

CHAPTER VII

MAINTAINING TOPOLOGICAL INTEGRITY

7.1 Overview

During the resizing process when edges and vertices are moved, the polygonal regions may twist or invert themselves, as demonstrated in Figure 7.1a. To prevent this break in the topology or connectivity of our map by inverted or self-intersecting regions, our method employs three topology maintaining mechanisms: hinge constraints, edge constraints, and intersection penalty constraints. These constraints will override area and shape spring forces, as well as dynamic area constraint forces, as required to correct topological discrepancies.

7.2 Hinge Constraints

A hinge constraint simply restricts the angle between two adjacent edges from opening or closing beyond its limits. When the angle approaches 0° or 360° as shown in Figure 7.1a, equal forces are applied to the incident edge endpoints to ensure that the angle, and consequently the region, does not invert. An opposite force is applied to the hinge point, ensuring that the net hinge force on the assembly sums to zero. This preserves the principal of virtual work and guarantees that we have passive, or lossless, constraints.

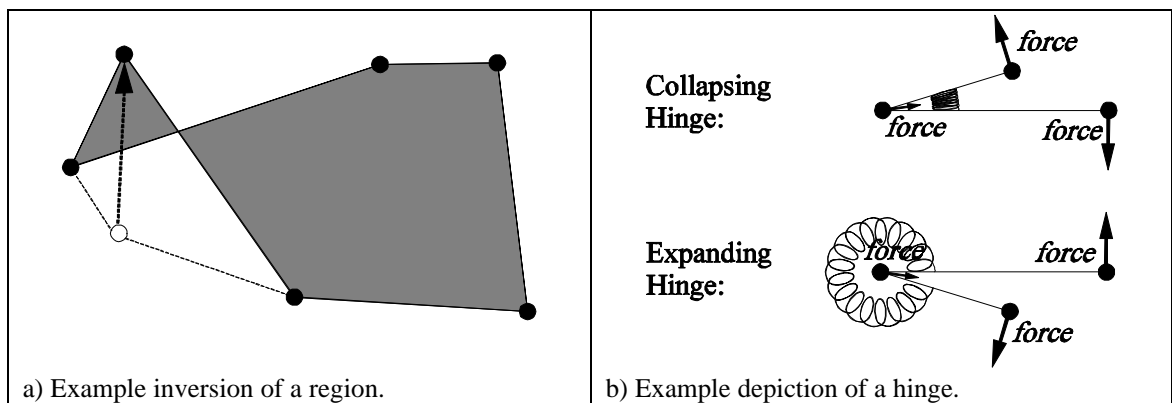


Figure 7.1: The hinge topological constraint.

We desire a force function that gradually introduces force as the hinge angle nears 0° or 360° and exhibits maximum force when the hinge exceeds its limits. Finding an acceptable force equation curve was a challenge: either the hinge prevented edges from even approaching each other by overcompensating with too much force too early or it allowed inversions by undercompensating with not enough force a little too late.

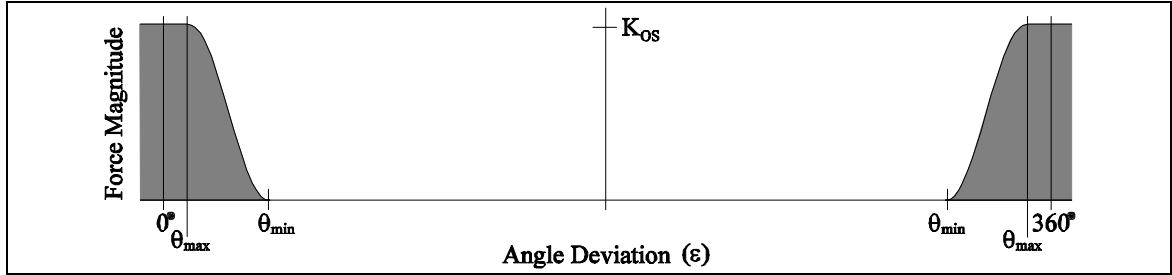


Figure 7.2: Graph of the hinge constraint force function.

