelizable	Yes	Yes	No
Shape Distortion Accumulates	No	Not much	Yes

Table 1. Comparison of three Snapshots Construction Methods

2.2. A New Value-by-Area Cartogram Algorithm

Among previous value-by-area cartogram algorithms the rubber-sheet based algorithms of [2, 3] produce the most

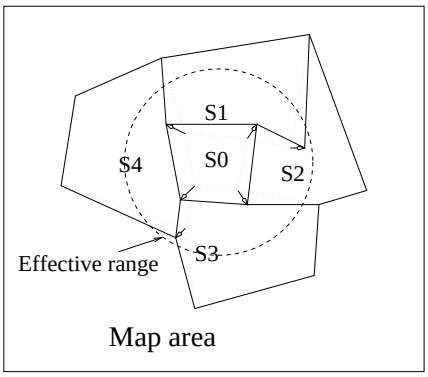

Figure 5. Effective Range

accurate cartograms although they may be slow. We review and improve the speed of these algorithms below.

In both rubber-sheet transformations [2, 3] the map is divided into small cells (polygons). Each cell has a "value" which describes the size of its desired area. The cell is inflated (if its actual area is smaller than the desired area) or deflated (if its actual area is larger than the desired area). An iterative process to inflate or deflate the cells is done until the difference between the actual cell area and the desired cell area is less than some error tolerance value.

In order to prevent holes in the map or overlaps among the cells, in [2, 3] the inflation/deflation of any cell influences all the corner vertices of the cells in the map. The two algorithms calculate slightly differently the influence values, but in both the further the corner vertex is from the center of the inflated/deflated cell the less it is influenced or changed.

In the computation of inflation/deflation influences on the corner vertices of cells, instead of computing each influence, it is practical to ignore some small influences. Suppose that a map with left bottom $(0, 0)$ and right top $(100, 100)$ is divided into 100×100 equal size square cells. If the cell with left bottom $(50, 50)$ and right top $(51, 51)$ inflates 1%, then the influence on a vertex say at $(52, 53)$ is about 0.0005, while the influence on a vertex at $(1, 2)$ is about 0.00002, which may be too small for consideration.

The influence of a particular cell's inflation/deflation on the corner vertices in the map decreases with the distance from the cell center to this corner vertex and increases with the percent the cell is inflated/deflated. Therefore, for any particular cell inflation/deflation, we can have an effective range. If a corner vertex is out of a cell inflation/deflation effective range, the change on this vertex will be too small for consideration and can be ignored.

For example, in Figure 5 the dotted lines represent the original cell division and $S0$ is inflated. Here the effective range is shown by the dashed cycle. The corner vertices within the effective range are changed resulting in the new cell division shown in solid lines.

VALUE-BY-AREA CARTOGRAM ALGORITHM

Input: A map with n cells and a value for each cell, cell area percent error tolerance ϵ,
Output: New coordinates for cell vertices that make them form a value-by-area cartogram

begin
  for $i = 1$ to n **do**
    Compute AD_i, the desired area of the ith cell
  end for
  repeat
    for $i = 1$ to n **do**
      Compute AC_i, the area of the ith cell,
      (x_i, y_i), the center of the ith cell, and
      $e_i = \frac{abs(AC_i - AD_i)}{AD_i}$, the percent area error
      if $e_i \geq \epsilon$ **then**
        $r_{eff} = \frac{100 * abs(AD_i - AC_i)}{\sqrt{\pi AC_i}}$
        for $j = 1$ to n **do**
          $d_j = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
          if $d_j \leq r_{eff}$ **then**
            for each corner vertex (x, y) in cell j **do**
              $d = \sqrt{(x - x_i)^2 + (y - y_i)^2}$
              if $d \leq r_{eff}$ **then**
                if $d \leq \sqrt{AC_i/\pi}$ **then**
                  $x = x_i + (x - x_i)\sqrt{\frac{AD_i}{AC_i}}$
                  $y = y_i + (y - y_i)\sqrt{\frac{AD_i}{AC_i}}$
                else
                  $x = x_i + (x - x_i)\frac{AD_i - AC_i}{2\pi d^2}$
                  $y = y_i + (y - y_i)\frac{AD_i - AC_i}{2\pi d^2}$
                endif
              endif
            end for
          endif
        end for
      endif
    end for
    for $i = 1$ to n **do**
      Recompute AC_i and e_i
    end for
  until $(\forall i,\ e_i < \epsilon)$
end

In the value-by-area cartogram transformation (see pseudocode above), if a cell $S0$ with center coordinate (x_0, y_0) inflates/deflates, then instead of computing the distances from (x_0, y_0) to each of the corner vertices (x_i, y_i) in the map to see if (x_i, y_i) is inside the effective range, we can simply compute the distance d_i, which is the distance from

(x_0, y_0) to the center of cell S_i. We can use the distance d_i to approximately represent the distances from (x_0, y_0) to each of the corner vertices of cell S_i in deciding if a corner vertex of cell S_j is inside the effective range. This will decrease the computational time if each cell has a lot of corner vertices.

