A Motion-Aware Approach to Continuous Retrieval of 3D Objects

Mohammed Eunus Ali[1] Rui Zhang[2] Egemen Tanin[1] Lars Kulik[1]

[1]*National ICT Australia*

[1,2]*Department of Computer Science and Software Engineering*

University of Melbourne, Australia

{eunus,rui,egemen,lars}@csse.unimelb.edu.au

***Abstract*— With recent advances in mobile computing technologies, mobile devices can now render 3D objects realistically. Many users of these devices such as tourists, mixed-reality gamers, and rescue officers, need real-time retrieval of 3D objects over a wireless network. Due to bandwidth and latency restrictions in mobile settings, efficient continuous retrieval of 3D objects remains a challenge. In this paper, we describe a motion-aware approach to this problem. We first introduce multi-resolution storage and retrieval methods for 3D data, which restrict access to only the necessary content based on the client's motion pattern. We then propose a motion-aware buffer management technique as well as an efficient index using multi-resolution representations of objects. Our experiments demonstrate the effectiveness of our solution to continuous retrieval of complex spatial data in mobile settings.**

I. Introduction

Recent advances in mobile computing have delivered competitive rendering capabilities, which have enabled a new realism of visualizing 3D representation of objects on small computing devices such as cell phones or head-mounted displays. For example, in LifeClipper [1], a head-mounted display is used to offer virtually enhanced travel experiences for tourists visiting foreign locations. Virtual 3D objects are added to the view according to the current position and viewing direction of the user. While the current status of this particular project requires the tourist to carry a portable storage device for providing the data, we envision that users should only need to wear a head-mounted display. In our scenarios, clients can obtain the data on the fly through a wireless connection. For example, a tourist can use a mobile device, such as a smartphone, to see the 3D interior details of restaurants along a street without physically entering them. An electrician with augmented-reality glasses can see 3D layouts of wiring and pipes inside a wall before a repair. A rescue officer can see the structure of a building even if the building is on fire and filled with smoke. In many of these application scenarios, real-time retrieval of up-to-date data is also beneficial.

The data access issues in all of these applications fit into a common client-server model. This model consists of a *mobile client*, a *server*, a large set of *3D objects* located on the server, and a *wireless link* between the client and the server. The server stores information about the 3D objects. The client has a *view* attached to it. At any time, according to the client's location and view direction, the client retrieves all the objects within the range of its view from the server through the wireless link, and then renders the objects in the display. As the client moves, new objects will continuously be retrieved. The process can be viewed as a *continuous window query on 3D object databases*. To guarantee a realistic visual experience, the results in the query window have to be retrieved at a high rate. Typically, the link between the client and the server will be through a wireless network such as a mobile phone network, which has a high latency and low bandwidth for our purposes [2]. Thus, the wireless link to the server forms the main bottleneck for our application domain.

The data retrieval yields the highest delay when the client changes its view rapidly. This is because (i) in the same amount of time, there are more objects that is swept by the client when the view moves at a high speed, leading to more objects to be retrieved per time unit from the server, (ii) for a moving client, the usable bandwidth of a connection in a network such as mobile phone networks drop to a fraction of the bandwidth that is available for clients at rest [2].

In this paper, we provide a systematic solution to this data retrieval problem. Our solution is based on the key insight that when the client's view is moving at a high speed, the client is only interested in and capable of absorbing high level information from the environment viewed. Even if we visualize the objects in full details when the client is in fast motion, the user will not be able to see most of the details presented. Therefore, we choose to represent objects in multiple resolutions and retrieve only the data necessary to visualize the objects at the required resolutions based on the speed of the movement of the client's view. Specifically, we use wavelet representations of 3D objects. Wavelets are ideal for our needs because of the following two advantages: (i) we can easily represent an object in different resolutions using different wavelet coefficients; (ii) for an increased object resolution, we only need to retrieve the difference between the two resolutions, which incurs only incremental costs.

We take a motion-aware approach to solve the problem of efficiently processing the continuous window query on 3D object databases. Our contributions are:

- Motion-aware data retrieval: (i) representing 3D objects in multiple resolutions through wavelet decomposition, (ii) retrieval of data necessary for a certain resolution, determined by the speed of the change in the view frame, (iii) incremental retrieval of the difference when increasing the resolution.