After considerable experimentation we determined that the force function graphed in Figure 7.2 produced the desired effect. The function defines the magnitude of force for a hinge with angle θ to be the function

$$F_{\text{HC}} = \begin{cases} K_{\text{HC}}, & \theta \leq 0^\circ, \theta \geq 360^\circ \\ K_{\text{HC}}, & \cos\theta > \cos\theta_{\text{max}} \\ K_{\text{HC}} \left(\frac{\cos\theta - \cos\theta_{\text{min}}}{\cos\theta_{\text{max}} - \cos\theta_{\text{min}}} \right)^h, & \cos\theta_{\text{min}} < \cos\theta < \cos\theta_{\text{max}} \\ 0, & \cos\theta \leq \cos\theta_{\text{min}}, \end{cases} \quad (7.1)$$

where K_{HC} is a user-specified constant and θ_{min} and θ_{max} define a “window” across which force is applied gradually. We have found that a value of 10 for the h power term, 5° for θ_{max} , and 30° for θ_{min} produces good results.

As seen in the graph, a hinge will encounter resistance only when its angle falls below θ_{min} of 0° or 360° . A gradually increasing resistance force is applied as the hinge angle approaches θ_{max} . Beyond θ_{max} , the hinge is projecting maximum force upon its vertices to prevent an inversion (or to *correct* one if the angle is already beyond 0° or 360°).

The force directions for each of the two hinge endpoints are found as perpendicular unit vectors pointing towards the interior of the hinge. The force magnitude is projected equally in each force direction, with a positive force closing the hinge and a negative force opening the hinge. Since the two force directions are rarely ever (and hopefully never) exact opposites, a negated sum of the two forces is applied on the hinge point to ensure that no net force is added to the system.

7.3 Edge Constraints

Another topological problem occurs when an edge is driven progressively smaller until it flips to, essentially, negative length. To counteract this, we assign each edge an edge constraint, as depicted in Figure 4-9b, that exhibits force outward along the edge's axis when its length approaches zero. The forces upon the edge's two endpoints are equal and opposite, keeping the integrity of our physically based environment.

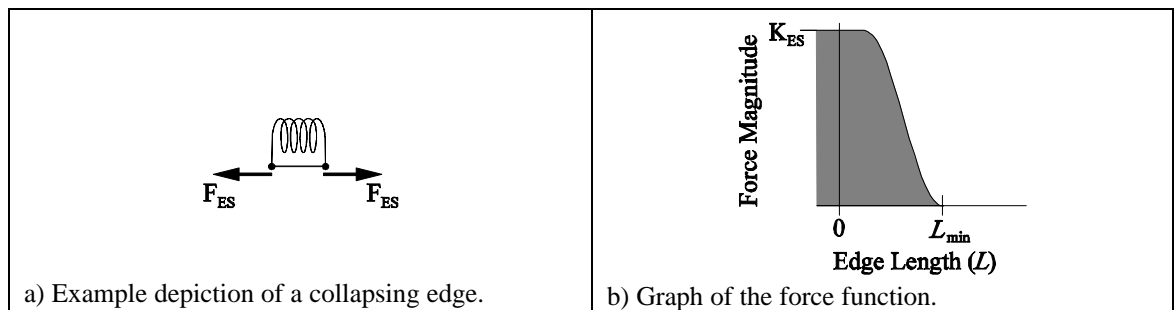


Figure 7.3: The edge topological constraint.

Our definition for the magnitude of an edge constraint force upon an edge \mathbf{e} is

$$F_{ES} = \begin{cases} 0, & \|\mathbf{e}\| > \|L_{min}\| \\ \frac{1}{2} K_{ES} (\|\mathbf{e}\| - L_{min}) \left[1 + \cos^p \left(\frac{\|\mathbf{e}\|}{\|L_{min}\|} \pi \right) \right], & \|\mathbf{e}\| \leq \|L_{min}\|, \end{cases} \quad (7.2)$$

where K_{ES} is a user-specified constant and L_{min} is the minimum acceptable length. We found that a p power term of 10 produces acceptable results. As shown in the function graph in Figure 7.3b, an edge falling below the minimum length L_{min} will experience an outward force by its edge constraint. A gradually increasing force is applied as the edge

length approaches zero. This force is projected parallel to the edge in a direction outward from the edge's center.