At the beginning of the algorithm there is a **for** loop which computes the desired area AD_i for each cell i, followed by a **repeat** loop, which does the value-by-area cartogram transformation.

Inside the **repeat** loop are four **for** loops. The first **for** loop computes the cell area AC_i, cell center (x_i, y_i) and the precent cell area error e_i for each center. If e_i is greater than the error tolerance ϵ, then inside the first **for** loop, the second **for** loop computes the radius of the effective range r_{eff} based on AD_i and AC_i. Then the second **for** loop computes the distance d_j as described before. If d_j is less than r_{eff}, then the third loop transforms the corner vertices of cell S_j which are inside the effective range. The goal of the fourth **for** loop inside the **repeat** loop is to compute the percent cell area error between the cell area and the desired cell area for each cell. Those are used to control the termination of the algorithm. The algorithm terminates if all the percent error between cell area and desired cell area is less than the error tolerance ϵ.

3. Implementation Results

We implemented in Visual C++ the algorithms and did some experiments on a 450 MHz Pentium-Pro PC with 128MB memory running Windows/NT.

3.1. Runtime Comparisons for Different Animation Methods

Table 2 gives the run time results for three problems: daily mean temperature, daily temperature spread, and monthly precipitation of the United States. Four strategies are used for creating animations. ParD is the Parallel Method with Dougenik et al.'s cartogram algorithm, ParN is the Parallel Method with the new cartogram algorithm, HybN is the Hybrid Method with the new cartogram algorithm the SerN is the Serial Method with the new cartogram algorithm. The temperature and precipitation data are from the website of the United States National Climatic Data Center ($http://www.ncdc.noaa.org$). For daily mean temperature and daily temperature spread animation, the periods are from January 1, 1993 to December 31,1994. For monthly precipitation animation, the period is from January 1948 to December 1997.

Table 2 shows that the Hybrid Method with the new cartogram algorithm runs much faster than the Parallel Method with Dougenik et al. cartogram algorithm.

Problem	Snapshots	Run time (seconds)			
		ParD	ParN	HybN	SerN
DMT	730	343	89	44	N/A
DTS	730	353	136	56	N/A
MPR	600	287	135	50	44

Table 2. Computational times for different problems (DMT: daily mean temperature. DTS: daily temperature spread. MPR: monthly precipitation)

We did several experiments to see the runtime results with different divisions of the map. Generally, the more cells a map has, the more time is required for finding the value-by-area cartogram. Figure 6 shows the relationship of the run time and the number of cells.

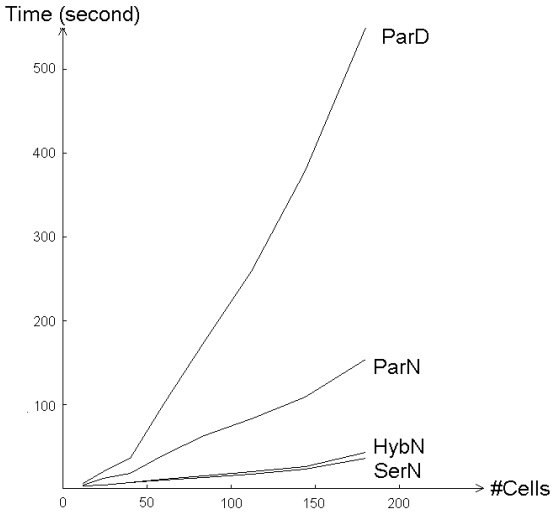

Figure 6. Runtime for different number of cells

The average number of edges for the polygonal cells will also affect the computation runtime, but is generally not as much as the number of cells. Figure 7 shows the relationship between the animation time and the average number of edges for cells when the number of cells dividing a map is fixed.

Figures 6 and 7 together suggest experimentally that Dougenik et al.'s algorithm has an average case complexity of $O(m \times n^2)$ while our cartogram algorithm has approximately $O(m \times n)$ complexity, where n is the number of cells and m is the average number of edges for the cells.

