The Use of Change Identifiers to Update Footprints of Dot Patterns in Real Time

Maximillian Dupenois and **Antony Galton**[1]

Abstract. This paper looks at the problems involved in maintaining footprints over non-static dot patterns and how to balance efficiency and accuracy. It introduces the concept of change identifiers as a method of choosing appropriate update points.

1 Introduction

In spatial information theory one often encounters the problem of representing groups or aggregates, which at a fine level of granularity appear as pluralities with a scattered distribution but at a coarser granularity may be treated as single coherent individuals with their own behaviour and properties. Familiar examples from the everyday world include forests (i.e., aggregates of trees), flocks and crowds (aggregates of animals or people), and conurbations (aggregates of buildings).

In recent research, attention has been paid to the problem of assigning a spatial location to an aggregate considered as a unit, given as inputs the spatial locations of the individual components [3, 7]. In abstract form, the two-dimensional problem is, given a set of *dots* (i.e., objects sufficiently compact to be idealised as points) in the plane, to determine a *footprint* representative of the spatial distribution of the collection of dots taken as a whole. The footprint will be a two-dimensional region, which, depending on one's purposes, may be required to satisfy various contraints such as polygonality, connectedness, topological regularity, convexity, etc [4]. The problem generalises to three dimensions in the obvious way, but for simplicity the discussion in this paper will be restricted to the two-dimensional version.

It cannot be too strongly emphasised that there does not exist a uniquely "best" or "correct" footprint for a given dot pattern. In [6] it was shown experimentally that footprints selected as "good" by human subjects represent optimal trade-offs between the conflicting goals of minimising the area and minimising the perimeter, but this certainly does not tell the whole story. For human purposes, an important feature of a good footprint is that it "looks" right, that is, it represents a shape that we "see" in the dot pattern itself. But for some purposes, one might prefer to use a footprint that is very easily computed (e.g., the minimal axis-aligned bounding rectangle) or which has well-known mathematical properties (e.g., the convex hull), even though in many cases these do not provide a close visual match to the dot pattern.

Many different algorithms for generating footprints from dot sets have been proposed, in contexts such as geographical information theory [1, 7], pattern recognition [10, 3], computer vision [9], and computational geometry [5], to cite only a few representative examples. In all these cases, however, the assumption is that the dot patterns are *static*. In reality, many examples of collectives or aggregates are dynamic, with either the location or the membership, or both, varying over time [13]. Of our examples above, flocks and crowds vary in both these respects over a short time scale; forests and conurbations also vary, but the time-scale is typically several orders of magnitude greater.

The problem we address in this paper is how to track the footprint of dynamically changing aggregates of dots. In the case of fast-moving aggregates, an added constraint is that the tracking should take place in real time. Footprint algorithms typically run in time $O(n \log n)$ or worse (sometimes much worse), where n is the number of dots. Hence recomputing the footprint *ab initio* every time there is a change in the dot pattern will be computationally costly, making real-time recomputation infeasible in many cases.

One possible approach would be to look for a way to *update* the footprint incrementally rather than recompute it entirely. In an ideal world, one could do this in such a way that the footprint assigned to the dots at any time is always identical to the footprint that would be obtained if it were recomputed. In general, for most types of footprint it is unlikely that such exact tracking can be accomplished with significantly less cost than recomputing the footprint every time.

Instead, we propose a method by which the position of the dots in relation to the most-recently computed footprint is continuously monitored, and the footprint is only recomputed when the mismatch between the dot positions and the current footprint exceeds some preassigned threshold of accuracy. Clearly there will be a trade-off between the level at which the accuracy threshold is set and the resultant frequency of recomputation, and we investigate the nature of this trade-off with a view to optimising it.

Our approach is to use a suite of easily computable *change identifiers*, each with its own threshold. Recomputation of the footprint is triggered when some aggregate value computed from the values returned by the change identifiers exceed a given threshold. In the simplest form this aggregate value could be a count of how many of the change identifiers individually exceed their thresholds, amounting in effect to a vote amongst the change identifiers. Alternatively, the change identifiers could be ranked in order of importance and a weighted combination of their values compared with some threshold. We investigate the effect of using different sets of change identifiers, and different ways of combining the results returned by them.

