Minimum-Distortion Continuous Cartograms by Numerically Optimized Meshes

Robert C. Sargent

November 26, 2024

Abstract

We present an algorithm for creating contiguous cartograms using meshes. We use numerical optimization to minimize cartographic error and distortion by transforming the mesh vertices. The vertices can either be optimized in the plane or optimized on the unit sphere and subsequently projected to the plane. We also present a hybrid "best of both worlds" method, where the vertices are optimized on the sphere while anticipating the distortion caused by the final projection to the plane. We show a significant improvement in the preservation of region shapes compared to existing automated methods. Outside the realm of cartograms, we apply this hybrid technique to optimized map projections, creating the Liquid Earth projection.

Thanks to Dr. Lawrence Washington for mentorship, encouragement, and feedback. Thanks to Justin Kunimune for encouragement, feedback, and programming help.

1 Introduction

Cartograms are data visualizations where each data point is represented as the area of a region. There are many different types of cartograms and

algorithms for creating them, including manual methods. (For a comprehensive list of cartogram types and algorithms, see [11].) This paper presents an algorithm for generating *continuous* cartograms (also called *contiguous* or *deformation* cartograms), which continuously deform an existing map so that the region areas match the data points. We focus on world maps of countries with area representing population, but the techniques in this paper are applicable to cartograms of subnational divisions as well, and with areas representing any desired data.

Our algorithm works by projecting the globe onto a mesh of triangles. We project the globe onto what we call the *initial mesh*, then transform the map by moving the mesh vertices to those of the *transformed mesh* (fig. 1). We generate the transformed mesh so that the end result is a cartogram with minimal distortion.

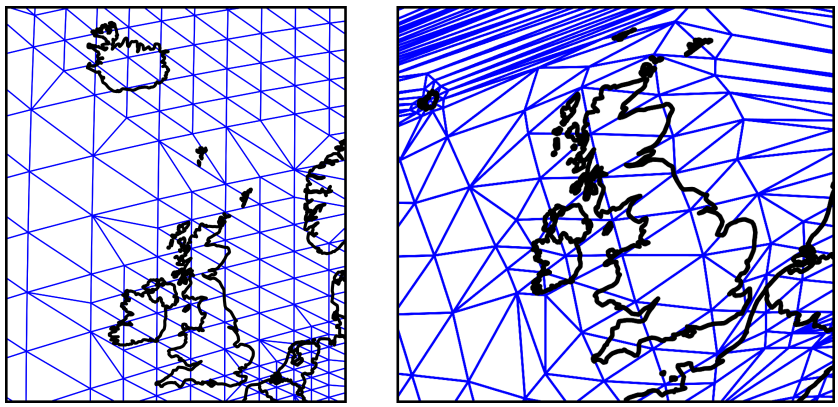

Figure 1: Part of the initial mesh and transformed mesh. Notice how the movement of the mesh vertices shrinks Ireland and Iceland.

To do this, we use numerical optimization, that is, finding a minimum of a multivariable function. We define a cost function whose inputs are the vertices of the transformed mesh. The cost function quantifies the total cartographic error (differences between the areas of regions and their desired values) and the total distortion. A local minimum of this function is an accurate cartogram with minimal distortion. The definition of this func-

tion depends on which version of the cartogram algorithm we use, as well as several hand-chosen parameters. For example, we choose to prioritize distortion of land much higher than distortion of water; we do this by giving the distortion of land triangles higher weight in the cost function.

Two key existing algorithms for automatic cartogram creation are the diffusion method of Gastner and Newman [3] and the rubber sheet method of Dougenik, Chrisman, and Niemeyer [2]. The diffusion method works by imagining the map is filled by a fluid whose density at each point corresponds to population density, then letting the fluid diffuse to equal density. The rubber sheet method applies expansion and contraction functions to the plane with effects centered at each region, repeating the process a small number of times until the cartogram is accurate.

Though these methods can produce accurate cartograms, the resulting maps have more distortion than necessary. For example, an island country can be given the desired area by scaling the region up or down, without distorting its shape at all. However, existing algorithms cannot avoid introducing distortion in this situation. Because the method we present is specifically based on minimizing distortion, it is capable of avoiding distortion in this and other situations (fig. 2).

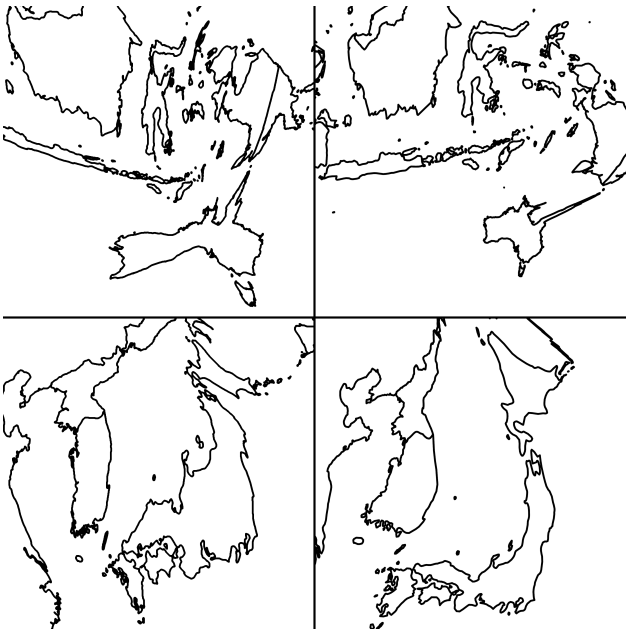

Figure 2: Comparison of shape preservation between the rubber-sheet method (left) and our mesh method (right). The mesh method preserves shapes as much as possible while preserving the connections (or near-connections) to adjacent landmasses. Rubber-sheet examples created with the program F4Carto by Sun [15].

Another issue with existing methods is that, with the exception of [8], they act only on the plane, not the sphere. This means that the globe must be projected to the plane before running the cartogram algorithm. This projection step bakes in distortion that the cartogram algorithm cannot correct. By contrast, our algorithm uses a spherical representation of the globe to calculate distortion, avoiding this problem. Li and Aryana [8] present a diffusion algorithm that acts on the sphere, producing a cartogram on a globe. However, a final projection step is needed to produce a planar cartogram, which introduces distortion.

Though our algorithm significantly improves on these points, it has a clear disadvantage in runtime. Though the aforementioned algorithms take only seconds or minutes to run [4], ours can take multiple hours. This is a hit to the practicality of our method. Optimizing the algorithm further is a

potential area of future research.

The use of numerically optimized meshes for cartography has been explored by Kunimune ([6], [7]) and by Loncaric [9] in the context of map projections. Kronenfeld [5] develops a method for *manually constructing* mesh-based cartograms. His method differs from ours in that, while we fix an initial mesh and optimize the transformed mesh, he fixes a transformed mesh and edits the initial mesh. This approach is helpful for dealing with small, population-dense regions ([5] pp. 81–82). However, if we varied the initial mesh as he does, our cost function would not be differentiable, which would make numerical optimization infeasible. Instead, we deal with small, dense regions by adding more detail to the initial mesh before generating the cartogram (fig. 3).

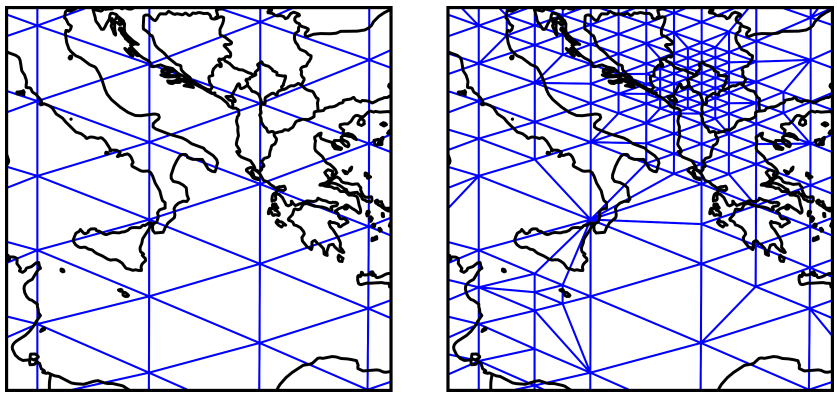

Figure 3: The initial mesh before and after subdivision.

We present three different versions of our cartogram algorithm. The first is the *plane cartogram* (fig. 4), in which the mesh vertices are projected to the plane before the cost function is minimized. Because the mesh is cut before the optimization happens, the boundary of the final map is irregular, and there may be self-intersections at the boundary. This method is the most similar to [6].

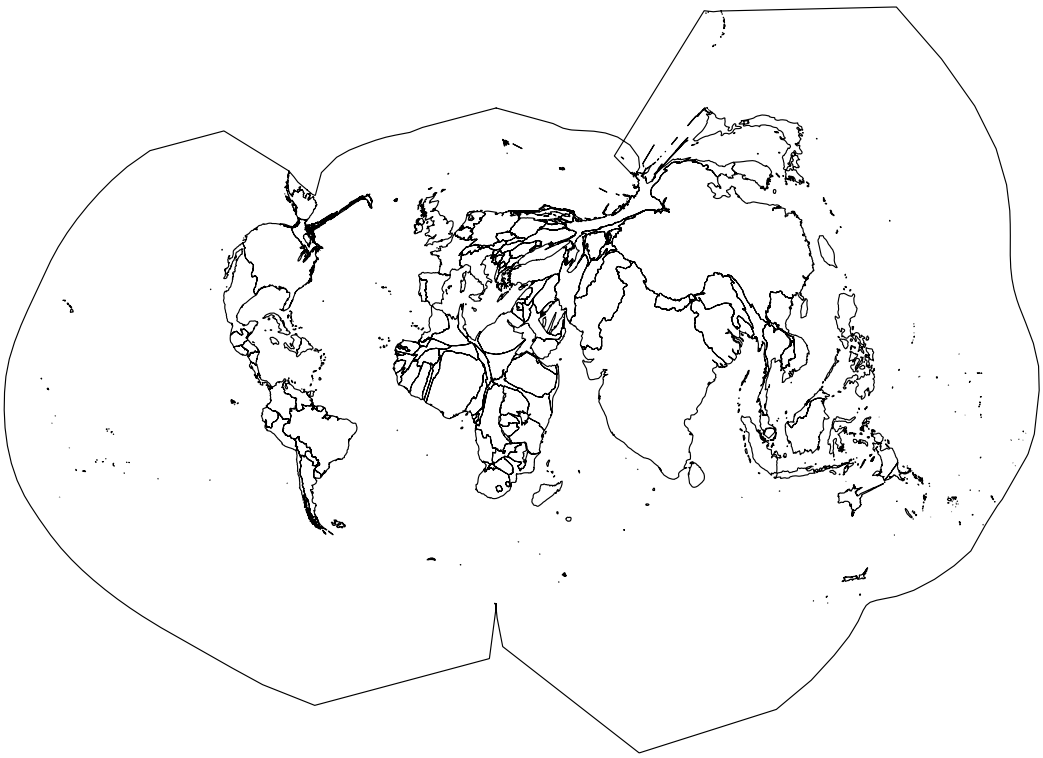

Figure 4: A plane cartogram.