- Motion-aware buffer management: A pre-fetching and caching strategy based on the view's movement.
- An efficient index for 3D objects in multiple resolutions represented by wavelets.

II. Related Work

Research on query processing for mobile computing is becoming increasingly important with the proliferation of location-aware mobile devices. This paper deals with continuous window query from mobile clients on static 3D object databases. It is important to note that our work is different from existing technologies such as Google Earth [3] that retrieves 2D images instead of 3D objects. Also, the main focus of this paper is to optimize the query processing for mobile wireless devices, which is not the case for Google Earth or online games. In this section, first, we briefly discuss current techniques for continuous query processing. Then, we review the concept of multi-resolution modeling of 3D objects that we exploit to cope with resource constraints of mobile devices. Next, we discuss different buffer management techniques that aim at reducing high latency for mobile wireless clients. Finally, we review existing spatial access methods for retrieving data in multiple resolutions.

Continuous Queries: Recent research focuses on continuous range and nearest neighbor queries over static and moving objects (e.g., [4], [5], [6], [7]). Besides, some research is specifically designed for mobile clients and proposes techniques for reducing communication overheads and extra processing in the server for large number of queries posed as clients change their positions [8], [9]. These techniques commonly focus on point data sets since only the locations of objects are relevant for these systems. Again, Lazaridis et al [10] focused on continuous queries for rendering objects in virtual tour-like applications and thus can be extended for non-point objects. However, none of the existing research utilizes the motion of clients for efficient data retrieval.

Multi-resolution Modeling: Triangular mesh based surface representation of 3D objects is common in computer graphics and geometric modeling [11]. These mesh representations are frequently large data sets limiting the transmission of complex 3D data to clients over low-bandwidth networks. In many applications the details of objects may not be necessary for the visualization on clients. To represent an object with various levels of details, a wide variety of models are available for mesh simplification, i.e., *multi-resolution* modeling. Two widely used approaches for multi-resolution modeling are: (i) progressive meshes [12], and (ii) wavelets [13]. Traditionally, graphics applications use progressive mesh based simplification due to the fact that this approach offers reduced costs for rendering of 3D objects. However, the wireless link forms the main bottleneck for mobile clients. It is also known that wavelet-based approaches offer a more compact coding for progressive transmission of data and thus require less bandwidth for wireless transmissions. In a wavelet-based approach, a given mesh is simplified to a base mesh, together with a sequence of missing details of the original mesh. These missing details are called wavelet coefficients. We use wavelet-based approach for representing and storing 3D objects data in multiple resolutions.

Buffer Management: To reduce the impact of high latency low-bandwidth wireless links, different pre-fetching schemes for mobile clients have been proposed [14], [15]. Since a mobile client has a limited buffer and all the data that are pre-fetched may not be used by the client, non-uniform one-dimensional motion patterns is used by [15] to define regions that are likely to be visited subsequently. Alternative techniques (e.g., [14]) assume linear movement of objects that use the speed and the direction of the client to define the region to be pre-fetched. These techniques for pre-fetching does not perform well when the movement patterns are non-trivial which is commonly the case for many mobile clients. In this paper, we propose sophisticated *state estimation*-based techniques to determine the regions that are likely to be visited where the buffer manager then fetches/caches complex 3D data based on these estimations.

Spatial Access Methods for Multi-resolution Data: Several R-tree [16] based access methods have been proposed to speed up the retrieval of progressive mesh based multi-resolution terrain data [17], [18], [19], [20]. However, none of these methods are suitable for wavelet-based multi-resolution representations that we are using in this paper. We propose a wavelet-based index for multi-resolution representations that we use for accessing 3D spatial data on mobile settings.

III. Wavelet Representation of 3D Objects

In this section, we give a brief reminder on wavelet-based multi-resolution representations of 3D objects [13]. 3D objects can be approximated by their surfaces using triangular meshes. Let M^j be a triangular mesh representation of the surface of a 3D object at resolution j. An object can be represented in different levels of resolution by a sequence of meshes M^0,M^1,...,M^J, where, M^0 is the *base mesh* and M^J is the *final mesh*. For example, Figure 1(a) shows a triangular mesh M^0 $(1, 2, 3)$ which is a coarse approximation for the surface of the given circle.