The plan for the remainder of this paper is as follows. In section 2 we discuss a range of possible change identifiers, evaluating them in terms of their ease of computation and informativeness in relation to the task in hand. In section 3 we consider combinations of change identifiers, and discuss the computation of aggregate values

[1] University of Exeter, England, email: m.p.dupenois@ex.ac.uk, a.p.galton@ex.ac.uk

and thresholds. In section 4, we describe the software engine we have used to implement the ideas previously described, and in section 5 we outline the protocol for using this engine to investigate different dynamic dot patterns, footprint algorithms, and change-identifier sets. Section 6 presents preliminary tests which have been conducted using the engine, using a convex hull algorithm for generating footprints. Finally, in section 7 we summarise the results obtained so far and outline our plans for future work.

2 Process

The basic process we implement is shown as Algorithm 1, which works as follows. The incoming data consists of a sequence of dot patterns (which might come from, e.g., observations relayed by sensor arrays). The dot pattern associated with time step i is denoted DP_i, and is referred to as the **current dot pattern** when i is the current time. An algorithm for generating footprints from dot patterns is assumed given (we shall refer to this as the **footprint algorithm**), and at the beginning of the sequence a footprint $f(DP_0)$ is generated for dot pattern DP_0 and saved as the **stored footprint** SFP_0. The dot pattern DP_0 from which it is generated is stored as the **stored dot pattern** (SDP_0). At subsequent time steps, the change identifiers are used to determine whether a new footprint should be computed; this is done by evaluating the extent to which the current dot pattern DP_i differs from the previously stored dot pattern SDP_{i-1}. If this value, $eval(DP_i, SDP_{i-1}, SFP_{i-1})$ exceeds some pre-set threshold, then a new footprint, $f(DP_i)$ is generated as the new stored footprint SFP_i, and the current dot pattern is used as the new stored dot pattern DP_i. Otherwise, the stored dot pattern and footprint are retained from the previous time step.

Algorithm 1 Main Process

$i = 0$
Input first dot pattern DP_0
$SFP_0 = f(DP_0)$
$SDP_0 = DP_0$
repeat
    $i = i + 1$
    Input DP_i
    if $eval(DP_i, SDP_{i-1}, SFP_{i-1}) > threshold$ **then**
        $SFP_i = f(DP_i)$
        $SDP_i = DP_i$
    else
        $SDP_i = SDP_{i-1}$
        $SFP_i = SFP_{i-1}$
    end if
until No more input available

3 Change Identifiers

Each change identifier returns a value representing some measure of change. To produce this value it has access to the stored dot pattern, the current dot pattern and the stored footprint. Most of the identifiers listed below do not use the stored footprint; this enables them to be used in conjunction with a wide range of footprint algorithms, since they make no assumptions concerning the nature of the footprint (e.g., whether it must be polygonal, can have holes or multiple components, etc.). To assess whether the value it returns should force a footprint update, a threshold is associated with each change identifier; and if change identifiers are to be combined, a method to normalise their values is required. These ideas are discussed in the section on change identifier sets §4. We term a change indentifier breaking its threshold as a fail as the stored footprint has failed to accurately represent the current dot patten.

The identifiers listed below are by no means an exhaustive list but they do present a solid base, covering a range of possible transformation types the dot pattern could undergo, e.g., changes in position, distribution and membership of the dot pattern.

3.1 Change in centroid scaled by the bounding box

This change value is given by the distance between the centroids of the current dot pattern and the stored dot pattern. The value is normalised by dividing it by the diagonal of the bounding box of the stored dot pattern. If the dot pattern has n dots, the total computation time is $O(n)$ (If the dots are held in a suitable tree data structure, the bounding boxes can be found in time $O(\log n)$, but this does not reduce the overall order-of-magnitude complexity.)

3.2 Change in variance from the centroid

The difference between the variances[2] of the current and stored dot patterns. We use variance rather than standard deviation so as to avoid the processing time involved in computing the square root. This measure can also be computed in time $O(n)$.

3.3 Change in axis-aligned medians

This is given by the distance between the 'medians' of the current and stored dot patterns, where the (axis-aligned) median of a dot pattern is defined as the point whose coordinates are the medians of the x-coordinates of the dots and the y-coordinates of the dots respectively. This is analogous to the centroid but computed using the median rather than the mean. However, unlike the centroid, it is not rotation-invariant.