The second version is the *sphere cartogram* (fig. 5). For this cartogram, the cost function takes in the positions of vertices on the unit sphere. Minimizing the cost function yields an accurate cartogram on the sphere. Since viewing a cartogram on a sphere is impractical, we then project the sphere to the plane using an equal-area map projection. The final projection step causes shape distortion. Since we optimize the cartogram on the sphere, topological problems are impossible. The boundary of the final map is regular; its shape is a consequence of the projection chosen.

Figure 5: A sphere cartogram shown in orthographic view and projected using the Mollweide projection.

Finally, the third version is the *hybrid cartogram* (fig. 6). Similarly to the sphere cartogram, it minimizes the cost function on the sphere, then projects the result to the plane. However, instead of calculating the distortion on the sphere, it anticipates how the mesh triangles will be transformed by the final projection and calculates distortion based on that. Minimizing this distortion function creates a cartogram that has minimal distortion *after projection to the plane*. As a result, the hybrid cartogram has the low distortion of the plane cartogram, while keeping the topological soundness and clean boundary

of the sphere cartogram. We call the final projection from the sphere to the plane the *target projection*, because the optimization is done with this projection in mind.

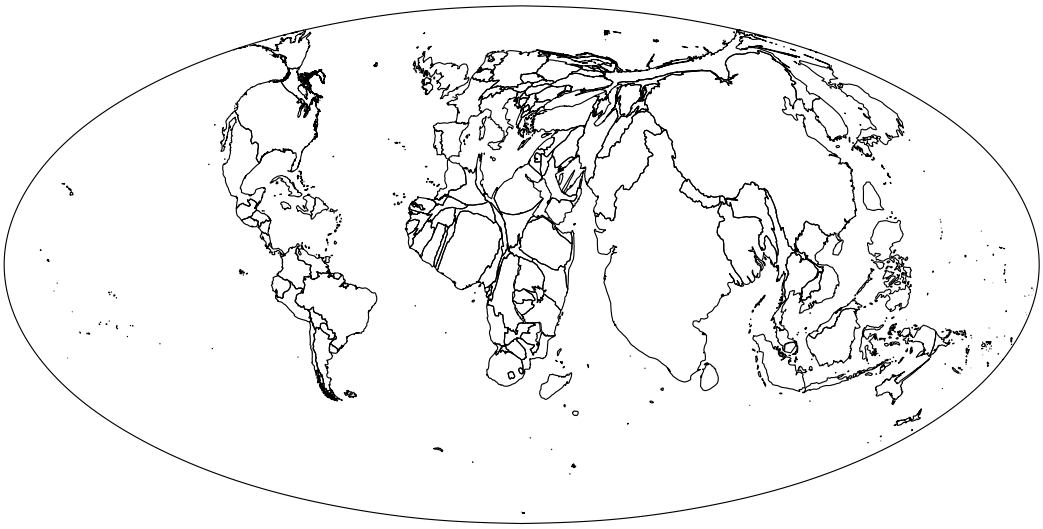

Figure 6: A hybrid cartogram whose target projection is Mollweide.

The technique used to create hybrid cartograms is also useful for creating new map projections. Kunimune ([6], [7]) prioritizes land distortion over water distortion to create some of the Danseiji and Elastic projections, as we do in our cartograms. However, he does his optimization on the plane. As with the hybrid cartogram, we can optimize a map projection on the sphere while anticipating the distortion caused by the final projection to the plane. This enables us to create a map projection that has exceptionally low distortion on land and whose boundary has a regular, familiar shape. Figure 7 is a particularly good result of this process; I have released it independently as the Liquid Earth projection [12].

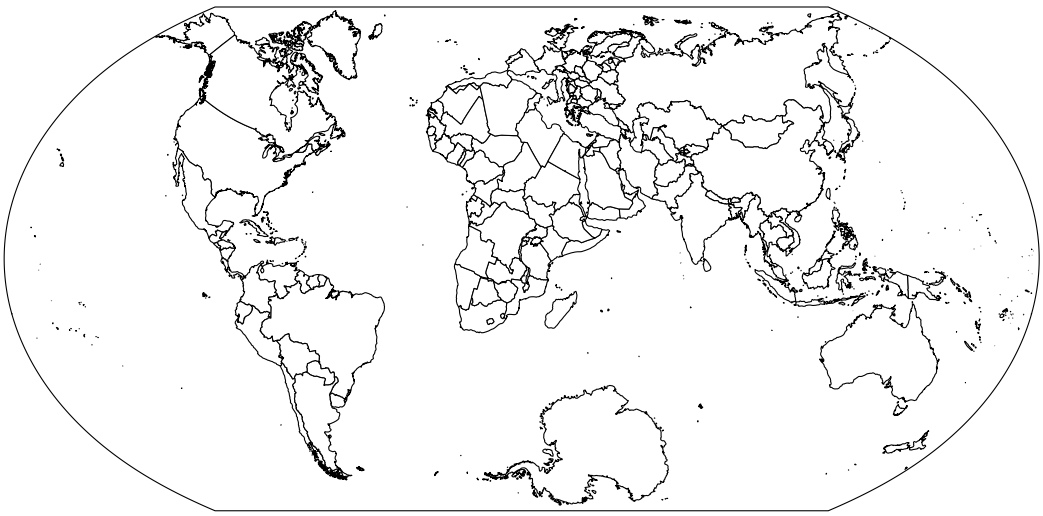

Figure 7: The Liquid Earth projection, an approximately equal-area map projection created by the hybrid technique. The target projection is Equal Earth.

Section 2 gives a high-level overview of the algorithm we present. Section 3 gives a complete definition of the cost function used to create the plane cartogram, and shows how to compute its gradient. The cost functions for the sphere and hybrid cartograms are modifications of the one for the plane cartogram; we describe these modifications in sections 4 and 5. Section 6 quantifies the accuracy and runtime of the algorithm. We give final remarks in section 7.

The source code for our algorithm is available at https://github.com/rsargentmath/cartogram-gradient-descent.

2 Algorithm overview

Throughout this paper, we use a right-handed coordinate system. Anticlockwise is the positive direction for both angles and polygons, including mesh triangles. The globe is always represented as the unit sphere, with the North Pole at $(0, 0, 1)$.

A cartogram is based on a set of *regions* $\mathcal{R}$, in practice either countries

or subnational divisions. Each region is a nonempty set of polygons on the unit sphere $S^2 \subseteq \mathbb{R}^3$. (None of these polygons overlap, whether they're in the same region or different regions.) The boundaries of these polygons are called *borders*. Each region $R \in \mathcal{R}$ has a number $r_R > 0$ associated with it, e.g., that country's population. These are the data the cartogram is trying to visualize. To be usable in the cartogram algorithm, we need to translate these values into areas we want the transformed regions to have. We define the desired area of each region by scaling the values r_R so that the total of the desired areas equals the total of the initial areas:

$$p_R = r_R \frac{\sum_{S \in \mathcal{R}} \mathring{\mu}_S}{\sum_{S \in \mathcal{R}} r_S},$$

where $\mathring{\mu}_S$ is the initial area of S. In a slight abuse of terminology, we call the desired area p_R the *population* of R.

A *mesh* consists of an array of vertices $\mathcal{V}$ and an array of triangles $\mathcal{T}$. The triangles in $\mathcal{T}$ do not contain vertices, rather, they are triples of indices into $\mathcal{V}$. This enables each vertex to be shared between multiple triangles. The choice to use triangles for our meshes differs from Kunimune ([6], [7]), who uses quads. We use triangles because the affine function that maps an initial triangle onto a transformed triangle scales all regions within the triangle evenly. This means only the portion that each region takes up of each triangle is needed to calculate the areas of transformed regions, as described below.

To create the specific mesh used in this paper, we start with a regular octahedron, then divide each face into 1024 triangles of roughly equal size, resulting in a mesh of 8192 triangles. After that, we subdivide the mesh further based on the borders and populations. We subdivide so that each region is covered by at least four triangles, and so triangles are expanded to at most 1/2048 of the sphere's area. We also subdivide once at international borders and near the North Pole. The final mesh has 15396 triangles. The

choice of an octahedron makes it easy to interrupt the map along a meridian, where two opposing vertices of the octahedron correspond to the poles.

The algorithm starts with a mesh $\left(\mathring{\mathcal{V}}, \mathcal{T}\right)$ on the sphere, representing the undistorted initial state of the globe. We project the borders onto this mesh radially. (Though this projection slightly distorts area, no cartographic error can be introduced here, because both the area calculation and the final map are based on these projected borders.) The algorithm outputs a new array of vertices $\mathcal{V}$, forming a mesh $(\mathcal{V}, \mathcal{T})$ where each triangle from the original mesh now references the new vertices. The final cartogram is created by projecting the borders from the original mesh to the new one. For each triangle, we use the unique affine transformation from the old triangle to the new one.

We want this process to result in an accurate cartogram that has minimal distortion. To achieve this, we use numerical optimization applied to a cost function $\mathfrak{C}$ whose input is an array of vertices $\mathcal{V}$. This function incorporates the *cartographic errors* (differences between the transformed areas of regions and their desired areas) as well as the total distortion of the map, which can be defined in various ways. The final vertices $\mathcal{V}$ are a local minimum of the function $\mathfrak{C}$. By adjusting the definition of $\mathfrak{C}$, we have fine control of the properties of the final cartogram. For example, we can weight scale distortion more or less heavily, and weight distortion of land more heavily than distortion of water.

To minimize $\mathfrak{C}(\mathcal{V})$, we start with an initial state $\mathcal{V}_0$, then step iteratively in the direction of lower cost until a local minimum is reached. For the sphere and hybrid cartograms, we simply set $\mathcal{V}_0 = \mathring{\mathcal{V}}$. For the plane cartogram, this initial state needs to be in the plane. To do this, we modify the initial mesh $\left(\mathring{\mathcal{V}}, \mathcal{T}\right)$ so that the vertices on the antimeridian (excluding poles) are duplicated, then create $\mathcal{V}_0$ by projecting these vertices to the plane, using a projection that interrupts the antimeridian. Each pair of duplicated vertices in $\mathring{\mathcal{V}}$ is separated by the interruption; these vertices form the outer edge of $\mathcal{V}_0$.