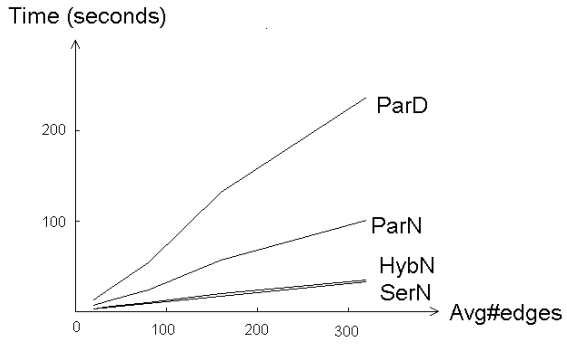

Figure 7. Runtime for different number of average number of cell edges

In particular, Figure 7 shows that they all grow linear in m, while Figure 6 shows that Dougenik et al.'s algorithm grows quadratically while our cartogram algorithm grows approximately linearly in n.

3.2. Accuracy of the Algorithms

Dougenik et al.'s algorithm runs slowly. The Serial Method runs fast but due to shape distortion accumulation it gives poor cartogram snapshots or even totally distorts the cartogram snapshots during the animation. Thus neither of them is suitable for animation.

The Parallel and the Hybrid Methods with new cartogram algorithm give highly accurate cartograms and run fast. More specifically, the Hybrid Method often runs faster and gives more accurate cartogram snapshots than the Parallel Method. Figure 8 shows the error comparison for the Parallel Method and the Serial Method during daily mean temperature animation.

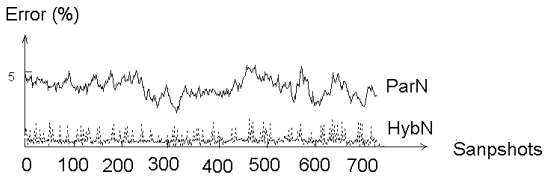

Figure 8. Area error comparison for Parallel Method and Hybrid Method

4. Related Work

Beside [2, 3] there are a number of other value-by-area cartogram snapshot algorithms [1, 4, 7]. Kocmoud and House [6] give an animation for the U.S. population cartograms from 1900 to 1996 and White et al. [8] give an animation for the infant mortality in the United Kingdom from 1856 to 1925. However, the drawback of both methods is that they only consider the parallel method of animation and require pre-computing all the snapshots, because they use much slower value-by-area cartogram algorithms than our algorithm. They can get away with this because their animations contain only a small (less than eleven) number of snapshots.

5. Conclusions

We presented a fast value-by-area cartogram transformation algorithm that can be used as a subroutine within animation algorithms. We also presented novel serial and hybrid animation methods that further enhance the speed of the animation while avoiding excessive cell shape distortion accumulation. The speed of our animation methods avoids the need to pre-compute all the snapshots. We also added the hybrid animation method as a user callable function within the MLPQ/GIS [5] user interface.

References

[1] D. Dorling. Visualizing Changing Social Structure from a Census. *Environment and Planning*, 27:353–378, 1995.

[2] J. A. Dougenik, N. R. Chrisman, and D. R. Niemeyer. An Algorithm to Construct Continuous Area Cartograms. *Professional Geographer*, 37(1):75–81, 1985.

[3] S. M. Gusein-Zade and V. S. Tikunov. A New Technique for Constructing Continuous Cartograms. *Geography and Geographic Information Systems*, 20(3):167–173, 1993.

[4] D. H. House and C. J. Kocmoud. Continuous Cartogram Construction. In *Proc. of IEEE Visualization Conference*. IEEE, 1998.

[5] P. Kanjamla, P. Revesz, and Y. Wang. MLPQ/GIS: A GIS Using Linear Constraint Databases. In *Proc. of the 9th COMAD International Conference on Management of Data*, pages 389–393, 1998.

[6] C. J. Kocmoud and D. H. House. Cartogram Animation of U.S. Population Cartograms from 1900 to 1996. In *http://www-viz.tamu.edu/faculty/house/cartograms/DecadeAnim.html*.

[7] W. R. Tobler. Pseudo-Cartograms. *The American Cartographer*, 13(1):43–50, 1986.

[8] B. White, I. Gregory, and H. S. H. Analyzing and Visualizing Long-term Change . In *http://www.geog.qmw.ac.uk/gbhgis/gisruk98*.