3.4 Percentage change in number of dots

This is the difference in number of dots between the current dot pattern and the stored dot pattern as a percentage of the number of dots in the stored dot pattern. This can be computed in $O(n + i)$ time, where i is the number of dots from the previous pattern.

3.5 Change in bounding box

This is computed as the area of the symmetric difference between the bounding boxes of the current and stored dot patterns, expressed as a fraction of the area of the bounding box of the stored dot pattern. This is the sum of the areas of the bounding boxes, less twice the area of their intersection. If the dots are held in a two-dimensional tree data structure, this can be computed in time $O(\log n)$.

3.6 Proportion of points outside the boundary of the stored footprint

The fraction of dots outside the current footprint. By using the ray-casting method [12] we can find this in $O(nm)$ time, where m is the number of edges of the footprint. This is only sensibly applied if the footprint algorithm does not allow outliers, i.e. dots present in the dot

[2] The variance is the mean squared distance of the dots from the centroid.

pattern but not in the completed footprint. It should be noted that this is the only change identifier on our list which makes use of the stored footprint.

4 Change Identifier Sets

While the identifiers could all be run separately, it seems likely that the best results will be achieved when a group of two or more identifiers is used. To this end we use an xml file to collect them. Listing 1 shows an example of this.

```
<changeidentifierset name="[set-name]" ver="1.0">
  <description>Description of the set</description>
  <threshold>50</threshold>
  <maxFails>2</maxFails>
  <concurrent>false</concurrent>
  <changeidentifier>
    <location>[location]</location>
    <classname>[classname]</classname>
    <priority>1</priority>
    <threshold>10</threshold>
    <multiplier>1</multiplier>
    <redrawOnFail>false</redrawOnFail>
  </changeidentifier>
  <changeidentifier>
    <location>[location]</location>
    <classname>[classname]</classname>
    <priority>2</priority>
    <threshold>10</threshold>
    <multiplier>1</multiplier>
    <redrawOnFail>false</redrawOnFail>
  </changeidentifier>
</changeidentifierset>
```

Listing 1. Change Indentifier Set

A quick description of some of the elements will explain how we have dealt with the parameters of the identifiers. Looking first at the global elements there are two values that affect the updates directly: <threshold> and <maxFails>. If the total change is over the threshold value or if the number of change identifiers that fail is over the the maximum allowed fails then we redraw the footprint. <concurrent> concerns the way the identifiers run, each identifier has a priority, if <concurrent> is false then identifiers are run in the order of descending priority. The lower the number the higher the priority. If <concurrent> is true then the change identifiers are threaded, the reason this option is in place is that over small test sets the amount of time taken to start a thread could be above the time taken to run the footprint algorithm. Each change identifier is considered to have failed if the change it returns is over the threshold. If <redrawOnFail > is set to true then, regardless of the other change identifiers, we force a footprint redraw. The last element of particular interest is the < multiplier>. The identifiers have different scales of measurement, adding the area change to the variance from the centroid is combining two very different units together and therefore may give undue importance to one identifier over another. The multiplier is applied to the change value of the identifier before it is added to the total value to prevent this inequality from occurring.

With regard to finding appropriate multipliers, thresholds and change identifier combinations there are too many variations to work through and test individually. To that end future work includes running an optimisation system of some kind to find appropriate sets and values.

5 Analysis

The purpose of using change identifiers is to enable the evolution of a footprint to be tracked more efficiently than by recomputing the footprint at each time step. The footprint is only recomputed when the change identifiers indicate that the dot pattern has changed sufficiently to make the mismatch with the current stored footprint unacceptably great. The number of footprint recomputations, and hence the total time taken to process a given sequence of dot patterns, will depend on the change identifiers used, and the threshold settings. We define variables as follows:

- $t_{FP}(i)$ is the time taken to compute the footprint from the dot pattern at step i.
- $t_{CI}(i)$ is the time taken to evaluate the change identifiers at step i.
- $r(i)$ is a Boolean variable, set to 1 if the footprint is in fact recomputed at step i, and zero otherwise.