Suppose n is the number of vertices in $\mathcal{V}$ and d is the dimension of each vertex (2 or 3). For the purpose of minimizing $\mathfrak{C}(\mathcal{V})$, we view $\mathcal{V}$ as a single vector in $\mathbb{R}^{nd}$ instead of a tuple of vectors in $\mathbb{R}^d$. With this view, the gradient vector $\nabla\mathfrak{C}(\mathcal{V})$ is also an element of $\mathbb{R}^{nd}$. The minimization algorithm we choose is L-BFGS [10] with backtracking line search [1], as was used by Kunimune [6]. This is a standard algorithm that works by computing and storing the gradient at each step, and using the gradients at the last few steps to choose the next step. The line search tries this step and computes its cost. If the cost does not decrease enough, or increases, it rejects the step and tries a smaller step. In particular, the line search always rejects a step if it leads to infinite cost.

In order to avoid states with infinite cost, $\mathfrak{C}$ must be continuous in the following sense. For any input $\mathcal{V}^*$ to $\mathfrak{C}$, we must have $\lim_{\mathcal{V}\to\mathcal{V}^*}\mathfrak{C}(\mathcal{V}) = \mathfrak{C}(\mathcal{V}^*)$, including in the case where $\mathfrak{C}(\mathcal{V}^*) = \infty$. If this fails and there are points where $\mathfrak{C}(\mathcal{V})$ discontinuously jumps to ∞, then the optimization algorithm can get stuck, because it has no way of steering around the disallowed states, despite knowing not to accept them.

Let $\mathfrak{E}(\mathcal{V})$ be the total cartographic error and let $\mathfrak{D}(\mathcal{V})$ be the total distortion. The definitions of $\mathfrak{E}$ and $\mathfrak{D}$ depend on both the type of the cartogram (plane, sphere, or hybrid) and various chosen weights, as described in this and following sections. We define maps with topological problems, such as having a triangle flipped over, to have infinite distortion; this ensures that the final output will not have these problems. The goal is to have $\mathfrak{E}(\mathcal{V})$ as close to zero as possible (so that the cartogram is accurate) while minimizing $\mathfrak{D}(\mathcal{V})$ subject to this constraint. Achieving this is complicated by the fact that the initial state of the map does not satisfy $\mathfrak{E}(\mathcal{V}) = 0$.

Our algorithm works by setting

$$\mathfrak{C}(\mathcal{V}) = W^{\text{error}}\,\mathfrak{E}(\mathcal{V}) + W^{\text{dist}}\,\mathfrak{D}(\mathcal{V}),$$

where $W^{\text{error}}, W^{\text{dist}} \in \mathbb{R}$ are positive weights. We minimize $\mathfrak{C}(\mathcal{V})$ multiple

times, each time prioritizing cartographic error higher by lowering W^{dist}. The output of one step becomes the starting point of the next step's minimization. This way, the total cartographic error gets closer and closer to zero while distortion is still minimized. This process is reminiscent of the interior-point method from numerical optimization. (We cannot set W^{dist} to zero because $\mathfrak{D}$ is responsible for maintaining the topology of the map.)

We define $\mathfrak{E}$ as follows. Let μ_R be the transformed area of a region R, which depends on the transformed vertices $\mathcal{V}$. (The method for computing μ_R depends on the type of the cartogram.) The cartographic error of R is $\varepsilon_R = \mu_R - p_R$. We define

$$\mathfrak{E}(\mathcal{V}) = \sum_{R \in \mathcal{R}} w_R^{\text{error}} \varepsilon_R^2,$$

where $(w_R^{\text{error}})_{R \in \mathcal{R}}$ is a family of positive weights. The function $\mathfrak{E}$ attains a minimum of 0 precisely when $\mu_R = p_R$ for all regions R, as desired. These weights allow us to choose between minimizing absolute error, given by $\mu_R - p_R$, and relative error, given by $(\mu_R - p_R)/p_R$. If $w_R^{\text{error}} = 1/p_R^2$, then $\mathfrak{E}(\mathcal{V})$ is the summed square relative error. For the cartograms in this paper, we split the difference and choose $w_R^{\text{error}} = 1/p_R$, since the extreme values given by $w_R^{\text{error}} = 1/p_R^2$ cause floating-point issues.

The total distortion $\mathfrak{D}(\mathcal{V})$ is computed by summing the scale distortion and shape distortion over all triangles $T \in \mathcal{T}$. As with cartographic error, we include arbitrarily chosen weights. This function may also include terms unrelated to these two types of distortion. For example, the plane cartogram described in section 3 includes terms that prevent the map from having self-intersections at the poles. We give definitions for the scale and shape distortions at a triangle based on cartogram type (plane, sphere, or hybrid) in the following sections.

3 Cost function for plane cartograms

In this section, we fully define the cost function for the plane cartogram and show how to compute its gradient. The cost and gradient calculations for the sphere and hybrid cartograms are based on those for the plane, with the necessary modifications.

For each triangle T and region R, we let $\psi_{R,T}$ be the portion R takes up of T, that is,

$$\psi_{R,T} = \frac{\text{area}(R \cap T)}{\text{area}(T)}.$$

Since the maps from the initial mesh triangles to the transformed triangles are affine, the values $\psi_{R,T}$ do not depend on $\mathcal{V}$. These constants are all we need to compute the transformed area of each region; there is no complicated dependence on the geometry of the regions. We also compute the portion *land* takes up of each triangle T, given by $\overline{\psi}_T = \sum_{R \in \mathcal{R}} \psi_{R,T}$.

For each triangle T, let $\mathring{\mathbf{a}}_T, \mathring{\mathbf{b}}_T, \mathring{\mathbf{c}}_T \in \mathring{\mathcal{V}}$ be the untransformed positions of the three vertices referenced by T, and let $\mathbf{a}_T, \mathbf{b}_T, \mathbf{c}_T \in \mathcal{V}$ be their transformed positions. For simplicity, we suppress T and write $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}}, \mathbf{a}, \mathbf{b}, \mathbf{c}$. To define the distortion at T, we need to define the distortion of the affine transformation that takes $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}}$ to $\mathbf{a}, \mathbf{b}, \mathbf{c}$ respectively. Note that $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}} \in \mathbb{R}^3$ while $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^2$. Let $\mathring{\mathcal{B}}$ be an orthonormal basis for the plane parallel to the triangle $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}}$. Then we can analyze this transformation by looking at the 2×2 matrix K taking $(\mathring{\mathbf{b}} - \mathring{\mathbf{a}})_{\mathring{\mathcal{B}}}$, $(\mathring{\mathbf{c}} - \mathring{\mathbf{a}})_{\mathring{\mathcal{B}}}$ to $\mathbf{b} - \mathbf{a}$, $\mathbf{c} - \mathbf{a}$ respectively. This matrix is given by

$$K = G\mathring{G}^{-1}, \tag{1}$$

where

$$\mathring{G} = \begin{bmatrix} | & | \\ (\mathring{\mathbf{b}} - \mathring{\mathbf{a}})_{\mathring{\mathcal{B}}} & (\mathring{\mathbf{c}} - \mathring{\mathbf{a}})_{\mathring{\mathcal{B}}} \\ | & | \end{bmatrix}, \quad G = \begin{bmatrix} | & | \\ \mathbf{b} - \mathbf{a} & \mathbf{c} - \mathbf{a} \\ | & | \end{bmatrix}. \tag{2}$$

Since the distortion of a transformation should not be affected by rotation,

the choice of $\mathring{\mathcal{B}}$ does not matter as long as it is correctly oriented (in the sense that it gives $\mathring{G}$ a positive determinant). These matrices also allow us to compute the initial area $\mathring{m}$ and transformed area m of T:

$$\mathring{m} = \frac{1}{2} \det \mathring{G}, \quad m = \frac{1}{2} \det G = \mathring{m} \det K. \tag{3}$$

There are many ways to define the distortion of a 2×2 matrix. The definition chosen by Kunimune [6] has two problems for use in cartograms: (a) one term represents both scale distortion and shape distortion, when we need these to be separate; (b) the scale distortion term, which is the squared log of the scale factor, does not adequately punish the extreme scale distortions that can arise in cartograms. We give a definition that fixes these problems and makes the gradient simple to compute.

Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ be any real 2×2 matrix. We define the shape distortion of A to be

$$\begin{cases} \dfrac{a^2 + b^2 + c^2 + d^2}{ad - bc} - 2 & ad - bc > 0 \\ \infty & \text{else.} \end{cases}$$

(In practice, such inequality checks are always implemented with a tolerance to compensate for floating-point error, e.g., checking $ad - bc > 10^{-12}$ instead of $ad - bc > 0$. For simplicity, we ignore this throughout this paper.) To justify this definition, notice that this fraction is the square of the Frobenius norm of A divided by the determinant of A. Both of these values are invariant when multiplying A by a rotation matrix on the left or right side. Thus, if $R_2 \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} R_1$ is a singular value decomposition of A (R_1, R_2 rotation matrices), then the shape distortion of A equals the shape distortion of

$\begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$. If $\det A = \sigma_1\sigma_2 > 0$, then this equals

$$\frac{\sigma_1^2 + \sigma_2^2}{\sigma_1\sigma_2} - 2 = \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1} - 2.$$

This value is minimized at zero precisely when $\sigma_1 = \sigma_2$, i.e., when A is a multiple of a rotation matrix. The constant term has no effect on the algorithm's outcome; its only purpose is cosmetic, ensuring that the shape distortion of a conformal transformation is measured as 0 instead of 2. We denote the shape distortion of our matrix K by δ^{shape}.

Note that, while we used the singular values to explain this definition, they are not necessary for computing this value. The distortion is given simply in terms of the matrix entries, which makes cost and gradient calculation easier. ([9] defined the same distortion function, but gave it only in terms of the singular values.) An important characteristic is that, when fixing σ_1, this function is convex in σ_2, and vice versa. This property is important to prevent "necking" artifacts in the final map, where some triangles are near-perfect while others are extremely stretched [6].