7.4 Intersection Penalty Constraints

Hinge and edge constraints together effectively prevent the “bow tie” inversion shown in the example in Figure 7.1a. Thus, the only possible topological violation remaining is the self-intersection of a region. Generally this involves two or more edges crossing over one or more other edges and can be quite involved, as demonstrated in the examples in Figure 7.4.

The first intersection category involves two adjacent edges intersecting a third edge, as shown in the Type “A” example in Figure 7.4a. A slightly more complex intersection is found in the Type “B” example in Figure 7.4b, where a group of edges have penetrated through a single edge. The Type “C” category shown in Figure 7.4c consists of the crossing of two pairs of adjacent edges. An extension to “C” is the Type “D” example shown in Figure 7.4d, where a group of edges has penetrated another group of edges.

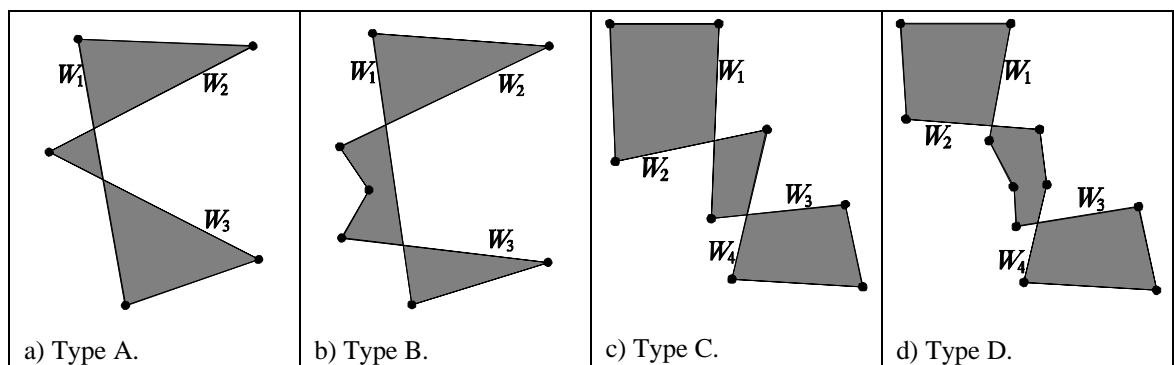


Figure 7.4: Examples of topological violations where a region has self-intersected.

In all cases of self-intersection, restoration of the topological integrity can be accomplished by exerting opposing penalty forces to all the points involved until the edges no longer intersect. The penalty forces should be distributed such that their sum is zero, thereby preserving the principle of virtual work. Successful implementation of

penalty forces can be complex when multiple intersections overlap or when many edges are involved. Additionally, perimeter intersection tests are also required to ensure that one region does not implant itself inside another.

7.4.1 Intersection Detection

The current method of detecting intersections works in most cases. Cases that are not handled correctly include those where we begin detecting intersections from *within* an intersection. If detection were to begin at the vertex P in Figure 7.4a, an inverse intersection would be detected since it is assumed the initial vertex is in a valid state. This would propel W_1 to the right of W_2 to W_3 , instead of moving P to the right of W_1 . Others that are not consistently handled correctly are complex Type “C” and “D” intersections in close proximity to each other (or even overlapping). In such cases, the detection algorithm finds multiple crossover points in the same vicinity and can pair up the intersection points incorrectly. A more robust detection algorithm is proposed in Chapter X but has not yet been implemented.

Intersection detection of the 4 types of intersections shown in Figure 7.4 proved to be a formidable challenge. The type classifications evolved as complex situations arose but they could easily be merged into a single definition that handles all cases.

Our method detects type A and B intersections using the pseudocode in Figure 7.5 and then detects types C and D using the pseudocode in Figure 7.6. The first order of business is to re-evaluate previous intersections and remove the ones that have been corrected. Referring to the examples in Figures 7.4a and b, we detect a type A or B intersection by first locating the intersecting pair of edges W_1 and W_2 and then searching for W_3 . To locate the type C or D example intersections depicted in Figures 7.4c and d, we first locate the intersecting pair of edges W_1 and W_2 and then search the edges following W_1 and preceding W_2 to find W_3 and W_4 , respectively. Note that all region edges are stored in clockwise order.

```

for each Region do
  RemoveCorrectedIntersections (Region);
  { ——— Find Type A & B Intersections ——— }
  for  $w1 = 1$  to numRegionEdges
     $w2 = w1 + 1$ ;
    found = false;
    while not found do
      if Intersection ( $w1, w2$ ) then
         $w3 = w2 + 1$ ;
        while  $w3 <> w1 - 1$  do
          if Intersection ( $w1, w3$ ) then
            if UniqueIntersection ( $w1, w2, w3$ ) then
              ComputePenaltyForces ( $w1, w2, w3$ );
              found = true;
            if  $w3 + 1 = \text{numRegionEdges}$  then
               $w3 = 0$ ;
            else  $w3 = w3 + 1$ ;
           $w2 = w2 + 1$ ;

