

## CHAPTER VI

### MAINTAINING AREA

#### 6.1 Overview

The first goal in constructing a cartogram is to size each region to its desired proportional area. The second goal is to maintain recognizable shape. Within our algorithm, a key component of the second goal is holding the region *areas* fixed while their *shapes* are readjusting. Without the ability to maintain area, our method would simply alternate between an unrecognizable cartogram and the original map. This holding ability is enabled through dynamic area constraints, a mechanism that cancels only those components of the shape forces that will lead to change in area. The resulting constrained dynamic environment enables shape adjustments to occur, but not at the expense of the maintenance of desired areas.

#### 6.2 Constrained Dynamics

The intent of constrained particle dynamics, as detailed in tutorial form by Witkin in [27], is to ensure that particles obey specific geometric constraints while, at the same time, adhere to physical laws. Our method modifies Witkin's approach to utilize Aristotelian dynamics, in which forces yield velocities rather than acceleration.

In the context of cartograms, we desire to constrain each region to its desired representative area. The area of a polygonal region with  $n$  vertices  $(x_i, y_i)$  is given in [8] as

$$\frac{1}{2} \sum_{i=0}^{n-1} (y_i + y_{i \oplus 1})(x_{i \oplus 1} - x_i), \quad (6.1)$$

where the operator  $\oplus$  indicates addition modulo  $n$ .

We describe the Aristotelian dynamics of our physical system through a set of differential equations involving the current system state and applied forces upon it. If  $\mathbf{x}_k$  is a state vector of the position of the vertices in region  $k$ , we can require the region to maintain its desired area,  $A_k$ , through the constraint equation

$$C_k(\mathbf{x}_k) = \frac{1}{2} \sum_{j=0}^{n-1} (y_j + y_{j\oplus 1}) (x_{j\oplus 1} - x_j) - A_k, \quad (6.2)$$

where  $j$  is an index of the  $n$  vertices in the region  $k$ . Rather than process each region separately, we define a state vector  $\mathbf{x}$  of all vertex positions, a state vector  $\dot{\mathbf{x}}$  of all vertex velocities, and a vector function  $C(\mathbf{x})$  that concatenates all of the scalar constraints from equation 6.2. Aristotelian dynamics relates point velocity directly to the applied force on the system by the global equation

$$\mathbf{F}_A = \dot{\mathbf{x}}, \quad (6.3)$$

with the assumption that all of the map vertices have unit mass. To guarantee that the applied forces do not violate our constraints, we add a set of constraint forces  $\mathbf{F}_C$ , yielding

$$\dot{\mathbf{x}} = \mathbf{F}_A + \mathbf{F}_C. \quad (6.4)$$

If we assume that all of the constraints are met in the initial state of the system<sup>1</sup>, then to ensure that the constraints will continue to be met we must guarantee that the constraint rate of change, or velocity, is fixed at zero, or

$$\dot{\mathbf{C}} = 0. \quad (6.5)$$

If  $\mathbf{J}$  is the *jacobian* matrix  $\partial C/\partial \mathbf{x}$ , then by the chain rule we have

$$\dot{\mathbf{C}} = \mathbf{J}\dot{\mathbf{x}}. \quad (6.6)$$

Substituting (6.4) and (6.5) into (6.6) we get

$$\mathbf{J}\mathbf{F}_C = -\mathbf{J}\mathbf{F}_A. \quad (6.7)$$

Since our constraints must never be violated, the vertex velocities  $\dot{\mathbf{x}}$  must not have a component in the direction of the gradient of the constraints, thus

$$\mathbf{J}\dot{\mathbf{x}} = 0. \quad (6.8)$$

In order for the constraint forces to preserve the principle of *virtual work*, we must require that they perform no work, namely

$$\mathbf{F}_C \cdot \dot{\mathbf{x}} = 0. \quad (6.9)$$

---

<sup>1</sup> This is not necessarily the case in our application, since we commonly do not achieve *all* desired areas the *first* time. This disparity allows the region areas to deviate somewhat but, as desired areas are achieved over time, the ability of the constraints to *hold* the areas fixed is magnified.