We define the scale distortion at this triangle in a similar way. We choose a value s to be the *intended scale* for this triangle. Then we let the scale distortion be

$$\delta^{\text{scale}} = \begin{cases} \dfrac{\det K}{s} + \dfrac{s}{\det K} - 2 & \det K > 0 \\ \infty & \text{else.} \end{cases}$$

This is minimized at zero precisely when $\det K = s$. Note that the inequality check is the same for both δ^{shape} and δ^{scale}.

If a triangle T is taken up entirely by one region R, then it's clear that the intended scale of T should equal the desired scaling of R, i.e., $s_T = p_R/\mathring{\mu}_R$,

where $\mathring{\mu}_R$ is the initial area of R. For any region R, we compute $\mathring{\mu}_R$ by

$$\mathring{\mu}_R = \sum_{T \in \mathcal{T}} \psi_{R,T}\, \mathring{m}_T. \tag{4}$$

In the general case where T intersects at least one region, possibly including water as well, we choose an average of these values, weighted by how much of each region appears in T:

$$s_T = \sum_{R \in \mathcal{R}} \frac{\psi_{R,T}}{\overline{\psi}_T} \cdot \frac{p_R}{\mathring{\mu}_R}.$$

The remaining case is when T contains entirely water, i.e., when $\overline{\psi}_T = 0$. In this case, it is not immediately clear how to choose s_T. For the cartograms in this paper, we blur the intended scale values across the mesh so that triangles with $\overline{\psi}_T = 0$ take on the values of nearby triangles, leaving triangles with $\overline{\psi}_T > 0$ unaffected. This improves the regularity of the mesh near coastlines.

Using (3), we can compute the transformed area μ_R of a region R, similar to (4):

$$\mu_R = \sum_{T \in \mathcal{T}} \psi_{R,T}\, m_T = \sum_{T \in \mathcal{T}} \psi_{R,T}\, \mathring{m}_T \det K_T.$$

Thus, the total cartographic error is

$$\mathfrak{E}(\mathcal{V}) = \sum_{R \in \mathcal{R}} w_R^{\text{error}} \varepsilon_R^2,$$

where

$$\varepsilon_R = \mu_R - p_R = \Big[\sum_{T \in \mathcal{T}} \psi_{R,T}\, \mathring{m}_T \det K_T \Big] - p_R.$$

We define the total distortion $\mathfrak{D}(\mathcal{V})$ by

$$\mathfrak{D}(\mathcal{V}) = \sum_{T \in \mathcal{T}} \mathring{m}_T \big(w_T^{\text{shape}} \delta_T^{\text{shape}} + w_T^{\text{scale}} \delta_T^{\text{scale}} \big), \tag{5}$$

where $(w_T^{\text{shape}})_{T\in\mathcal{T}}$ and $(w_T^{\text{scale}})_{T\in\mathcal{T}}$ are families of positive weights. We multiply by $\mathring{m}_T$ because distortion should be punished more if it affects a larger area.

The choice of weights for distortion has a large effect on the look of the final cartogram. For our cartograms, we incorporate both the presence or absence of land and the intended scale into these weights. To do this, we define auxiliary weight values that we multiply to obtain the final weights. We weight triangles that are entirely water much less than triangles that contain land, defining

$$\alpha_T^{\text{land}} = \begin{cases} 1 & \overline{\psi}_T > 0 \\ 0.1 & \text{else.} \end{cases}$$

Triangles that are mostly water but contain some of a region border should still receive full weight, because they are key to maintaining the visual shapes of regions. A small but significant weight on entirely-water triangles is still necessary to maintain the relative positions of regions and to avoid topological problems.

In (5), we multiply each triangle T's distortion by $\mathring{m}_T$, its initial area. This means that the distortion of a region is judged using that region's size on the globe, not its size on the final map. In practice, this causes low–population density regions to be privileged over dense ones, e.g., leaving Canada mostly undistorted while the US warps to accommodate it. It would make more sense to judge distortion based on a triangle's final area. However, we can't just replace $\mathring{m}_T$ with m_T in (5), because that would incentivize high-distortion triangles to shrink! To get around this, we include a weight factor based on each triangle's intended scale s_T instead. So that very low-density areas are not completely ignored, we set

$$\alpha_T^{\text{density}} = 0.2 + 0.8 s_T.$$

Lastly, we choose how we weight shape distortion and scale distortion

overall. For our cartograms, we choose

$$\alpha^{\text{shape}} = 0.5, \quad \alpha^{\text{scale}} = 0.2.$$

For unclear reasons, higher values of α^{scale} cause much longer runtimes, as the optimization takes many more steps to find a local minimum. Lower values improve runtime, but cause aesthetic problems. The given values are a compromise.

We define the final weights by

$$w_T^{\text{shape}} = \alpha^{\text{shape}}\, \alpha_T^{\text{land}}\, \alpha_T^{\text{density}}, \quad w_T^{\text{scale}} = \alpha^{\text{scale}}\, \alpha_T^{\text{land}}\, \alpha_T^{\text{density}}.$$

With this, we have given a complete definition of the error and distortion functions for the plane cartogram. However, using this definition unmodified for the plane cartogram fails to prevent self-intersections at the poles. This is not an issue for the sphere and hybrid cartograms, so to keep the discussion generally applicable, we momentarily ignore this problem and proceed with the gradient calculation.

Recall that $\mathfrak{C}(\mathcal{V}) = W^{\text{error}}\, \mathfrak{E}(\mathcal{V}) + W^{\text{dist}}\, \mathfrak{D}(\mathcal{V})$, so

$$\nabla\mathfrak{C}(\mathcal{V}) = W^{\text{error}}\, \nabla\mathfrak{E}(\mathcal{V}) + W^{\text{dist}}\, \nabla\mathfrak{D}(\mathcal{V}),$$

as long as $\mathfrak{C}(\mathcal{V}) < \infty$. Expanding this,

$$\nabla\mathfrak{E}(\mathcal{V}) = \nabla\Big[\sum_{R\in\mathcal{R}} w_R^{\text{error}} \varepsilon_R^2\Big] = 2\sum_{R\in\mathcal{R}} w_R^{\text{error}} \varepsilon_R\, \nabla\varepsilon_R,$$

$$\nabla\varepsilon_R = \nabla\mu_R = \nabla\Big[\sum_{T\in\mathcal{T}} \psi_{R,T}\, \mathring{m}_T\, \det K_T\Big] = \sum_{T\in\mathcal{T}} \psi_{R,T}\, \mathring{m}_T\, \nabla(\det K_T).$$

Rearranging,

$$\begin{aligned}\nabla\mathfrak{E}(\mathcal{V}) &= 2\sum_{R\in\mathcal{R}} w_R^{\text{error}}\varepsilon_R\Big[\sum_{T\in\mathcal{T}}\psi_{R,T}\,\mathring{m}_T\,\nabla(\det K_T)\Big]\\ &= 2\sum_{T\in\mathcal{T}}\Big[\sum_{R\in\mathcal{R}} w_R^{\text{error}}\varepsilon_R\,\psi_{R,T}\Big]\mathring{m}_T\,\nabla(\det K_T).\end{aligned}$$

Furthermore,

$$\begin{aligned}\nabla\mathfrak{D}(\mathcal{V}) &= \nabla\Big[\sum_{T\in\mathcal{T}}\mathring{m}_T\big(w_T^{\text{shape}}\delta_T^{\text{shape}} + w_T^{\text{scale}}\delta_T^{\text{scale}}\big)\Big]\\ &= \sum_{T\in\mathcal{T}}\mathring{m}_T\big(w_T^{\text{shape}}\,\nabla\delta_T^{\text{shape}} + w_T^{\text{scale}}\,\nabla\delta_T^{\text{scale}}\big),\end{aligned}$$

so

$$\begin{aligned}\nabla\mathfrak{C}(\mathcal{V}) &= 2W^{\text{error}}\sum_{T\in\mathcal{T}}\Big[\sum_{R\in\mathcal{R}} w_R^{\text{error}}\varepsilon_R\,\psi_{R,T}\Big]\mathring{m}_T\,\nabla(\det K_T)\\ &\quad + W^{\text{dist}}\sum_{T\in\mathcal{T}}\mathring{m}_T\big(w_T^{\text{shape}}\,\nabla\delta_T^{\text{shape}} + w_T^{\text{scale}}\,\nabla\delta_T^{\text{scale}}\big)\\ &= \sum_{T\in\mathcal{T}}\mathring{m}_T\Bigg[2W^{\text{error}}\Big[\sum_{R\in\mathcal{R}} w_R^{\text{error}}\varepsilon_R\,\psi_{R,T}\Big]\nabla(\det K_T)\\ &\qquad + W^{\text{dist}}\big(w_T^{\text{shape}}\,\nabla\delta_T^{\text{shape}} + w_T^{\text{scale}}\,\nabla\delta_T^{\text{scale}}\big)\Bigg].\end{aligned}\tag{6}$$

Since δ_T^{shape}, δ_T^{scale}, and $(\det K_T)$ are defined in terms of the entries of K_T, their derivatives (with respect to each component of $\mathcal{V}$) are simple to compute from the entries of K_T and its derivatives. Finally, when differentiating with respect to each component of $\mathcal{V}$,

$$K_T' = (G_T\,\mathring{G}_T^{-1})' = G_T'\,\mathring{G}_T^{-1},$$

since $\mathring{G}_T$ is constant. The matrix G_T' is immediate from (2); for example,

the derivative of G_T with respect to the x component of $\mathbf{a}_T$ is $\begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}$.

Each triangle is only affected by the three vertices it references. This means that each element of the outermost sum in (6) has at most six nonzero components, namely the derivatives with respect to the components of the three vertices of that triangle. When computing $\nabla\mathfrak{C}(\mathcal{V})$ in code, we compute these six derivatives for each triangle T, then place them at the vertex indices referenced by T.

This cost function definition and gradient computation also work for the sphere cartogram. As described in section 4, the only changes necessary are (a) the dimension of the vectors in $\mathcal{V}$; (b) the definition of the matrices K_T. We now describe a modification necessary to prevent the plane cartogram from self-intersecting; this modification does not apply to the sphere or hybrid cartograms.

The self-intersection issue is worst at the South Pole, caused by Antarctica contracting. The land around the pole is shrank nearly to a point, while the surrounding water is shrank less. This situation makes the mesh behave like a negatively-curved surface, causing the angle at the pole to expand far past 360 degrees (fig. 8). (Whether or not the cartogram actually represents country populations, Antarctica will likely have a data value near zero. This value cannot be exactly zero because that leads to division by zero. For our cartograms, we use an estimate of the temporary population of Antarctica, which has a small nonzero value.)