The total computation time over a run of n dot patterns is thus

$$T_{CI} = t_{FP}(0) + \sum_{i=1}^{n}(t_{CI}(i) + r(i)t_{FP}(i)).$$

The value of T is minimum when the change identifier threshold is set so high that the footprint is never recomputed after the start of the sequence (so $r(i) = 0$ for $0 < i \leq n$):

$$T_{\min} = t_{FP}(0) + \sum_{i=1}^{n} t_{CI}(i).$$

It is maximum when the change identifier threshold is set so low that the footprint is recomputed at every time step (so $r(i) = 1$ for all i):

$$T_{\max} = t_{FP}(0) + \sum_{i=1}^{n}(t_{CI}(i) + t_{FP}(i)).$$

If change identifiers are not used at all, and the footprint recomputed at every time step, then the total time taken is

$$T_{NCI} = \sum_{i=0}^{n} t_{FP}(i) = t_{FP}(0) + T_{\max} - T_{\min}.$$

If it is assumed that always $t_{CI}(i) < t_{FP}(i)$ (for if not, there would be little point in using change identifiers) then $T_{\min} < T_{NCI} < T_{\max}$, so the relative size of T_{CI} and T_{NCI} — which provides a measure of the time advantage, if any, gained by using change identifiers — depends on the threshold settings.

This time advantage must be set against the accuracy with which the footprint is tracked. The cost of using change identifiers comes from the fact that, most of the time, the stored footprint differs from the true footprint. To measure this cost, we need a way of quantifying the extent of this mismatch. The difference between two footprints can be measured in various ways, e.g., using Hausdorff distance, or symmetric area difference (see [8, Ch. 7] for a discussion). Here we will use only the symmetric area difference, but the principles described below would apply equally to other measures.

The symmetric difference between two regions comprises the parts of each region that do not overlap the other; it is given by

$$R_1 \Delta R_2 = (R_1 \setminus R_2) \cup (R_2 \setminus R_1) = (R_1 \cup R_2) \setminus (R_1 \cap R_2).$$

We use the area of this as a measure of the dissimilarity between two footprints; and since we are only interested in comparisons, not absolute values, we normalise this area by expressing it as a fraction of the area of the true footprint. Thus the aggregate mismatch between the stored footprint and the true footprint over a dot-pattern sequence of length n is given by

$$mismatch = \sum_{i=0}^{n} \frac{||FP_i \Delta SFP_i||}{||FP_i||},$$

where $||X||$ denotes the area of region X.

If the footprint is recomputed every time, corresponding to total computation time $T_{\max}$, we have $SFP_i = FP_i$ for every i, so $mismatch = 0$. At the other extreme, the maximum value of $mismatch$ is obtained when the footprint is never recomputed, corresponding to $T_{\min}$. There is thus a trade-off between accuracy and computation time, as indicated in Figure 1, where different choices of change identifier thresholds correspond to different positions on the curve. The optimal setting for the change identifier threshold depends on the relative importance attached to the conflicting goals of minimizing both computation time and accumulated footprint error; but in any case no time advantage can be obtained for mismatches below the value m at which $T_{CI} = T_{NCI}$.

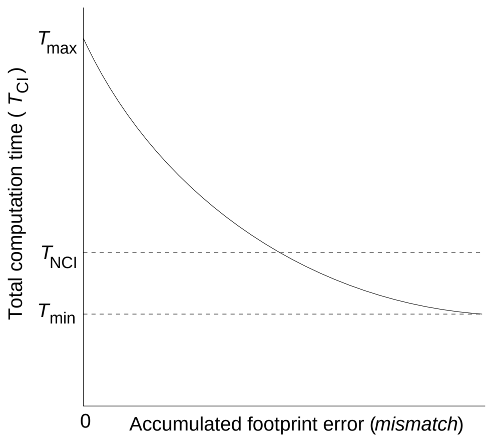


Figure 1. Total computation time against aggregate footprint error

6 Implementation

6.1 System

We have implemented a system to run test the ideas presented in this paper. The system is split into modular parts: the engine, the change identifiers, the application and the footprint algorithms. The application initialises an instance of the engine, passing it the footprint algorithm to use and the indentifier set to process, then it starts the instance. The engine sits in a waiting state checking an internal queue to see if it has dot patterns to process. The application passes dot patterns to the engine, as in a live system it does not wait for a response but sends them continuously. The engine processes the dot patterns and notifies the application each time it generates a new stored footprint. Once the application has sent all the patterns to the engine it sends a command to stop. If we are running this instance as a test then the engine processes the dot patterns without the change identifiers and records the data.