```

Figure 7.5: Pseudocode for detecting type A and B intersections.

```

for each Region do
  RemoveCorrectedIntersections (Region);
  { ——— Find Type C & D Intersections ——— }
  for  $w1 = 1$  to numRegionEdges
     $w2 = w1 + 2$ ;
    found = false;
    while not found do
      if Intersection ( $w1, w2$ ) then
         $w3 = w1 + 1$ ;
        while  $w3 \leq \text{numRegionEdges}$  and not found do
           $w4 = w2 - 1$ ;
          while  $w4 <> w1 + 1$  and  $w3 < w4$  and not found
            if Intersection ( $w3, w4$ ) then
              if UniqueIntersection ( $w1, w2, w3, w4$ ) then
                ComputePenaltyForces ( $w1, w2, w3, w4$ );
                found = true;
               $w4 = w4 - 1$ ;
             $w3 = w3 + 1$ ;
           $w2 = w2 + 1$ ;

```

Figure 7.6: Pseudocode for detecting type C and D intersections.

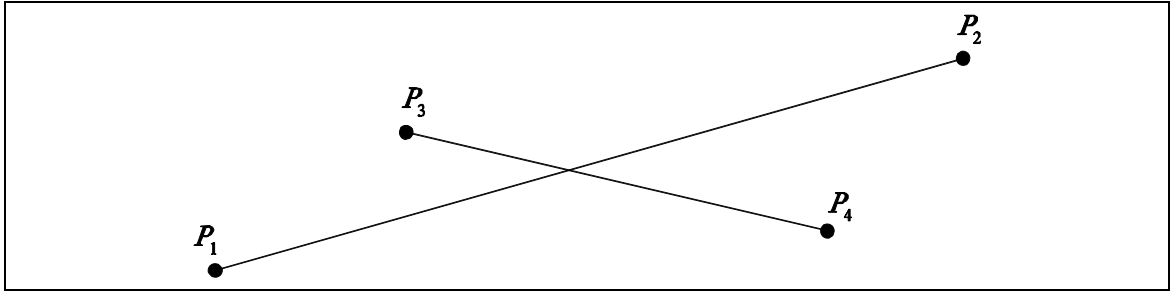


Figure 7.7: Example of intersecting edges.

To actually detect an intersection between two edges, such as those shown in Figure 7.7, we use the line clipping method in [8] in which simultaneous equations are solved. Both of the edges can be described as directed line segments in parametric form by

$$P_1P_2: \quad x = x_1 + t_1(x_2 - x_1), \quad y = y_1 + t_1(y_2 - y_1) \quad (7.3)$$

$$P_3P_4: \quad x = x_3 + t_2(x_4 - x_3), \quad y = y_3 + t_2(y_4 - y_3). \quad (7.4)$$

These equations describe any point (x, y) along the directed line segment by the parameter t in the range $[0, 1]$. We detect an intersection between our two edges by setting their line equations equal to each other and solving for t_1 and t_2 :

$$\begin{bmatrix} (x_2 - x_1) & -(x_4 - x_3) \\ (y_2 - y_1) & -(y_4 - y_3) \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix}. \quad (7.5)$$

The parameters are found after row reduction, and are

$$t_1 = \begin{cases} -1, & x_1 = x_2, \quad x_3 = x_4 \\ \frac{x_4 - x_3}{x_2 - x_1} t_2 + \frac{x_3 - x_1}{x_2 - x_1}, & \text{Otherwise,} \end{cases} \quad (7.6)$$

$$t_2 = \begin{cases} -1, & x_1 = x_2, \quad x_3 = x_4 \\ \frac{(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)}{(x_4 - x_3)(y_2 - y_1) - (y_4 - y_3)(x_2 - x_1)}, & \text{Otherwise.} \end{cases} \quad (7.7)$$

If t_1 and t_2 lie in $[0, 1]$ then the two edges intersect. These equations yield -1 when the two edges are parallel, thus avoiding a divide by zero in the denominator of t_2 . Note that the two edges should be swapped if P_1P_2 is vertical, so as to avoid a divide by zero in the denominator of t_1 .