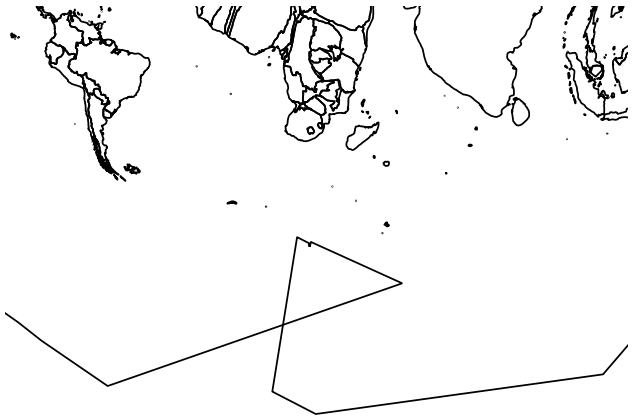

Figure 8: A large self-intersection caused by Antarctica contracting.

To prevent this problem, we add a term to the definition of $\mathfrak{D}(\mathcal{V})$ that compares the x coordinates of vertices on the boundary to those of the poles. Points in the lower-right quadrant of the boundary are be required to be to the right of the South Pole, while those in the lower-left quadrant must be to its left. We do the same for the North Pole.

We define the sets $Q_0, Q_1, Q_2, Q_3 \subseteq \mathcal{V}$ of boundary vertices as in fig. 9, along with the poles $\mathbf{p}^{\text{north}}, \mathbf{p}^{\text{south}} \in \mathcal{V}$. We define a cost value for the boundary by

$$\begin{aligned}\delta^{\text{boundary}} = &\sum_{\mathbf{v}\in Q_0} \frac{1}{\mathbf{v}_x - \mathbf{p}_x^{\text{north}}} + \sum_{\mathbf{v}\in Q_1} \frac{1}{\mathbf{p}_x^{\text{north}} - \mathbf{v}_x} \\ &+ \sum_{\mathbf{v}\in Q_2} \frac{1}{\mathbf{p}_x^{\text{south}} - \mathbf{v}_x} + \sum_{\mathbf{v}\in Q_3} \frac{1}{\mathbf{v}_x - \mathbf{p}_x^{\text{south}}},\end{aligned}$$

with $\delta^{\text{boundary}} := \infty$ if any of these denominators is not positive. We then modify the distortion function (5) to be

$$\mathfrak{D}(\mathcal{V}) = \Big[\sum_{T\in\mathcal{T}} \mathring{m}_T\big(w_T^{\text{shape}}\delta_T^{\text{shape}} + w_T^{\text{scale}}\delta_T^{\text{scale}}\big)\Big] + w^{\text{boundary}}\delta^{\text{boundary}}.$$

We set w^{boundary} to be quite small, specifically 10^{-6}, so that it only has an effect when a boundary point is very close to crossing over. The corresponding

modification to $\nabla\mathfrak{D}(\mathcal{V})$ is straightforward.

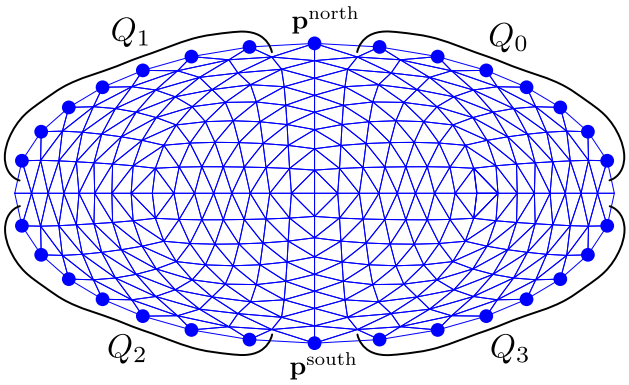

Figure 9: The sets of boundary vertices Q_0, Q_1, Q_2, Q_3 and the vertices $\mathbf{p}^{\text{north}}$ and $\mathbf{p}^{\text{south}}$. Mesh resolution reduced for clarity.

This addition does not prevent all self-intersections in the final map, but it avoids a common and severe case. The sphere and hybrid cartograms do guarantee a lack of topological problems by construction.

4 Sphere cartograms

While the transformed vertices in the plane cartogram are in $\mathbb{R}^2$, the transformed vertices in the sphere cartogram are in $\mathrm{S}^2 \subseteq \mathbb{R}^3$. The borders in each triangle are mapped using the affine function taking the initial vertices $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}}$ to the transformed vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$, then projected radially onto S^2.

Care must be taken when adapting the cost function for the sphere. With this radial projection to the sphere, the maps from the initial triangles to the transformed triangles of the cartogram are no longer affine. In order to measure distortion using the methods in section 3, we need to choose affine approximations of these maps. The obvious choice is to ignore the projection to the sphere and just look at the map from $\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}}$ to $\mathbf{a}, \mathbf{b}, \mathbf{c}$. However, this results in the cost function being discontinuous. Points that are collinear on the sphere are in general not collinear in $\mathbb{R}^3$, so if a triangle is in the process

of flipping over, it maintains significant positive area right up until it flips. This causes the shape and scale distortion values to discontinuously jump to infinity. The option we choose instead is to project the transformed triangle onto the tangent plane at its midpoint, then calculate distortion based on this projected triangle.

Define the function nzd ("normalized") by $\operatorname{nzd}(\mathbf{v}) = \dfrac{\mathbf{v}}{\|\mathbf{v}\|}$. Let $\mathbf{a}, \mathbf{b}, \mathbf{c}$ be the transformed vertices of some fixed triangle. We define the midpoint of this triangle to be

$$\mathbf{n} = \operatorname{nzd}\left(\frac{\mathbf{a}+\mathbf{b}+\mathbf{c}}{3}\right) = \operatorname{nzd}(\mathbf{a}+\mathbf{b}+\mathbf{c}). \tag{7}$$

We project $\mathbf{a}, \mathbf{b}, \mathbf{c}$ perpendicularly onto the tangent plane to the sphere at $\mathbf{n}$, yielding the points $\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$. The point $\mathbf{a}^{\mathrm{tp}}$ is given by

$$\mathbf{a}^{\mathrm{tp}} = \mathbf{a} + (1 - \mathbf{a}\cdot\mathbf{n})\,\mathbf{n},$$

with $\mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$ given similarly. Finally, we modify the definition of the matrix K [(1) and (2)] by setting

$$G = \begin{bmatrix} | & | \\ (\mathbf{b}^{\mathrm{tp}} - \mathbf{a}^{\mathrm{tp}})_{\mathcal{B}} & (\mathbf{c}^{\mathrm{tp}} - \mathbf{a}^{\mathrm{tp}})_{\mathcal{B}} \\ | & | \end{bmatrix}, \tag{8}$$

where $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is a correctly oriented orthonormal basis for the plane parallel to this tangent plane. (Since $\mathring{G}$ is constant, there is no need to modify its definition in a similar way.)

The area of the planar triangle $(\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}})$ is not quite equal to the area of the spherical triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$, and the radial projection to the sphere is not area-equivalent, so a small amount of cartographic error is introduced. We estimate this error in section 6.

The basis $\mathcal{B}$ is dependent on the tangent plane at $\mathbf{n}$, which depends on $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Since $\mathcal{B}$ is not constant as the mesh vertices vary, we must give a

fixed definition for it and pay attention to how it moves. We choose $\mathcal{B}$ to align with the longitude–latitude graticule, meaning that $\mathcal{B}_1$ points east and $\mathcal{B}_2$ points north. This choice is convenient when defining hybrid cartograms (section 5). (If $\mathbf{n}$ is at a pole, we choose $\mathcal{B}$ arbitrarily.) With this choice made, it is possible to describe each component of G directly in terms of $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and use those formulas to calculate the gradient of the cost function. However, it is simpler to calculate the gradient with respect to $\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$, then use the chain rule to find the gradient in terms of $\mathbf{a}, \mathbf{b}, \mathbf{c}$, as described below.

Notice that, if $\mathbf{a}^{\mathrm{tp}}$, $\mathbf{b}^{\mathrm{tp}}$, or $\mathbf{c}^{\mathrm{tp}}$ were to vary in the direction perpendicular to the tangent plane, this would have no first-order effect on the distortion of the triangle $(\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}})$. This means that, when computing the gradient, we only need to consider the movement of these points within the tangent plane. As for the basis, notice that as the midpoint $\mathbf{n}$ varies in the east–west direction, $\mathcal{B}$ rotates so that the second basis vector points points towards the North Pole. This is a first-order effect, so it affects the derivatives of G. However, our area and distortion calculations are unaffected by rotation. Thus, when computing the gradient of the cost, we can view both the tangent plane and the basis as fixed. If the rotation of G does matter, as in the case of the hybrid cartogram, we correct for this after the fact, as described in section 5.

Since we view $\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$ as varying within a fixed plane, the computations for the derivatives of $\det K$, δ^{shape}, and δ^{scale}—with respect to these vertices moving along $\mathcal{B}_1$ and $\mathcal{B}_2$—are the same as the corresponding computations for the plane cartogram. The resulting gradient vectors are expressed in the basis $\mathcal{B}$; we then use $\mathcal{B}$ to express them in global coordinates. Finally, we use the chain rule to find the derivatives of $\det K$, δ^{shape}, and δ^{scale} with respect to the components of $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. This involves finding the derivatives of $\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$ with respect to each of these components. Let $\mathbf{d} = \mathbf{a} + \mathbf{b} + \mathbf{c}$.

To find each of these derivatives, we compute

$$\mathbf{d}' = \mathbf{a}' + \mathbf{b}' + \mathbf{c}',$$

$$\begin{aligned}\mathbf{n}' = \mathrm{nzd}(\mathbf{d})' &= \left(\frac{\mathbf{d}}{\|\mathbf{d}\|}\right)' = \left((\mathbf{d}\cdot\mathbf{d})^{-1/2}\,\mathbf{d}\right)' \\ &= -\frac{1}{2}(\mathbf{d}\cdot\mathbf{d})^{-3/2}\,2(\mathbf{d}\cdot\mathbf{d}')\,\mathbf{d} + (\mathbf{d}\cdot\mathbf{d})^{-1/2}\,\mathbf{d}' \\ &= -(\mathbf{d}\cdot\mathbf{d})^{-3/2}\,(\mathbf{d}\cdot\mathbf{d}')\,\mathbf{d} + (\mathbf{d}\cdot\mathbf{d})^{-1/2}\,\mathbf{d}'.\end{aligned}$$

Then

$$\begin{aligned}(\mathbf{a}^{\mathrm{tp}})' &= \big(\mathbf{a} + (1 - \mathbf{a}\cdot\mathbf{n})\,\mathbf{n}\big)' \\ &= \mathbf{a}' - (\mathbf{a}'\cdot\mathbf{n} + \mathbf{a}\cdot\mathbf{n}')\,\mathbf{n} + (1 - \mathbf{a}\cdot\mathbf{n})\,\mathbf{n}',\end{aligned}$$

and similarly for $(\mathbf{b}^{\mathrm{tp}})'$ and $(\mathbf{c}^{\mathrm{tp}})'$. Notice that each of $\mathbf{a}^{\mathrm{tp}}, \mathbf{b}^{\mathrm{tp}}, \mathbf{c}^{\mathrm{tp}}$ depends on all of $\mathbf{a}, \mathbf{b}, \mathbf{c}$.