To obtain a higher resolution approximation of the given surface, the triangle $(1, 2, 3)$ is first divided into four sub-faces by introducing new vertices $(4', 5', 6')$ at edge midpoints as shown in Figure 1(b). The new set of vertices are now deformed to make the mesh to fit the surface to be approximated. For example, the vertex $4'$ is shifted to a new position on the surface and is renamed as 4. The new, finer resolution mesh M^1, is shown in Figure 1(c). Since the mesh in Figure 1(b) is obtained from the mesh in Figure 1(a) by using a regular subdivision, level 1 mesh M^1 can be obtained by adding the required displacement of three midpoint vertices $4', 5'$, and $6'$. In this case, the missing details of the mesh M^0 from the mesh M^1 can be shown as the displacement of the vertices

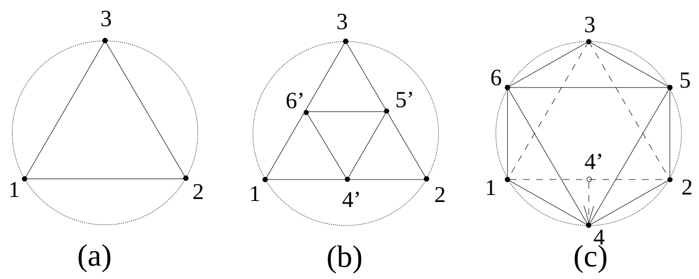


Fig. 1. (a) A coarse approximation (level 0) of a circle by a triangle, mesh M^0, (b) Mesh obtained by a splitting of M^0, (c) M^1 a level 1 approximation of the circle.

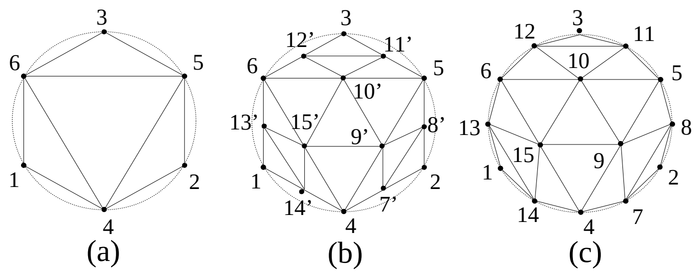


Fig. 2. (a) A level 1 approximation of the circle by a mesh, (b) Regular sub-division of the mesh at level 1, (c) M^2 as a level 2 approximation of the circle.

4, 5, and 6 from the vertices $4', 5'$, and $6'$, respectively. Thus, the wavelet coefficients that represent the difference between M^0 and M^1 are d_4^0, d_5^0, and d_6^0. For example, d_4^0 is obtained by $v_4^1 - \frac{v_1^0+v_2^0}{2} = v_4^1 - v_{4'}^1$. Similarly, Figure 2 shows the steps for obtaining a level 2 mesh M^2 that is a better approximation of the surface than level 1 mesh M^1. Each face of the mesh in Figure 2(a) is divided into four faces by introducing edge midpoints as shown Figure 2(b). Then, the new set of vertices are shifted towards the original surface to obtain M^2.

The wavelet decomposition of a mesh M^J produces a base mesh M^0 and the sets, $\{W_0, W_1, ..., W_{J-1}\}$, of wavelet coefficients. Each W_i contains a set of wavelet coefficients at level i which represent the missing details between mesh M^i and M^{i+1}. Wavelet coefficients describing a 3D object can be used as a selection criteria whether they are necessary or not at a given point of time for a client. A set of coefficients is selected according to the client's region of interest (query window), and the geometrical influences (how much effect a certain coefficient has on the visualization) of coefficients. The geometric influence of a coefficient may be determined by the speed of navigation, the resolution level of the screen, or the terminal's processing power of the client. Again, the geometric influence of a coefficient is proportional to the value of the coefficient obtained from the wavelet decomposition process. The larger the value of the coefficient, the greater the significance it has on representing the overall structure of the object. Since the smaller magnitude coefficients have less role to play in the overall structure of the object, the bandwidth utilization can be improved by discarding lower valued coefficients. Also, in a selective transmission scenario, coefficients are retrieved that are only necessary to modify the currently available version of objects in the client. Thus, at a given point in time, only a subset of coefficients are needed to be retrieved by the client.