During complex type D intersections, where several edges have penetrated through other edges, an intersection will commonly be detected *inside* the complex intersection as the penetrating edges attempt to return to the “other side.” To counteract this, a hierarchical check is performed prior to the calculation and distribution of forces, removing all intersections that lie completely within a complex intersection.

7.4.2 Calculation of Penalty Forces

When one group of edges intersects another group, we exert an equal and opposite penalty force upon the two groups, satisfying the principle of virtual work. The force is then apportioned within each group relative to the number of vertices in the group. We define the magnitude of the force as

$$F = K_p \tau \dot{\mathbf{x}}_{avg}, \quad (7.8)$$

where K_p is a user-defined constant, τ is the duration of time this intersection has been active, and $\dot{\mathbf{x}}_{avg}$ is the average velocity of the groups' vertices at the moment of the initial intersection. The velocity term is used as a weighting factor, rightfully applying more countering force in intersections with faster moving edges. This force grows linearly with time until the intersection is resolved.

In the type A intersection example in Figure 7.8a, we have a group consisting of the edge W_1 and a group consisting of the edges W_2 and W_3 . The endpoints of W_1 both receive half of the perpendicularly applied force whereas, in the second group, it is advantageous to apply more repulsion force upon the penetrating vertex than other two endpoints.

Similarly, in the type B example in Figure 7.8b, the endpoints of W_1 each receive half of their group force, which is projected perpendicular to W_1 . The other group, however, gives a total of one-quarter of the force to the endpoints and then divides the remainder among the penetrating vertices. In this case, each of the three penetrating vertices receives one-third of the remaining three-fourths of the force. The drawback to this method of force distribution, however, is that it can be biased, exhibiting more force upon the endpoints of W_1 than upon the penetrating vertices.