It remains to discuss how to actually carry out this optimization when the mesh vertices' domain is the sphere. In our implementation, we represent the vertices with 3D Cartesian coordinates, as we do mathematically. After each step (in the search direction provided by L-BFGS), we normalize each vertex back onto the sphere. Explicitly, if $\mathcal{V} = (\mathbf{v}_0, \ldots, \mathbf{v}_{n-1})$ is the current vertex array ($\mathbf{v}_i \in \mathrm{S}^2 \subseteq \mathbb{R}^3$), and $\mathcal{S} = (\mathbf{s}_0, \ldots, \mathbf{s}_{n-1})$ is the step produced by L-BFGS and the line search, then the next positions of the mesh vertices, $\mathcal{V}^{\mathrm{next}} = (\mathbf{v}_0^{\mathrm{next}}, \ldots, \mathbf{v}_{n-1}^{\mathrm{next}})$, is given by $\mathbf{v}_i^{\mathrm{next}} = \mathrm{nzd}(\mathbf{v}_i + \mathbf{s}_i)$ for all i. This normalization step introduces a small challenge in the implementation.

The line search is a standard part of the optimization algorithm. The program tries the step suggested by L-BFGS and compares the resulting reduction in cost to the reduction that is expected based on the gradient. If the cost increases or the reduction is otherwise too small, we try again with a smaller step. This is repeated until an acceptable step is found. Viewing

$\mathcal{V}$, $\mathcal{V}^{\text{next}}$, $\nabla\mathfrak{C}(\mathcal{V})$, and $\mathcal{S}$ as vectors in $\mathbb{R}^{3n}$, we define $\mathcal{S}$ to be acceptable if it satisfies Armijo's condition [1]:

$$\mathfrak{C}(\mathcal{V}^{\text{next}}) - \mathfrak{C}(\mathcal{V}) \leq c\big(\mathcal{S} \cdot \nabla\mathfrak{C}(\mathcal{V})\big), \tag{9}$$

where $c \in (0, 1)$ is a constant (we choose 0.1).

If there is no normalization, then we are guaranteed to find a step satisfying (9), since $\mathcal{S} \cdot \nabla\mathfrak{C}(\mathcal{V})$ is a first-order approximation for $\mathfrak{C}(\mathcal{V} + \mathcal{S}) - \mathfrak{C}(\mathcal{V})$ as $\mathcal{S}$ varies, by the definition of the gradient. However, the normalization step causes a problem. The gradient $\nabla\mathfrak{C}(\mathcal{V})$ can direct points to move off the sphere, in which case the step $\mathcal{S}$ will likely move points off the sphere. When these points are normalized, the reduction in cost is smaller than is expected based on $\mathcal{S}$ (fig. 10). If $\mathcal{S}$ and $\nabla\mathfrak{C}(\mathcal{V})$ have significant off-sphere components, then (9) may not be satisfied no matter how small $\mathcal{S}$ is chosen to be. If this happens, the line search cannot find an acceptable step, and the algorithm fails.

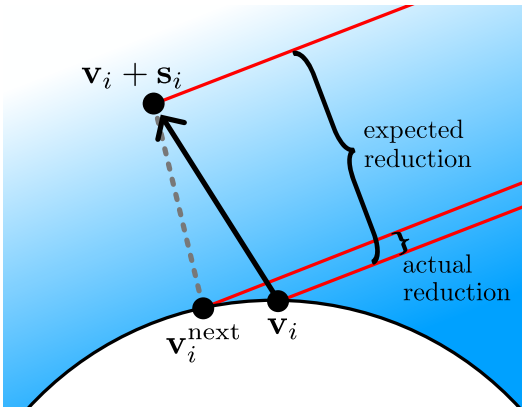

Figure 10: How normalizing vertices changes the cost reduction. The shading represents cost as $\mathbf{v}_i$ varies, with blue representing higher cost.

To fix this problem, we modify the gradient before it's passed into the optimization algorithm. Write $\nabla\mathfrak{C}(\mathcal{V}) = (\mathbf{g}_0, \ldots, \mathbf{g}_{n-1})$. We remove the off-sphere component from each $\mathbf{g}_i$, defining the modified gradient $\mathcal{G}(\mathcal{V}) =$

$(\widetilde{\mathbf{g}}_0, \ldots, \widetilde{\mathbf{g}}_{n-1})$ by

$$\widetilde{\mathbf{g}}_i = \mathbf{g}_i - (\mathbf{g}_i \cdot \mathbf{v}_i)\, \mathbf{v}_i.$$

We then pass $\mathcal{G}(\mathcal{V})$ into the optimization algorithm in place of $\nabla\mathfrak{C}(\mathcal{V})$. In particular, we use $\mathcal{G}(\mathcal{V})$ in place of $\nabla\mathfrak{C}(\mathcal{V})$ in (9). Then the off-sphere component of $\mathcal{S}$ no longer contributes to the dot product in (9), so we can always satisfy the condition with a sufficiently small step. Modifying the gradient in this way means there is no need to modify the optimization algorithm, with the exception of including the normalization step.

If the reader wants to build their own implementation, and modifying the optimization algorithm to include this normalization step is not an option, a possible approach is to represent the mesh vertices with two coordinates each by parametrizing the sphere. Any parametrization of the sphere has at least one singularity, which could cause problems with this approach. One option is to choose a different parametrization for each vertex, so that the singularity for each vertex is opposite its original position, making it very unlikely for problems to arise.

5 Hybrid cartograms

To create the hybrid cartogram, we calculate distortion based on how a triangle is transformed after it is projected to the plane (using a fixed equal-area projection). To describe this, we need the Jacobian matrix of this map projection at each point on the sphere. Let $\omega : \mathrm{S}^2 \to \mathbb{R}^2$ be the target projection. To represent the total derivative of ω as a matrix, we parametrize S^2 using longitude and latitude, denoted λ and φ respectively. The point on S^2 referenced by λ, φ is

$$\big(\cos\varphi\cos\lambda,\ \cos\varphi\sin\lambda,\ \sin\varphi\big).$$

Using this parametrization, we view ω as a map $[-\pi, \pi] \times [-\pi/2, \pi/2] \to \mathbb{R}^2$. At each point $\mathbf{p} \in \mathrm{S}^2$, we choose the basis for the tangent space at $\mathbf{p}$ to align with the graticule, as we did in the previous section. Then, writing the output of ω as (x, y), if $\mathbf{p}$ has the coordinates λ, φ, then the Jacobian matrix of ω at $\mathbf{p}$ is given by

$$\mathrm{J}_\omega(\mathbf{p}) = \begin{bmatrix} \dfrac{1}{\cos\varphi}\dfrac{\partial x}{\partial \lambda} & \dfrac{\partial x}{\partial \varphi} \\ \dfrac{1}{\cos\varphi}\dfrac{\partial y}{\partial \lambda} & \dfrac{\partial y}{\partial \varphi} \end{bmatrix}.$$

For use in the cost function, we measure the Jacobian at the center point of each triangle. Unfortunately, $\mathrm{J}_\omega(\mathbf{p})$ jumps discontinuously as $\mathbf{p}$ crosses an interruption of the projection. In particular, for the pseudocylindrical projections we use, J_ω is discontinuous along the antimeridian. Since the cost function must be continuously differentiable, we blur J_ω near the interruption. Specifically, we choose a continuously differentiable function $H : \mathrm{S}^2 \to \mathbb{R}^{2\times 2}$ such that $H(\mathbf{p}) = \mathrm{J}_\omega(\mathbf{p})$ unless $\mathbf{p}$ is close to the interruption, and use $H(\mathbf{p})$ in place of $\mathrm{J}_\omega(\mathbf{p})$ when calculating distortion.[1] We omit the specific definition of H.

We only take H into account when calculating shape distortion, leaving the scale distortion and area calculations unaffected. This means there is no need to ensure that $H(\mathbf{p})$ always has determinant 1. (Since ω is equal-area, $\mathrm{J}_\omega(\mathbf{p})$ always has determinant 1, but the blur can change this.) For each

[1] Because we choose bases for the tangent spaces to align with the longitude/latitude graticule, and this graticule has singularities at the poles, it is not sufficient to simply have H continuously differentiable as a function $\mathrm{S}^2 \to \mathbb{R}^{2\times 2}$. The correct condition is as follows. Instead of viewing $H(\mathbf{p})$ as a member of $\mathbb{R}^{2\times 2}$ for each $\mathbf{p}$, we can say that $H(\mathbf{p})$ is a linear map $\mathrm{T}_\mathbf{p}\mathrm{S}^2 \to \mathrm{T}_\mathbf{p}\mathrm{S}^2$, where $\mathrm{T}_\mathbf{p}\mathrm{S}^2$ is the tangent space at $\mathbf{p}$. Suppose that $\mathcal{B}(\mathbf{p})$ is a basis for $\mathrm{T}_\mathbf{p}\mathrm{S}^2$ for each $\mathbf{p}$. Then, using $\mathcal{B}(\mathbf{p})$, we can view each $H(\mathbf{p})$ as a linear map $H_\mathcal{B}(\mathbf{p}) : \mathbb{R}^2 \to \mathbb{R}^2$. This creates a function $H_\mathcal{B} : \mathrm{S}^2 \to \mathbb{R}^{2\times 2}$. The condition we need is that $H_\mathcal{B}$ must be continuously differentiable for *any* smooth choice of bases $\mathcal{B}(\mathbf{p})$ (defined over any subset of S^2). In practice, we achieve this by choosing $H(\mathbf{p})$ to be the identity when $\mathbf{p}$ is a pole.

triangle, we multiply the output of H by the matrix K [(1), (2), and (8)], defining

$$\widetilde{K} = H(\mathbf{n})\, K,$$

where $\mathbf{n}$ is the midpoint of the triangle, given by (7). The matrix $\widetilde{K}$ approximately represents the map taking the initial triangle $(\mathring{\mathbf{a}}, \mathring{\mathbf{b}}, \mathring{\mathbf{c}})$ to the projected triangle $\big(\omega(\mathbf{a}), \omega(\mathbf{b}), \omega(\mathbf{c})\big)$. We use $\widetilde{K}$ to calculate δ^{shape}, with the scale distortion and area just using K as before.