The information we record is listed below:

- Time taken to run the engine over the entire set.
- Time taken to run the footprint algorithm.
- The state of the change identifier set at each timestep:
  - How long each change identifier took to run.
  - Which change identifiers failed.
  - The value each change identifier returned.
  - What the total change was.
  - If the change identifier set redrew then which change identifier was the one that caused the set to fail.
- Time taken to run the set.
- The current dot pattern at each time step.
- The stored footprint at each time step.
- The 'true' footprint at each time step.

The other component of note in the system is a properties holder linked to the dot patterns. The change identifiers often compare values from the current and last updated from dot patterns. Once this value has been found it is inefficient to work it out again so the pattern stores it in a mapping table.

6.2 Testing Methodology

As mentioned in §6.1 we also run the set again without change identifiers. By running this control we can see how much time is saved using the change identifier sets and draw similarity comparisons, giving us quantative data to see how far out of step the stored footprint at any time step is from the true footprint. We use the methods described in §5 to produce two graphs: the first being the symmetric area difference against time step and the second being the time taken for each time step. The uses of the symmetric area difference have already been discussed, so we shall just note that the time taken graph is important for more than the the total time of the run. If a change identifier set takes nearly as long as the footprint to process and the dot patten stream does not require an update often the total time for the run with change identifiers will still be less than for without, even though it's total update time ($t_{CI}(i) + r(i)t_{FP}(i)$) is nearly double the footprint time ($t_{FP}(i)$). The time taken graph will show this large increase as a tall spike.

Another method of judging footprints is the entirely qualitative approach of whether or not human intuition calls it a good fit. We can record the streams as 'movies' of the footprint evolving with the dot pattern. These 'movies' can be played to a selection of people and they can be asked to rate how well they felt the footprint kept up with the dot pattern. Importantly the test should be set up such that the notion of a good footprint is disentangled from how well it can be tracked. Results from this experiment would indicate just how important people think accuracy is. This data will allow us to state which change identifier sets give acceptable accuracy for high efficiency and may help us say something about what properties of the dot pattern are most important when generating a footprint.

Also of interest will be the comparison between the quantative and the qualitative data. Comparing the sets seen as accurate within the study to the sets given as accurate by the quantative testing may tell us which change identifiers are most important to human intuition.

6.3 Current Results

Current tests have been run on streams of 500 dot patterns of up to 1000 dots. In default of real data we have implemented a collective motion pattern generator which can use different methods to produce streams of dot patterns. The method that generated the patterns for the current tests makes use of the Boid behaviours of seperation, cohesion and aggregation[11] to dictate the movement of the dots. The algorithm used has been the upper and lower convex hull algorithm as given in [2]. A separate program has been written to showcase the two footprints for each timestep (one with change identifiers the other without) and time details from the test (See Figure 2[3]).

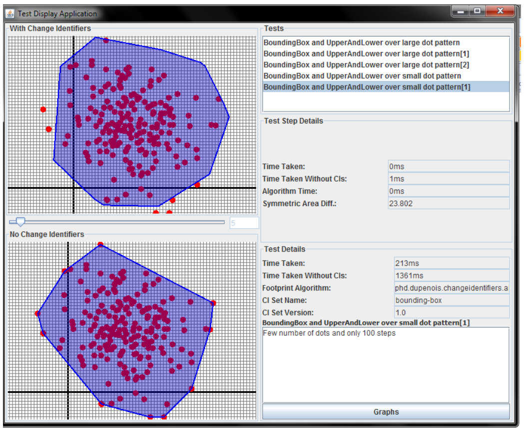
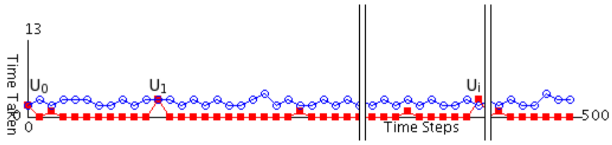


Figure 2. Screenshot of Result Display App.

