

Angus Forbes

UC Santa Barbara  
Media Arts & Technology  
angus.forbes@mat.ucsb.edu

## Coil Maps : Animated Deformation of Geographic Nodes

### Introduction

A Coil Map is an animated geographic representation which dynamically transforms nodes of information in response to input data. At the same time, the transformation maintains each node in loose proximity to a recognizable geographical position. Coil Map uses a visualization technique similar to the TreeMap visualization as originally developed by Ben Scheiderman. The Coil Map is placed over geographic image, and is initially divided uniformly into a number of cells. The Coil Map listens for events that occur within the geographic regions as defined by those cells. After each event occurs, the map is transformed. The event causes a chain of shifts in various nodes as it "uncoils"-- propagates the new information upwards-- recursively upward to from the leaf node (where the event occurred) to the root node. The effect of this series of transformations is that the more dramatic changes occurring near the area where the event occurred are mitigated as it propagates to more distant neighbors.

### Initialization of Coil Map

The Coil Map is initialized in the following way: 1) A map is chosen; 2) A set of nodes are defined where each node is recursively binarily subdivided, alternately horizontally and vertically, up to a maximum depth; 3) Colors are assigned to each of the nodes based on the original map image; 4) Each leaf node in the Coil Map is assigned the

same initial value; 5) The initial values are recursively propagated upwards to the root node by summing up the values of its children and setting the children's parent to that summed value.

For instance, assume we have a simple Coil Map with a depth of 4 levels. Each leaf node, at level 4, is initialized with a value of 1. The parent of the leaf node, at level 3, sums up its children and now has a value of 2. The parent of that parent node, at level 2, sums up its children and now has a value of 4. Its parent then sums up the two children and now has a value of 8. The root node sums up its 2 children and thus has a value of 16. There will be  $2^{\text{depth}}$  nodes, and thus each node at each level will be initialized to  $2^{(\text{maxDepth}-\text{level})}$ .

### Propagation of events

Once the Coil Map is initialized we then listen for events. When an event is received in the geographic region defined by a particular leaf node, the value of that node is incremented by 1. We recursively propagate this value upwards to the root node as in initialization step 5 above. We then draw the leaf nodes based upon the values in the nodes in the following manner: At each level, from the root node down to the parents of the leaf nodes, we take the sum of the values of both children and position each child them within their parent's space based on what percentage of the total value it has. This is done until we reach the parent of the leaf node. At this point we draw a line to indicate the separation between the leaf nodes. Since the leaf nodes cover the entire map, and since each node visualizes a portion of the map, the net effect is a slow transformation of the map as the lines separating the leaf nodes shift based upon the events received. Since a change in value at one of the leaf nodes will cause the balance between it and its other side to change much more than at the higher level  $[(1+1) / 2]$  versus  $[(2^{(\text{maxDepth}-1)} + 1) / 2^{(\text{maxDepth}-1)}]$  the net effect of the recalculation is to cause a greater shift locally and only a minor change globally.

For instance, assume we have our simple Coil Map as initialized above and receive a single event in the top-leftmost leaf node. The value of the leaf node will increase by one, and this increment will be propagated recursively through to the root node. The position of the line between the two leaf nodes will be a pixel value based on the 2:1 ratio as indicated by the values in the nodes. That is, the line will move from being centered 50% within the bounds of the parent node to 66.67%. Likewise, at level 2, the line between the nodes parent will be a pixel value based on the 3:2 ratio (60%); at level 1, the line will be based on a 5:4 ratio (.55%); at the top level, the line will be a 9:8 ratio (.53%).

### Animation details

In addition to the base algorithm, a number of different parameters can be applied to the Coil Map which control the animation. First, the increment value when an event occurs can be changed. If the default value of an event is changed to something larger, then the initial change to the map is far more dramatic. This can be useful if there are many events spread more-or-less evenly throughout the map-- the large "shock" to the map will be initially very noticeable when it first appears before returning to a more uniform position. Second, we can have the value automatically return to its initial value so that the deformation of the map is only temporary. Third, the speed of the change can be altered to be more or less jarring. If the speed is slow enough that many events will occur before any of the animations are complete the effect is a "rippling" as the transformations occur.

### The algorithm in detail

1. A map image is chosen.
2. A set of nodes are defined the map in the following recursive manner:

```

Initialize:
    max = getMaxDepth();
    depth = 1;
    orientation = VRT;
    leaf = new ListOfNodes();
    parent = createNodeFromMapImage(map);
    topHalf = createSide1Node(node, VRT);
    bottomHalf = createSide2Node(node, VRT);
    subdivide(parent, topHalf, depth+1, max, HRZ);
    subdivide(parent, bottomHalf, depth+1, max, HRZ);

subdivide(parent, node, depth, orientation)
{
    if (depth == max)
        leaf.add(parent);
        return;

    else if (orientation == VRT)
        //cut area in half vertically
        topHalf = createSide1Node(node, VRT);
        bottomHalf = createSide2Node(node, VRT);
        subdivide(parent, topHalf, depth+1, max, HRZ);
        subdivide(parent, bottomHalf, depth+1, max, HRZ);

    else if (orientation == HRZ)
        //cut area in half horizontally
        rightHalf = createSide1Node(node, HRZ);
        leftHalf = createSide2Node(node, HRZ);
        subdivide(parent, rightHalf, depth+1, max, VRT);
        subdivide(parent, leftHalf, depth+1, max, VRT);

    parent.add(node);
}

```

3. We now have a grid of uniformly sized leaf-nodes representing a rectangular area which completely covers the map image. Assuming we have a simple two colored map where color1 = land and color2 = water, we assign a color to each of the nodes based upon whether there are pixels of color1 or color2 in the area represented by the node.

4. We then assign each of the leaf-nodes nodes a value of 1, and then recurse upwards to the parent node, each time assigning the parent node a value of the sum of its children. So each leaf node would have a value of 1, the parent-node of the leaf nodes would have a value of 2, that parent-node's parent would have a value of 4, and so on up until

we reach the parent, which would be initialized with a value of  $2^{\text{maxDepth}}$  (which is the number of total leaf nodes).

5. Whenever an event occurs within a geographic region defined by one of the leaf nodes we increment the value of that leaf node by 1, which in turn increases the value of its parent by 1, which in turn increases the value of its parent by 1, and so until we reach the parent.

6. As the values for each node change, we recalculate the position of the line separating side1 and side2. Since a change in value at one of the leaf nodes will cause the balance between it and its other side to change much more than at the higher level  $[(1+1) / 2]$  versus  $[(2^{(\text{maxDepth}-1)} + 1) / 2^{(\text{maxDepth}-1)}]$  the net effect of the recalculation is to cause a greater shift locally and only a minor change globally.