Recall that when $\mathbf{n}$ moves in the east–west direction, the basis for the tangent space at $\mathbf{n}$ rotates, and that we ignore this effect when doing the calculations in section 4. The rotation of the basis has the effect of multiplying K by a rotation matrix on its left (see fig. 11). Since $H(\mathbf{n})$ is subsequently multiplied on the left, this has a non-rotational effect on $\widetilde{K}$, so it must be taken into account when calculating the gradient.

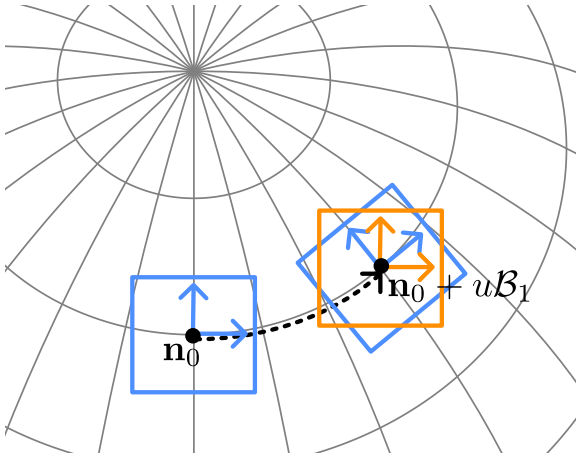

Figure 11: The effect on the local basis caused by moving east in the Northern Hemisphere. Orange represents the result of moving the basis at $\mathbf{n}_0$ without rotating it, as we do in section 4. From the perspective of the graticule-aligned basis (blue), moving east causes a clockwise rotation.

For the purpose of finding derivatives, we call the current midpoint of the triangle $\mathbf{n}_0$ and view the midpoint $\mathbf{n}$ as varying in the tangent plane at $\mathbf{n}_0$. We write $\mathbf{n} = \mathbf{n}_0 + u\mathcal{B}_1 + v\mathcal{B}_2$, $u, v \in \mathbb{R}$, where $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is the orthonormal

basis for the tangent space at $\mathbf{n}_0$. Since the notational distinction between $\mathbf{n}_0$ and $\mathbf{n}$ is just for the purpose of taking derivatives, all of these derivatives are evaluated at $\mathbf{n} = \mathbf{n}_0$. We represent the induced rotation relative to the basis vectors by the rotation matrix $R(u, v)$. For example, in fig. 11, $R(u, v)$ is a clockwise rotation. The formula for $\widetilde{K}$ (for the purpose of finding derivatives) becomes

$$\widetilde{K} = H(\mathbf{n})\, R(u, v)\, K,$$

so each derivative of $\widetilde{K}$ (evaluated at $\mathbf{n} = \mathbf{n}_0$) is

$$\begin{aligned}
\widetilde{K}' &= [H(\mathbf{n})\, R(u, v)\, K]' \\
&= [H(\mathbf{n})\, R(u, v)]'\, K + [H(\mathbf{n})\, R(u, v)]\, K' \\
&= [H(\mathbf{n})\, R(u, v)]'\, K + [H(\mathbf{n}_0)\, R(0, 0)]\, K' \\
&= [H(\mathbf{n})\, R(u, v)]'\, K + H(\mathbf{n}_0)\, K'.
\end{aligned}$$

The calculations for K and K' are covered in the previous section, so it only remains to find $[H(\mathbf{n})\, R(u, v)]'$.

It suffices to find the derivatives of $H(\mathbf{n})\, R(u, v)$ with respect to u and v; the derivatives with respect to the components of the triangle's vertices are then found using the chain rule. We can write $H(\mathbf{p}) = H(\lambda, \varphi)$ using the usual parametrization of the sphere. Let λ_0, φ_0 be the coordinates corresponding to $\mathbf{n}_0$. The derivative with respect to v is simple:

$$\begin{aligned}
\frac{\partial}{\partial v}[H(\mathbf{n})\, R(u, v)]\Big|_{\mathbf{n}=\mathbf{n}_0} &= \frac{\partial H}{\partial v}(\mathbf{n}_0)\, R(0, 0) + H(\mathbf{n}_0)\, \frac{\partial R}{\partial v}(0, 0) \\
&= \frac{\partial H}{\partial v}(\mathbf{n}_0)\, I + H(\mathbf{n}_0)\, 0 \\
&= \frac{\partial H}{\partial v}(\mathbf{n}_0) = \frac{\partial H}{\partial \varphi}(\lambda_0, \varphi_0),
\end{aligned}$$

since varying v (i.e., moving north–south) induces no rotation of the basis.

Deriving with respect to u, we have

$$\begin{aligned}\frac{\partial}{\partial u}[H(\mathbf{n})\,R(u,v)]\Big|_{\mathbf{n}=\mathbf{n}_0} &= \frac{\partial H}{\partial u}(\mathbf{n}_0)\,R(0,0) + H(\mathbf{n}_0)\,\frac{\partial R}{\partial u}(0,0)\\ &= \frac{\partial H}{\partial u}(\mathbf{n}_0)\,I + H(\mathbf{n}_0)\,\frac{\partial R}{\partial u}(0,0)\\ &= \frac{1}{\cos\varphi_0}\,\frac{\partial H}{\partial \lambda}(\lambda_0,\varphi_0) + H(\lambda_0,\varphi_0)\,\frac{\partial R}{\partial u}(0,0), \qquad (10)\end{aligned}$$

since a change in u is $\cos\varphi_0$ times the corresponding change in the longitude λ.

It remains to find $\dfrac{\partial R}{\partial u}(0,0)$. To find this, we view $\mathbf{n}$ as varying along the latitude line at φ_0. Having moved a distance of u along this line, the basis $\mathcal{B}$ has rotated by some angle $\theta(u)$. Then we have

$$R(u,0) = \begin{bmatrix}\cos(-\theta(u)) & -\sin(-\theta(u))\\ \sin(-\theta(u)) & \cos(-\theta(u))\end{bmatrix} = \begin{bmatrix}\cos\theta(u) & \sin\theta(u)\\ -\sin\theta(u) & \cos\theta(u)\end{bmatrix}.$$

The negation appears because $R(u,0)$ represents the rotation *from the perspective of* the basis. Then

$$\frac{\partial R}{\partial u}(0,0) = \frac{\partial}{\partial u}\begin{bmatrix}\cos\theta(u) & \sin\theta(u)\\ -\sin\theta(u) & \cos\theta(u)\end{bmatrix}\Bigg|_{u=0} = \theta'(0)\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}. \qquad (11)$$

The value $\theta'(0)$ is the radians of rotation of the basis per distance travelled along the latitude line. Since the basis is aligned with the line, this is the radians of rotation of the line per distance travelled. By definition, this is the geodesic curvature of this line. To find this value, we find the total geodesic curvature of the latitude line using the Gauss–Bonnet theorem. The area of the region of S^2 enclosed (to th**e** north) by the latitude line at φ_0 is $2\pi(1-\sin\varphi_0)$, and the Gaussian curvature of the unit sphere is 1, so the total geodesic curvature is $2\pi - 1\cdot 2\pi(1-\sin\varphi_0) = 2\pi\sin\varphi_0$. Since the length of

the latitude line is $2\pi \cos \varphi_0$, the geodesic curvature at each point is

$$\frac{2\pi \sin \varphi_0}{2\pi \cos \varphi_0} = \tan \varphi_0,$$

so $\theta'(0) = \tan \varphi_0$. Combining this fact with (10) and (11), we finally get

$$\begin{aligned}
&\frac{\partial}{\partial u}[H(\mathbf{n})\, R(u,v)]\Big|_{\mathbf{n}=\mathbf{n}_0} \\
&= \frac{1}{\cos \varphi_0} \frac{\partial H}{\partial \lambda}(\lambda_0, \varphi_0) + \tan(\varphi_0)\, H(\lambda_0, \varphi_0) \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.
\end{aligned}$$

This completes the calculation of the derivatives of $\widetilde{K}$, which completes the calculation of $\nabla \mathfrak{C}(\mathcal{V})$.

Unlike for the sphere cartogram, it does not make sense to manually rotate the sphere after optimizing, because that would reintroduce distortion that we worked hard to optimize away. This means that, if we want the interruption to remain in a specific place, we must enforce this by modifying the cost function. For our cartograms, we place the interruption at 169° west, through the Bering Strait. We rotate the borders so that the Cartesian coordinates on this line satisfy $y = 0$, which makes it simple to compute the deviation of points from this line.

We add a term to the definition of $\mathfrak{D}(\mathcal{V})$ that measures the deviation of points on the antimeridian from the interruption line. (Since the mesh is based on an octahedron, there is a line of mesh vertices that falls on the antimeridian.) We also add a term that fixes the North Pole in place. Let M be the set of all vertices initially on the antimeridian (not including the poles), and let $\mathbf{p}^{\text{north}}$ be the vertex initially at the North Pole. We then

modify the definition of the total distortion (5) to be

$$\mathfrak{D}(\mathcal{V}) = \sum_{T \in \mathcal{T}} \mathring{m}_T \big(w_T^{\text{shape}} \delta_T^{\text{shape}} + w_T^{\text{scale}} \delta_T^{\text{scale}} \big) + w^{\text{pole}} \big((\mathbf{p}_x^{\text{north}})^2 + (\mathbf{p}_y^{\text{north}})^2 \big) + \sum_{\mathbf{v} \in M} w_{\mathbf{v}}^{\text{antimer}} \mathbf{v}_y^2,$$

where w^{pole} and $(w_{\mathbf{v}}^{\text{antimer}})_{\mathbf{v} \in M}$ are nonnegative weights. We set w^{pole} to be very high so that the North Pole is fixed, which helps to maintain the shapes of regions near the pole. We set $w_{\mathbf{v}}^{\text{antimer}}$ highest for vertices in the north, to ensure the interruption passes through the Bering Strait. We set $w_{\mathbf{v}}^{\text{antimer}}$ lower for vertices in the mid latitudes, and set $w_{\mathbf{v}}^{\text{antimer}} = 0$ around Antarctica.