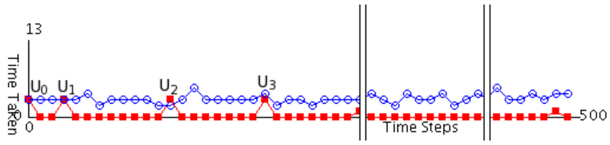
Currently the only two change identifiers that full tests have been run on are the difference in area of the bounding box (§3.5) and the change in the number of dots (§3.4), however both of these consistently show better run times for with change identifiers than without.

The results display application produces the two graphs described in §6.2. The time taken graph (See Figure 3) has two lines: The squares are on the line representing the run with change identifiers, and the circles are on the line representing the run without. The black bars represent where the graph has been cut and stitched, this is because the graph was simply too long at 500 steps to display in its entirety. As would be expected, with change identifiers is consistently below without, in fact it generally takes less than 1ms to run and therefore is less than 1ms over the footprint algorithm time when it updates. The time steps at which it updates can clearly be seen on the graph $U_0 - Ui$. Figure 3(a) is the time step difference when the threshold of the bounding box is set at 20%, Figure 3(b) is when the theshold is 10%. The 10% threshold updates more often and we have a total time of 90ms for the run compared to 61ms for the 20%, both are far below the comparison run which updates each timestep however, that being 1331ms for the 20% run and 1342ms on the 10%.

The area difference graph (See Figure 4) also clearly shows the update times ($U_0 - Ui$). More interesting is the information it can tell us about the change of the dot pattern. The regularity with which these updates occur show us the how static the dot pattern is and, if we know the change identifier(s) used, how it changed. The cropping makes it unclear but on Figure 4(a), towards the end the area difference levels out. This leveling out indicates that the bounding box of the dot pattern did not change by over 20% for these time steps. The area difference at during this static period is around 16%, if this is within allowed footprint error then we are saving large amounts of time across the period by not updating. If, however, 16% is considered too great a footprint difference then we need to change the threshold values on the identifier set to update earlier. Figure 4(b) is a run with the bounding box threshold set at 10% , as mentioned above, this causes far more updates. Interestingly Figure 4(b) does not level out like Figure 4(a) does, this shows that lowering the threshold picked up change ignored by the larger. The accumulated error (as described in §5) for Figure 4(b) and Figure 4(b) is 4545.5 and 2826 respectively, these seem like large numbers but are accumulated over 500 time steps and give us an average error of 9.091 and 5.652. Whether or not this is acceptable will depend on specific application requirements.

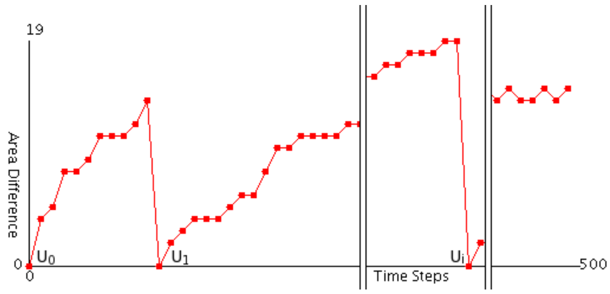


(a) Bounding Box Threshold at 20%

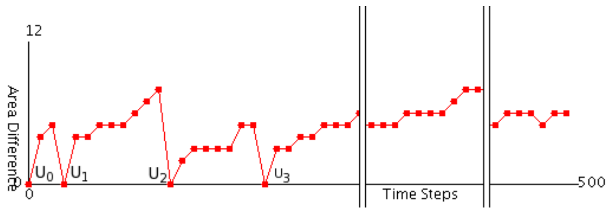


(b) Bounding Box Threshold at 10%

Figure 3. Graph of Time Taken against Time Steps



(a) Bounding Box Threshold at 20%



(b) Bounding Box Threshold at 10%

Figure 4. Graph of Footprint Area Difference against Time Steps

7 Conclusions amd Further Work

The principles behind the change identifiers appear to be sound. The graphs show a consistent saving of 5ms per time step using only the

[3] The screenshot is from a smaller test than the one mentioned above so that the footprints are clearly visible on the small image

bounding box change identifier. There have not yet been enough tests performed to say whether or not any change identifier is better than an other, however the bounding box has shown itself to be able to identify dot pattern changes and update accordingly.