Equations 6.8 and 6.9 can be met only when the constraint forces are scalar multiples of the columns of the jacobian. This occurs when the constraint forces are parallel to the gradient of their associated constraint function. This can be expressed in the form

$$\mathbf{F}_C = \mathbf{J}^T \boldsymbol{\lambda}, \quad (6.10)$$

where  $\boldsymbol{\lambda}$  is an unknown vector of Lagrange multipliers. Substituting equation 6.10 into 6.7 we get the final constrained dynamics equation

$$\mathbf{J}\mathbf{J}^T \boldsymbol{\lambda} = -\mathbf{J}\mathbf{F}_A, \quad (6.11)$$

where the matrix  $\mathbf{J}\mathbf{J}^T$  is a square  $m \times m$  matrix and  $\boldsymbol{\lambda}$  is of length  $m$  for a system of  $m$  constraints. We solve for  $\boldsymbol{\lambda}$  using the bi-conjugate gradient approach [18]. Substituting the result into equation 6.10 yields the necessary constraint forces.

### 6.3 Dynamic Area Constraints

As given in the pseudocode in Figure 4.8, the area constraining forces are computed *following* the distribution of the shape restoration forces. This is necessary since the constraints serve as a mediator, canceling shape forces as necessary to keep the region areas fixed. As given in equation 6.10, we must compute the transposed matrix  $\mathbf{J}$  and then solve for the vector  $\boldsymbol{\lambda}$ .

The components of the matrix  $\mathbf{J}$  are found by taking the partial derivative of equation 6.2, giving

$$\frac{\partial C_i(\mathbf{x})}{\partial \mathbf{x}_j} = \frac{1}{2} [\dot{y}_{j-1} - \dot{y}_{j+1}, \dot{x}_{j+1} - \dot{x}_{j-1}], \quad 1 \leq i \leq m, \quad 1 \leq j \leq n. \quad (6.12)$$

$\mathbf{J}$  is an  $m \times n$  matrix as shown in Figure 6.1 where  $m$  is the number of area constraints (i.e. the number of regions) and  $n$  is the total number of vertices. This matrix could be quite huge for maps with hundreds of vertices and many regions. However, each of our constraints typically influences only a handful of vertices, with the derivative of all other components being zero. Therefore, we implement  $\mathbf{J}$  as a *sparse matrix*, as detailed in [27], with  $m$  indexed lists of the non-zero entries in each row. Equations 6.10 and 6.11 require the implementation of two operations upon the sparse matrix: *matrix times vector*

and *matrix-transpose times vector*. Both operations are straightforward, performing normal matrix multiplication using the indexed offsets within each list.

$$\begin{bmatrix} \frac{\partial C_1}{\partial x_1} & \frac{\partial C_1}{\partial x_2} & \cdots & \frac{\partial C_1}{\partial x_n} \\ \frac{\partial C_2}{\partial x_1} & \frac{\partial C_2}{\partial x_2} & \cdots & \frac{\partial C_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial C_m}{\partial x_1} & \frac{\partial C_m}{\partial x_2} & \cdots & \frac{\partial C_m}{\partial x_n} \end{bmatrix}$$

**Figure 6.1:** The jacobian matrix  $\mathbf{J}$ .

To find  $\lambda$  in equation 6.11 we first compute  $\mathbf{J}\mathbf{J}^T$  and  $\mathbf{J}\mathbf{F}_A$ , where  $\mathbf{F}_A$  represents the applied shape forces in our system. The bi-conjugate gradient method [22] solves our sparse system for  $\lambda$ . Note that when the system is overconstrained there may be no exact solution to equation 6.11. In this case, the bi-conjugate gradient method gives a solution which will result in the least mean squared error.

Having found  $\mathbf{J}$  and  $\lambda$ , we solve equation 6.10 for our constraint forces and apply them to the vertices. The only other forces applied during the shape restoration process are from topological constraints to ensure the integrity of the map topology. Since the topological constraints *follow* the dynamic area constraints, some of our maintaining forces can be canceled to preserve the topology. The cancellation of area constraint forces, numerical drift, and an overconstrained system can result in regions drifting slightly from their fixed area. However, our iterative relaxation process periodically adjusts area; thus correcting these problems over time.