To create the Liquid Earth projection (fig. 7), we simply ignore the cartographic error, minimizing $\mathfrak{D}(\mathcal{V})$ instead of $\mathfrak{C}(\mathcal{V})$. We set the intended scale s_T of each triangle to 1. We weight triangles than contain land 100 times more than water triangles, otherwise weighting different triangles equally. To produce a nearly equal-area map with minimal shape distortion, we minimize multiple times, each time weighting scale distortion higher. We start with scale distortion weighted 0.1 times more than shape distortion and end with scale distortion weighted 100 times more than shape distortion. Prioritizing shape distortion in the beginning is necessary to find a good local minimum. We do most of the optimization using an octahedral mesh of 18432 triangles ($8 \cdot 48^2$), then subdivide in particularly distortion-prone places, then finish the optimization. To improve the look of the graticule, we fix the North Pole exactly in place, similar to how we fix the mesh vertices to the sphere in our sphere and hybrid cartograms.

6 Algorithm performance

In this section, we briefly lay out the runtime and accuracy of our cartogram algorithm.

For the tables below, we calculate relative cartographic error for each region R as $(\mu_R - p_R)/p_R$, as described in previous sections. For the sphere and hybrid cartograms, the computation of the transformed area μ_R is based on the transformed mesh without considering the projection to the sphere. This projection step introduces a small amount of relative error that the tables do not capture.

To estimate this error, recall that the area of each triangle is computed after projecting the vertices to the tangent plane on the triangle's midpoint $\mathbf{n}$. Consider an infinitesimal neighborhood of a point $\mathbf{v}^{\text{tp}}$ on this triangle in the tangent plane. Using similar triangles, it can be seen that the projection from the plane triangle to the spherical triangle scales this neighborhood by a factor of $\|\mathbf{v}^{\text{tp}}\| = \sqrt{1 + \|\mathbf{v}^{\text{tp}} - \mathbf{n}\|^2}$. The relative area error of the projection on this neighborhood is then $\sqrt{1 + \|\mathbf{v}^{\text{tp}} - \mathbf{n}\|^2} - 1 \approx \|\mathbf{v}^{\text{tp}} - \mathbf{n}\|^2/2$. This means the maximum error incurred on this triangle is on the order of the area of the triangle. Since the mesh was subdivided so that the transformed triangle areas are less than $4\pi/2048$, and since the squared radius of an equilateral triangle is $4/3^{3/2}$ times its area, the maximum (relative) error incurred is less than 0.003 in most cases.

The runtimes below do not include the time taken to compute the area portion values $\psi_{R,T}$. Depending on the resolution of the region border data, this can take several hours in Python. However, since the code for this is not vectorized, it could be greatly sped up by using a compiled programming language. Creating the meshes takes a negligible amount of time.

As described in section 2, we do the optimization in stages, minimizing $\mathfrak{C}(\mathcal{V}) = W^{\text{error}}\,\mathfrak{E}(\mathcal{V}) + W^{\text{dist}}\,\mathfrak{D}(\mathcal{V})$ multiple times, setting W^{dist} to be smaller each time. At each stage, finding a local minimum of $\mathfrak{C}(\mathcal{V})$ requires thousands of search steps, stopping when $\nabla\mathfrak{C}(\mathcal{V}) = 0$ is achieved. A key performance improvement is possible by not requiring a true local minimum to be found, instead stopping when the size of the gradient is below a certain threshold. At each stage, we choose a threshold value $\gamma > 0$ and stop the optimization

when $\|\nabla \mathfrak{C}(\mathcal{V})\|_\infty < \gamma$. (For the sphere and hybrid cartograms, we remove the off-sphere component of $\nabla \mathfrak{C}(\mathcal{V})$ before doing this check.) For the first stage, we choose $W^{\text{error}} = 1$, $W^{\text{dist}} = 0.1$, and $\gamma = 0.01$. Going from one stage to the next, we multiply W^{dist} and γ by 0.1 while keeping W^{error} fixed.

Our implementation uses NumPy; the runtimes below were recorded for an AMD Ryzen 5 5600 CPU. We may sacrifice accuracy for speed by stopping after a smaller number of stages. The last few stages are unnecessary for the sphere and hybrid cartograms because of the aforementioned error caused by projection.

Plane cartogram

Stage	Steps	Total time	Median rel. error	Max rel. error
1	76375	1h 4m 17s	0.0425	4.19
2	2626	1h 6m 27s	0.00472	3.91
3	4345	1h 9m 58s	0.000502	2.59
4	304	1h 10m 13s	$4.56 \cdot 10^{-5}$	1.06
5	6331	1h 15m 20s	$4.81 \cdot 10^{-6}$	0.256
6	12212	1h 23m 40s	$4.89 \cdot 10^{-7}$	0.0355
7	248	1h 23m 50s	$6.60 \cdot 10^{-8}$	0.00377
8	4209	1h 26m 37s	$6.14 \cdot 10^{-9}$	0.000378
9	232	1h 26m 46s	$6.40 \cdot 10^{-10}$	$3.78 \cdot 10^{-5}$
10	309	1h 26m 59s	$4.71 \cdot 10^{-11}$	$3.78 \cdot 10^{-6}$

Sphere cartogram

Stage	Steps	Total time	Median rel. error	Max rel. error
1	48362	1h 19m 38s	0.0410	3.79
2	4369	1h 26m 39s	0.00446	3.57
3	526	1h 27m 27s	0.000472	2.53
4	1391	1h 29m 43s	$5.05 \cdot 10^{-5}$	1.04
5	207	1h 30m 0s	$4.95 \cdot 10^{-6}$	0.262
6	2270	1h 32m 52s	$5.53 \cdot 10^{-7}$	0.0367
7	361	1h 33m 19s	$6.62 \cdot 10^{-8}$	0.00389
8	2431	1h 36m 19s	$6.56 \cdot 10^{-9}$	0.000391
9	202	1h 36m 34s	$5.54 \cdot 10^{-10}$	$3.92 \cdot 10^{-5}$
10	108	1h 36m 43s	$4.64 \cdot 10^{-11}$	$3.92 \cdot 10^{-6}$

Hybrid cartogram

Stage	Steps	Total time	Median rel. error	Max rel. error
1	69268	2h 9m 51s	0.0518	4.64
2	9289	2h 27m 12s	0.00578	4.28
3	5105	2h 37m 1s	0.000657	2.95
4	4649	2h 45m 41s	$5.88 \cdot 10^{-5}$	1.20
5	9197	3h 2m 42s	$6.05 \cdot 10^{-6}$	0.303
6	35513	3h 58m 5s	$6.60 \cdot 10^{-7}$	0.0442
7	6575	4h 8m 8s	$7.03 \cdot 10^{-8}$	0.00471
8	1476	4h 10m 22s	$7.54 \cdot 10^{-9}$	0.000474
9	902	4h 11m 44s	$7.60 \cdot 10^{-10}$	$4.74 \cdot 10^{-5}$
10	772	4h 12m 59s	$6.44 \cdot 10^{-11}$	$4.74 \cdot 10^{-6}$

7 Final Remarks

The algorithm we present in this paper gives a significant improvement in shape preservation over existing methods. It achieves very low cartographic error in the sphere and hybrid cases and arbitrarily low error in the plane case, barring floating-point issues. Potential areas of future research include

improving the algorithm's runtime and testing whether the resulting cartograms are rated favorably by map readers.

It is unclear how much mesh subdivision needs to be done to guarantee that an accurate cartogram is possible. For example, if two regions lie entirely within a single triangle, it is clearly impossible to make both regions' areas match their populations. Finding a sufficient condition for area accuracy to be possible could be of independent mathematical interest. It is also unclear why the weight on scale distortion has a strong effect on runtime. Resolving these questions is a potential area of future research.

The on-sphere techniques presented here could be applied to other cartogram algorithms. Though Li and Aryana [8] adapted the diffusion method to the sphere, the same has not been done for rubber-sheet cartograms. Such on-sphere adaptations of cartogram methods could also be made to take into account the projection to the plane, similar to our hybrid cartograms.

References

[1] Armijo, L. (1966). Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1), 1–3. https://doi.org/10.2140/pjm.1966.16.1

[2] Dougenik, J. A., Chrisman, N. R., Niemeyer, D. R. (1985). An algorithm to construct continuous area cartograms. *The Professional Geographer*. https://doi.org/10.1111/j.0033-0124.1985.00075.x

[3] Gastner, M., Newman, M. E. J. (2004). Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences*. https://doi.org/10.1073/pnas.0400280101

[4] Jawaherul Alam, M., Kobourov, S. G., Veeramoni, S. (2015). Quantitative Measures for Cartogram Generation Techniques. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.12647

[5] Kronenfeld, B. J. (2017). Manual construction of continuous cartograms through mesh transformation. *Cartography and Geographic Information Science*, 45(1), 76–94. https://doi.org/10.1080/15230406.2016.1270775

[6] Kunimune, J. H. (2020). Minimum-error world map projections defined by polydimensional meshes. *International Journal of Cartography*. https://doi.org/10.1080/23729333.2020.1824174

[7] Kunimune, J. H. (2023). Introducing the Elastic projections. Web. https://kunimune.blog/2023/12/29/introducing-the-elastic-projections/

[8] Li, Z., Aryana, S. (2017). Diffusion-based cartogram on spheres. *Cartography and Geographic Information Science*, 45:5, 464–475. https://doi.org/10.1080/15230406.2017.1408033

[9] Loncaric, M. (2024). Map Projections 2: Solving Numerically. Web. https://graphallthethings.com/posts/map-projections-2

[10] Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151), 773–782. https://doi.org/10.1090/S0025-5718-1980-0572855-7

[11] Nusrat, S., Kobourov, S. (2016). The state of the art in cartograms. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.12932

[12] Sargent, R. C. (2024). Introducing the Liquid Earth projection. Web. https://rsargentmath.github.io/posts/liquid_earth/

[13] Sun, S. (2013). A fast, free-form rubber-sheet algorithm for contiguous area cartograms. *International Journal of Geographic Information Science*, 27 (3): 567–93. https://doi.org/10.1080/13658816.2012.709247

[14] Sun, S. (2013). An optimized rubber-sheet algorithm for continuous area cartograms. *The Professional Geographer*. https://doi.org/10.1080/00330124.2011.639613

[15] Sun, S. (2015). Carto3F program: A fast, free-form algorithm implementation for area cartograms. Web. http://www.sunsp.net/download.html