Dynamic Choropleth Maps - Using Amalgamation to Increase Area Perceivability:  
SUPPLEMENTARY MATERIAL

Liam McNabb  
Visual & Interactive Computing Group  
Swansea University  
Swansea, Wales  
661370@swansea.ac.uk

Robert S. Laramee  
Visual & Interactive Computing Group  
Swansea University  
Swansea, Wales  
r.s.laramee@swansea.ac.uk

Richard Fry  
National Centre for Population Health and Wellbeing Research, Medical School  
Swansea University  
Swansea, Wales  
r.j.fry@swansea.ac.uk

Algorithm 1 - Are polygons neighbors?

Input–
$p_1$ : polygon one
$p_2$ : polygon two
1: procedure isNeighbor($p_1$, $p_2$)
2:   if isOverlapping( $p_1$.boundingBox(), $p_2$.boundingBox() ) then
3:     return commonVertices($p_1$, $p_2$)
4:   endif
5: return FALSE

Local Variables–
counter : number of matching vertices
MIN = 2 : minimum number of matching vertices required to be neighbors
1: procedure commonVertices($p_1$, $p_2$)
2:   counter = 0
3:   for $i = 0$; $i < p_1$.length(); $i++$
4:     if $p_2$.intersects($p_1[i]$) then
5:       counter++
6:     endIf
7:   endFor
8:   if counter ≥ MIN then
9:     return TRUE
10:   endIf
11: endIf
12: endFor
13: return FALSE

Desc: Compares two polygons and tests for overlapping boundaries. $p_1 \cap p_2$. Returns true if the minimum number of common vertices are found.

Algorithm 2 Contiguous Regions

Input– $L_p$ : non-empty list of polygons
Output– $L_{islands}$ : list of contiguous islands (or land masses)
Local Variables–
island : current island  
neighborFound : flag designating if neighbor is part of existing island
1: procedure identifyContiguousRegions($L_p$)
2:   // For each polygon
3:   while !$L_p$.isEmpty() do
4:     // Assume Island
5:     island = $L_p$.popFirst()
6:     // For each island
7:     for $i = 0$; $j < L_{islands}.length(); i++$
8:       // For each polygon on each island
9:       for $j = 0$; $j < L_{islands}[i].length(); j++$
10:         if isNeighbor(island, $L_{islands}[i][j]$) then
11:           neighborFound = true
12:           break
13:         endIf
14:       endFor
15:     if neighborFound then
16:       island.appendList($L_{islands}[i]$)
17:     $L_{islands}$.removeIslandAt(i)
18:     i--
19:   endIf
20: endFor
21: $L_{islands}$.append(island)
22: endwhile
23: return $L_{islands}$

Desc: Partitions a list of non-contiguous polygons into separate contiguous regions such as islands and land masses. A contiguous region has connected neighbors where no area is completely separated by water.
Algorithm 3 Identify Boundary Range

Input:
start : starting index of shared boundary line
end : last index of shared boundary line
Vc : current polygon vertices in clockwise order
Vn : neighbor polygon vertices in clockwise order

Local Variables:
longestC : found by comparing distance between common vertices
common : list of found commonVertices

1: procedure IDENTIFYBOUNDARYRANGE(start, end, Vc, Vn)
2: for int i = 0; i < Vc.length(); ++i do
3:    if Vn.contains(Vc[i]) then
4:        common.append(Vc[i])
5:    endIf
6: endFor (i)
7: longestC = longestSharedBoundaryChain(common, Vc)
8: *end = Vc.indexOf(common[longestC.next()])
9: *start = Vc.indexOf(common[longestC])
10: return

Desc: Identifies b for Vc with Vn as a neighbor. Required for parent node.

Algorithm 4 Longest Shared Boundary Chain

Input:
common : list of found commonVertices
Vc : current polygon vertices in clockwise order

1: procedure LONGESTSHAREDBOUNDARYCHAIN( common, Vc )
2:    if isLongestChain(Vc, &longest, Vc.indexOf(common.longest(last())), Vc.indexOf(common.first(i))) then
3:        longestIndex = common.length()-1
4:    endIf
5:    for i = 1; i < common.length(); ++i do
6:        if isLongestChain(Vc, &longest, Vc.indexOf(common(i)), Vc.indexOf(common(i-1))) then
7:            longestIndex = i
8:        endIf
9:    endFor (i)
10: return longestIndex

New Input:
longestL : The current longest distance between two common vertices
currI : current index to test

Local Variables:
length : length of current chain

1: procedure ISLONGESTCHAIN(Vc, longestL, currI, nextI)
2:    length = nextI - currI
3:    if length < 0 then
4:        length = length + current.size() - 1
5:    endIf
6:    if *longestL < length then
7:        *longestL = length
8:        return true
9:    else
10:       return false
11: endIf

Desc: Identifies the longest absence of a common vertex. We can assume that this signifies the beginning and end points of b.

Algorithm 5 Build Binary Tree

Input:
Lcontig : contiguous list of polygon sorted by area
neighborI : index of selected neighbor
p : parent node of two neighbor areas

1: procedure BUILDINARYTREE(Lcontig)
2:    if Lcontig.length() > 1 then
3:        //Neighbor Selection
4:        neighborI = selectNeighbor(list) //neighbor of list.first()
5:        //Create Parent Area
6:        p = new Node()
7:        p.identifyVertices(Lcontig.first(), Lcontig.at(neighborI))
8:        p.setLeftChild(Lcontig.first())
9:        p.setRightChild(Lcontig.at(neighborI))
10: Lcontig.first().setParent(p)
11: Lcontig.at(neighborI).setParent(p)
12: //Update Sorted List with Parent
13:    updateList(Lcontig, p, i)
14: return buildBinaryTree(Lcontig)
15: endIf
16: // Base Case - Lcontig == 1
17: return Lcontig

Desc: Builds hierarchy of polygons using a list of merge candidates recursively.
Fig. 1: An example of zooming out of Italy’s administrative units where $m = 1\%$.

Fig. 2: Example of the contiguous regions procedure applied to the Ahupua’a boundaries of the state of Hawaii [2]. There are 10 visible contiguous islands each with their own color.

Fig. 3: Example of derived parent area’s from original unit-areas. The example uses the State of Texas and shows multiple boundary merges [3] so that $m$ is 15%. Our example uses a color palette from colorbrewer [1].

REFERENCES