The continuation of this work includes implementing the rest of the change identifiers and running basic tests on them, as with the bounding box, to see if they affect the update times with any regularity. Once done, an application using the principles of optimisation will be created to sort through the variations of change identifier sets over a particular dot pattern stream with a particular footprint algorithm. The results of this will be plotted on to a graph (as described in §6.2) of area under the area difference graph against time taken. This application will need to be run over several footprint algorithms and dot pattern streams. With regard to the different types of footprint algorithm; the χ-hull algorithm from [3], the α-shape from [5] and the swinging-arm algorithm from [7] will be implemented. The majority of non-convex footprint algorithms require some external parameter (α in the α-shape, line length in the χ-hull and arm length in the swinging-arm), fortunately the selection of this parameter does not greatly concern us. We are interested in how well we can track the footprint, not how appropriate the footprint is for the dot pattern.

[13] mentions several collective movement types, having sets of dot pattern streams that replicate these movements would lend weight to the accuracy rating of the change identifiers. It would show that the identifier in question was accurate over all types, accurate only for some or for none.

Other accuracy measures will also be implemented (Hausdorff distance etc.) and it will be interesting to see how they relate to each other. A side interest will be to see how they relate to the accuracy ratings from the human study, it may be that one of the measures is more implicitly used by the human mind than others.

The human study requires some research into the execution as it is all to easy to sway results simply with inaccurate wording on the introduction. With this in mind we will liase with the University of Exeter psychology department in an attempt to avoid accidental bias.

REFERENCES

[1] A. Arampatzis, M. van Kreveld, I. Reinbacher, C. B. Jones, S. Vaid, P. Clough, H. Joho, and M. Sanderson, 'Web-based delineation of imprecise regions', *Computers, Environment and Urban Systems*, **30**, 436–459, (2006).

[2] Mark Berg, Otfried Cheong, Marc Kreveld, and Mark Overmars, *Computational Geometry: Algorithms and Applications*, Springer, 3rd edn., April 2008.

[3] M. Duckham, L. Kulik, M. Worboys, and A. Galton, 'Efficient generation of simple polygons for characterizing the shape of a set of points in the plane', *Pattern Recognition*, **41**(10), 3224–3236, (2008).

[4] Max Dupenois and Antony Galton, 'Assigning footprints to dot sets: An analytical survey', in *Spatial Information Theory: Proceedings of the 9th International Conference COSIT 2009*, eds., K. S. Hornsby, C. Claramunt, M. Denis, and G. Ligozat, pp. 227–244, Berlin, (2009). Springer.

[5] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, 'On the shape of a set of points in the plane', *IEEE Transactions on Information Theory*, **IT-29**(4), 551–559, (1983).

[6] A. P. Galton, 'Pareto-optimality of cognitively preferred polygonal hulls for dot patterns', in *Spatial Cognition VI: Learning, Reasoning and Talking about Space*, eds., C. Freksa, N. S. Newcombe, P. Gärdenfors, and S. Wölfl, pp. 409–425. Springer, (2008).

[7] A. P. Galton and M. Duckham, 'What is the region occupied by a set of points?', in *Geographic Information Science: Proceedings of the 4th International Conference, GIScience 2006*, eds., M. Raubal, H. J. Miller, A. U. Frank, and M. F. Goodchild, pp. 81–98. Springer, (2006).

[8] Antony Galton, *Qualitative Spatial Change*, Oxford University Press, 2000.

[9] Gautam Garai and B. B. Chaudhuri, 'A split and merge procedure for polygonal border detection of dot pattern', *Image and Vision Computing*, **17**, 75–82, (1999).

[10] M. Melkemi and M. Djebali, 'Computing the shape of a planar points set', *Pattern Recognition*, **33**, 1423–1436, (2000).

[11] Craig W. Reynolds, 'Flocks, herds and schools: A distributed behavioral model', in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, New York, NY, USA, (1987). ACM.

[12] Paul S. Heckbert and Eric Haines, *Berg, Mark and Cheong, Otfried*, chapter A Ray Tracing Bibliography, Morgan Kaufmann, 2002.

[13] Zena M. Wood and Antony P. Galton, 'A taxonomy of collective phenomena', *Applied Ontology*, **4**, 267–292, (2009).