REDUCING PROCESSING TIME FOR REAL-TIME MOBILE HOSTED LOCATION BASED SERVICES

Paul Gilbertson
Dept. Communication Systems,
Lancaster University
InfoLab21, South Drive,
Lancaster University, LA1 4YW
UK

Reuben Edwards
Dept. Communication Systems,
Lancaster University
InfoLab21, South Drive,
Lancaster University, LA1 4YW
UK

Paul Coulton
Dept. Communication Systems,
Lancaster University
InfoLab21, South Drive,
Lancaster University, LA1 4YW
UK

ABSTRACT

This paper presents the results of a successful development of an algorithm to reduce processing time for mobile hosted location aware software applications. It starts with a review of location based services and discusses issues surrounding various designs to place the algorithm in context.

The algorithm draws on various methodologies pioneered in the high-speed 3D graphics industry to reduce processing time checking for collisions between a user and location aware data points. The algorithm and experiment are presented in a concise manner before the results are shown.

The results conclude that the algorithm presents a very significant improvement over brute force methods. With a database of 100,000 data points a 99% reduction in processing time is observed.

I. INTRODUCTION

Location Based Services are once more on the rise in the mobile arena. Problems associated with the granularity of position information that was formerly based on network Cell ID have been tackled with the invention of new technologies and methods. These new systems allow a service to pin-point a user’s position to accuracy levels previously unheard of.

Location based products and services are currently a 10 billion Euro market, and Nokia predicts a double-digit growth rate in the next ten years [1]. This can easily be achieved with the roll-out of location-aware mobile devices and accessories to the mass market. With the explosion of satellite navigation systems into the consumer market [2] consumers are becoming more aware of the potential uses for location based data.

One method of obtaining position information is by using a GPS receiver. Although the techniques and studies in this paper can be used with any system capable of determining latitude and longitude, GPS is used throughout. GPS devices are becoming common, often as Bluetooth or SDIO accessories for PDAs and mobile phones, and are increasingly integrated with the latest mobile devices from manufacturers such as Nokia, Motorola and Ben-Q.

Location aware data is simply data that is attached to a specific geographic position. As an example can be mentioned photographs and user-created content containing information about the location covered. LocoBlog [3] is an example of GPS data being used to record the position a blog entry was made from, and allows a user to track his movements during a holiday or conference.

Section 2 of this paper looks at the structure of Location Based Services and the issues presented by a limited storage and processor device. Formats for the storage and presentation of position data are briefly reviewed in Section 3. An algorithm to reduce processing time for detecting the proximity of location aware data when in motion is discussed in Section 4. Section 5 covers a series of experiments performed comparing the presented algorithm against a brute force method and their results. The paper ends with some conclusions and discussion of further improvements to be made.

II. LOCATION AWARE DATA ISSUES

There are two major software designs for Location Based Services. The primary method is for the mobile device to be merely a client to a back-end server system. In this design the mobile device simply senses the user’s current location, passes this in a request to the servers and presents the result. All proximity checking and data storage is performed by large commercial-grade databases and servers. In this system memory and CPU time on the mobile device is not severely constrained.

The potential for a new design has been presented by the explosion of satellite navigation systems. In this design data storage and processing are performed on the device, with a few irregular requests to any central service for information updates. The major advantage of this system is that information can be easily presented in real-time with no requirement for network coverage. Mobile phones and PDAs are constrained in both CPU speed and available memory and this presents some issues.
III. ISSUES

One potential application arising from the new design paradigm is that of personal and community position-aware information access [4]. Information is stored attached to an active geographic area (typically a circle around a specific point) on the mobile device. When this device enters the active area the attached information is presented.

For this application to succeed we need to overcome a number of obstacles. First the current position of the device needs to be checked against the database frequently to ensure that data is presented to the user promptly. Secondly large data sets may need to be stored. Mobile devices have constrained storage areas, so only data that is likely to be required in the near future should be stored on the device and transactions to a central server to retrieve new information should be kept to a minimum.

Applications using this design may need to check the proximity of many individual items of data when each change in position is detected. This would require a large amount of processing power if the data set is large. For mobile devices where CPU time is at a premium, this is a significant issue. Reducing this processing time is a key objective of any algorithm.

Previous algorithms have been used to reduce the CPU time required for spatial searches. The most renowned of these is R-Tree [5] and its descendants. R-Tree subdivides the total data space into smaller areas which contain proximity based subgroups of data points. These areas are subdivided repeatedly to form a tree. To find data within a certain area the search algorithm only needs to traverse the tree to the relevant leaves and perform collisions checks on the data within those leaves. Various algorithms for performing R-Tree searches have been developed.

R-Tree suffers from the requirement that all searches start from the top of the tree, so effectively all data needs to be on or easily accessible to the device. This is not always possible on mobile devices.

IV. ALGORITHM DESIGN

The aim of any search algorithm is to reduce the amount of expensive calculations required to achieve a correct result. For the designs discussed previously, a number of assumptions can be made that simplify the operation. The first is that the mobile device will not jump randomly around the data space. In other words each update will occur close to the previous update. The second assumption is that the active areas for the data points are going to be small compared to the entire data space. This assumption has been made to reduce the need for checking points that are a significant distance from the current location.

Searches can be grouped into two categories. The first category is update searches. These searches occur when the mobile device position changes and for these searches there is no real need to check the entire data space, only nearby data points. For the second category we have full space searches that are performed on start-up or when update searches are no longer viable due to the mobile device moving away from previously close points.

The algorithm optimises the update searches significantly, and reduces the need for expensive full space searches. By reducing collision checks between the device and the active areas of various data points to only those data points in vicinity of the device, we are able to reduce processing time significantly.