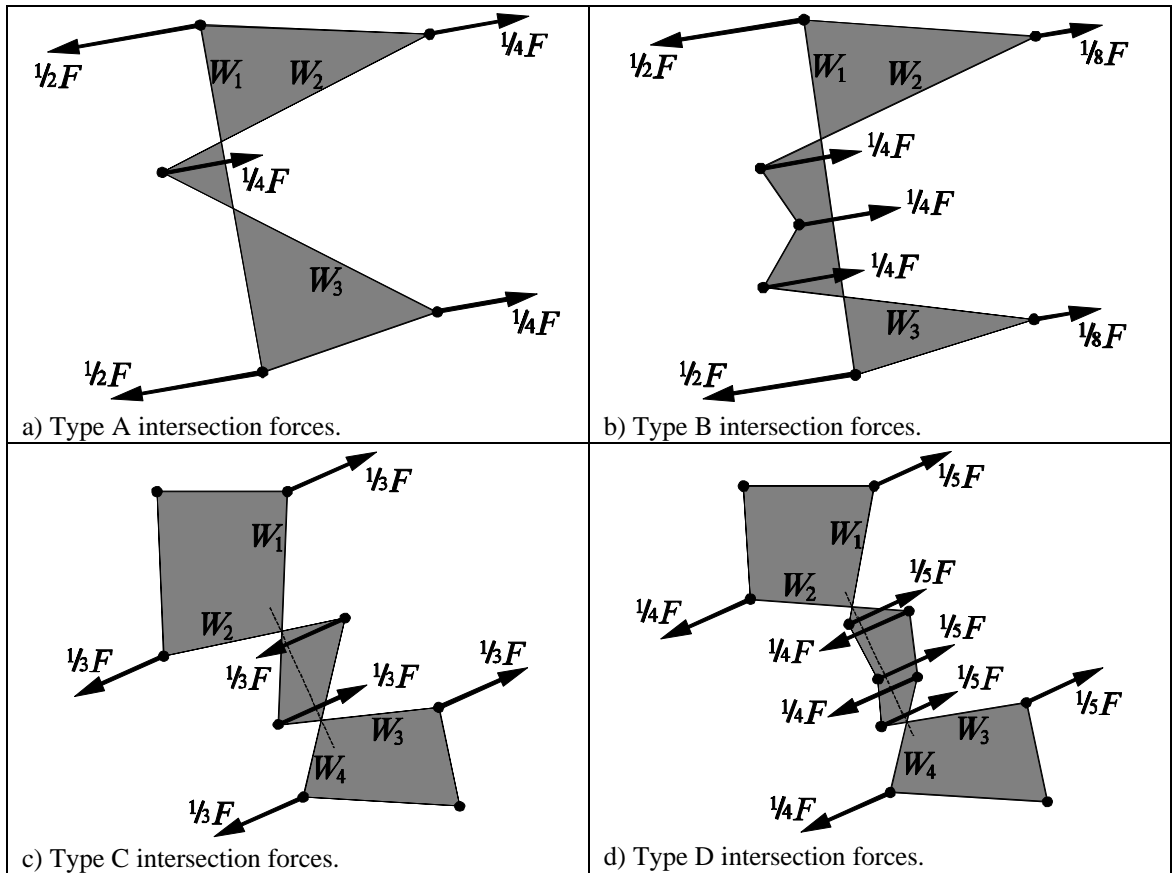


Figure 7.8: Examples of intersection penalty forces.

In the type C and D examples in Figure 7.8c and d, the group forces are simply divided evenly among the group vertices. The forces are projected in a direction perpendicular to a line connecting the two intersection crossing points. These points are found by substituting the known values of t_1 and t_2 from equations 7.6 and 7.7 into the parametric line equations 7.3 and 7.4. However, determining the correct direction of each group's penalty force is not trivial.

To determine force direction, we define a vector \mathbf{v}_1 that connects the intersection point of W_1 and W_2 with the intersection points of W_3 and W_4 . We also define a vector \mathbf{v}_2 , as shown in Figure 7.9a, that connects the first intersection point with the origin of W_2 (recall that vertices are ordered clockwise). If the cross product of \mathbf{v}_1 with \mathbf{v}_2 is negative, then the force direction for the first group (W_1 and W_3) is defined as a 90° clockwise rotation of \mathbf{v}_1 , with the second group's force exactly opposite. A positive cross product

defines the first group force direction as a -90° rotation of \mathbf{v}_1 . While sufficient for our examples in Figures 7.9a and b, the resulting force directions need to be reversed in cases where the second group forms an interior concave angle, as in the Figure 7.9c example.

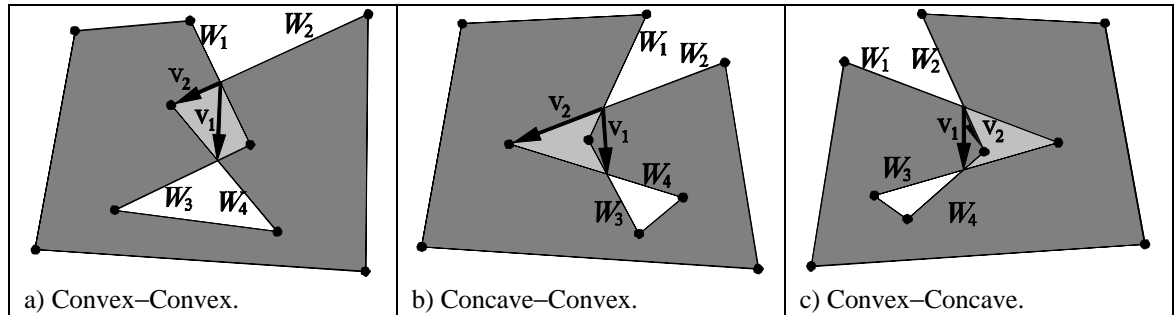


Figure 7.9: Examples of three complex intersections.

Once the force magnitudes and directions have been computed, the forces continue to grow linearly over time until the intersection is resolved. Note that intersection detection and correction is performed also on boundary regions that contain a clockwise ordering of the perimeter edges.

7.4.3 Correcting Reconstructed Hinge Inversions

Self-intersection of regions is also detected and corrected after reconstruction of the map to full resolution. This is necessary because hinges can invert as a result of the reconstruction process. The initial simplification of Pennsylvania is shown in Figure 7.10a. If Pennsylvania's most northern vertex was shifted to the northwest during the cartogram creation, the reconstruction would result in the hinge inversion shown in Figure 7.10b.

Due to the complexity of computing true angles, which is addressed in Chapter VIII, simply enabling the hinge will not produce a repulsion force. We detect inverted hinges by testing for an intersection between a few edges preceding and following each hinge. If an intersection is found, the hinge is forced to close or open to a valid state, as demonstrated in the detail in Figure 7.10c. The forces upon the endpoints are directed perpendicular to the hinge edges. An opposite force, calculated as the negated vector

sum the two endpoint forces, is exerted upon the hinge point in order to ensure no force is added to the system.

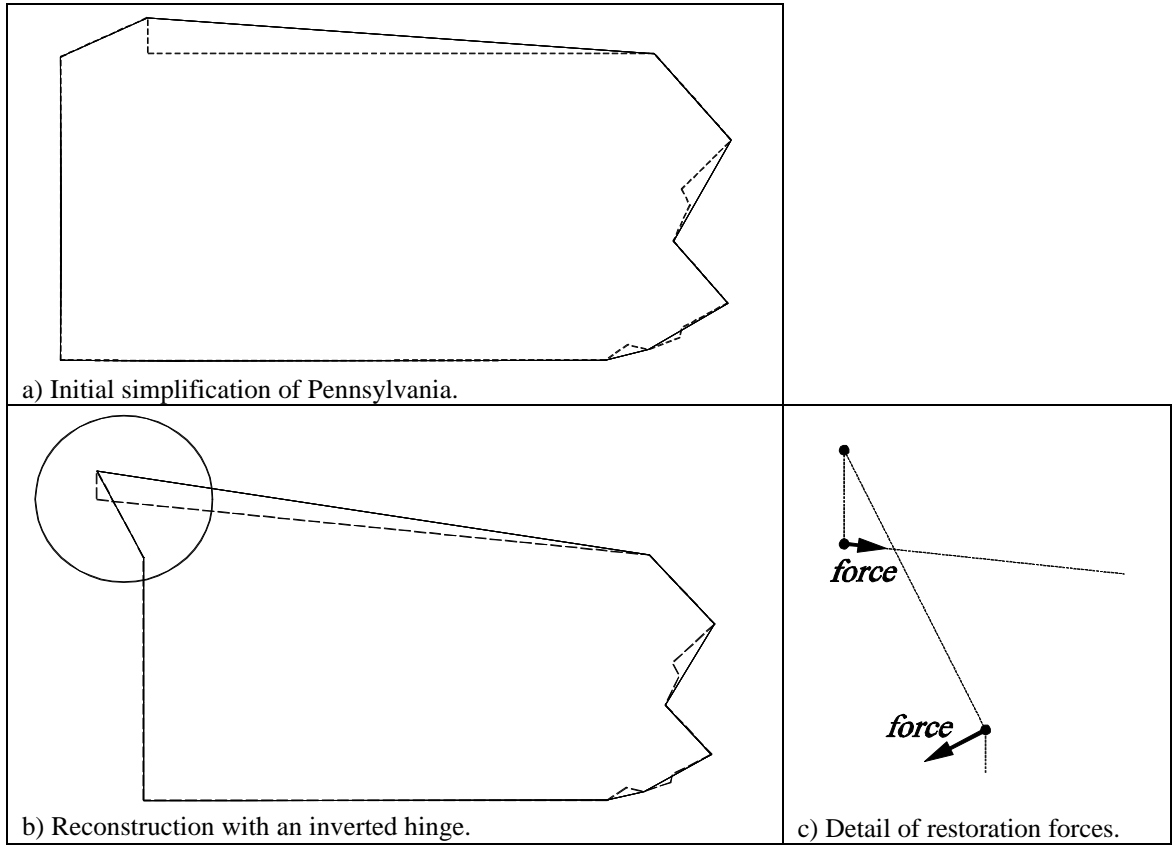


Figure 7.10: Example of a hinge inverted during reconstruction.