A lot of the design inspiration comes from work carried out in the high-speed graphics arena for reducing computational workload on detailed terrains [6]. In these situations only the terrain close to the camera is required to be updated when the camera moves. Terrain far away, where the movement would not produce any visible change on screen, does not require any expensive computation to be performed on it.

V. ALGORITHM IMPLEMENTATION

The algorithm makes use of a series of classes that store and organise data based on its location. This enables the software to restrict its comparison of the user’s current location to data locations that are in the vicinity.
Upon initialization data is organised into a series of five buckets based on distance from the initial position. Each bucket contains information about the minimum and maximum distances of data that it contains, and the location of the centre point for this circular area. As shown in Figure 1, this creates a series of five concentric rings that represent the buckets into which data is stored. The complete database of data positions is then queried and data is organised into the buckets. Data points that fall outside the fifth bucket are ignored at this stage. Each ring is twice as wide as the previous ring.

When a user’s position changes, the data within the innermost bucket is checked for position collisions. These collisions occur when the user moves within a certain arbitrary distance of the data location. Since only the innermost bucket of data is checked, this reduces the CPU overhead required compared to checking all data points. Once a user has moved half the distance towards a bucket’s minimum range, the bucket is invalidated. When a bucket is invalidated, the distance between the user’s position and the position of the data within that bucket is compared to the bucket’s minimum and maximum ranges. Data is moved inwards and outwards to other buckets accordingly. The central position of the bucket is reset to the user’s position. This way, as a user moves towards different data positions, data that no longer needs to be checked moves outwards, and new data moves into the inner bucket and is consequently checked for collisions. When the outermost bucket is invalidated, the full database is queried for data points which now fall within the outer bucket.

The innermost bucket is invalidated each time the user moves, but outer buckets are invalidated only when required. The result of this is some lag in bucket central locations as shown in Figure 2. Five buckets and the doubling width of each bucket are used to keep the CPU time used by bucket invalidations to a minimum.

VI. EXPERIMENT DETAILS

A virtual ‘arena’ is set up measuring 30 arcminutes by 30 arcminutes and filled with a number of randomly placed data points. A virtual user is then simulated to move diagonally across this terrain from upper left to bottom right in one tenth of an arcsecond steps. Data points that come within 10 metres of the user are recorded.

The time taken for the software to run this simulation was recorded using the computer clock. Each experiment runs the simulation for a specific number of points placed in the ‘arena’ using both the algorithm detailed above and the brute force method of checking each data point at each step. The same data is used for both methods and the times of five runs are used and averaged for comparison.

The software was written in C# utilising .NET 2.0 and run on an IBM-compatible PC running Windows XP, with a 2.4 GHz Pentium IV processor and 512MB RAM. Such a high spec machine was used to reduce experiment processing time. Mobile devices with a tenth of the processing power would only scale any differences in efficiencies recorded.
VII. RESULTS

As can be seen in Table 1, in all but the least dense data conditions the above algorithm significantly outperforms the brute force method. The algorithm’s efficiency is most visible in data sets of 100,000 items as shown in Fig. 3, and early indications show that this trend continues as the density increases. The extra overhead of maintaining the buckets is more than outweighed by reduction in expensive distance checks.

Table 1. Results of Comparing the Proposed Algorithm to a Brute-Force Method over a Virtual Arena

<table>
<thead>
<tr>
<th>Data Points</th>
<th>Brute-Force Method Time (ms)</th>
<th>Algorithm Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>103.125</td>
<td>178.125</td>
</tr>
<tr>
<td>10</td>
<td>246.875</td>
<td>178.125</td>
</tr>
<tr>
<td>100</td>
<td>1,690.625</td>
<td>193.75</td>
</tr>
<tr>
<td>1,000</td>
<td>16,021.875</td>
<td>303.125</td>
</tr>
<tr>
<td>10,000</td>
<td>164,565.625</td>
<td>1,584.375</td>
</tr>
<tr>
<td>100,000</td>
<td>1,625,968.75</td>
<td>14,646.875</td>
</tr>
</tbody>
</table>

The algorithm begins to show some signs of stress at 100,000 items which is in part due to the overhead of moving new data points into the outermost bucket. This can be improved by developing better database search routines, as currently the algorithm brute force checks every item in the collection.

VIII. FURTHER WORK

Major improvements can be made in the way the algorithm draws data into the outer bucket. The current methodology is to check the entire data set for data that falls within the inner and outer bounds. This could be improved dramatically by only testing data that falls within a certain range of values. This would create a bounding box around the outer bucket and cull all data that falls outside this box. Such tests are cheap compared to the range calculation.

Mobile devices that utilise fixed-point processors may find present some quantisation issues, particularly in the range calculation. However, these issues are not expected to be significant with careful programming and testing on a variety devices, both ‘PDA’ and ‘Mobile Phone’ types is planned.

The algorithm described utilises point based data. It is trivial to modify the routines to use data of varying radii. In these changes, the range used for bucketing purposes is the distance between both central points minus the data point’s radius.

Additional work needs to be done to adopt the algorithm for use with non-circular areas. This too should be not too difficult. The shapes can first, for efficiency reasons, approximated to one or more circles, and then a more accurate collision check can be performed.

IX. CONCLUSIONS

The algorithm presents a vast improvement over simple approaches to data proximity checking for software designs that need to update as the user moves. It will enable a vast range of new real-time applications in the location-based services and location-aware gaming arenas. Other potential arenas include augmented reality systems and satellite navigation Points of Interest databases.

REFERENCES