There are many other methods for decomposing the surface of 3D objects; in this section we only show a simple process. However, our techniques described in subsequent sections can also work with many other wavelet-based representation of 3D objects.

IV. Motion-Aware Continuous Data Retrieval

In this section we introduce a motion-aware data retrieval scheme. In our approach a client uses a function to map the speed of the view to a resolution of 3D objects that are necessary for the visualization at a given point of time. The client can tune this function depending upon its environment e.g., display size, available bandwidth. Since 3D objects are decomposed and represented in multiple resolutions using wavelets, the client maps the speed to wavelet coefficients for the objects with required resolutions.

In our proposed system, a mesh representing a 3D object is decomposed into a set of wavelet coefficients. Each wavelet coefficient representing a vertex of the mesh has an associated value $0 \leq w \leq 1.0$. The larger the value of a coefficient, the greater the significance it has on the visualization of the object. Let the speed of the moving client be s, and the view (window) of the client be R. The client needs to retrieve all the coefficients necessary for the visualization of objects that fall inside the query window R. For example, when the speed is very low ($s \approx 0$) the client needs to see the objects inside R with full resolution (or full level of details). In this case all the coefficients whose values range from 0.0 to 1.0, and satisfy R need to be retrieved. Similarly, when the speed is higher, say $s = 0.5$, the client needs to retrieve less details for objects inside R. In this later case the client may only retrieve wavelet coefficients whose values range from 0.5 to 1.0, since these coefficients are sufficient for the visualization of objects for this client moving at speed $s = 0.5$. It can be observed from the above discussion that to retrieve objects with required resolutions from the server, the client needs to set the following parameters: a query window, a range determining the candidate wavelet coefficients for visualization with required resolutions.

Our algorithm incrementally retrieves spatial data from a data server. As the client moves, it sends queries to the server with different timestamps. The client only retrieves the data that is not already retrieved by the previous query. Figure 3 shows two rectangles (A, B, C, D) and (A', B', C', D') for the two query frames Q_{t-1} and Q_t at times $t-1$ and t, respectively. Assume that the client has already obtained all the data for the objects at Q_{t-1}. The client needs all the vertices for an object that fall inside the query frame and also all the neighboring vertices that connect to this set of visible vertices. This is required to properly render the objects inside a query frame. For example, for the query frame Q_{t-1}, the client retrieves the vertices $\{1, 6\}$ that fall inside the query frame and the neighboring vertices $\{3, 4, 5\}$ that connect to 1

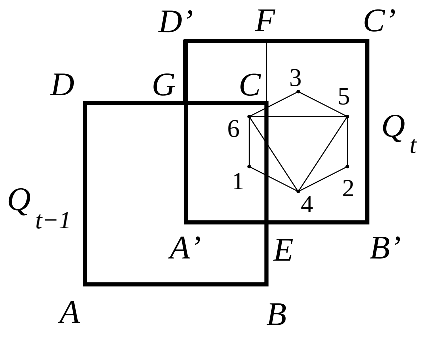


Fig. 3. Continuous data retrieval

and/or 6. Thus, at timestamp t, the client only needs to retrieve information for the region $Q_t - Q_{t-1} = (E, B', C', D', G, C)$. After receiving the region (E, B', C', D', G, C) from the client, the server divides the region along the x-axis into two rectangles (G, C, F, D') and (E, B', C', F) and executes these sub-queries separately.

After retrieving the results for all the sub-queries, the server filters the results to avoid transmitting the data that is already available at the client. In this example, the server filters out those vertices who connect to the vertices that fall inside the previous query rectangle (A, B, C, D). Therefore, the server filters out the vertices $\{3, 4, 5\}$ and only sends the vertex 2 to the client as the final result for the query frame Q_t.

Algorithm 1: ContinuousDataRetrieval

```
1.1  $O_t \leftarrow Q_t \cap Q_{t-1}$;
1.2  $N_t \leftarrow Q_t - Q_{t-1}$;
1.3  $r_t \leftarrow$ **MapSpeedToResolution**$(s_t)$;
1.4  **if** $(O_t \neq \emptyset)$ **then**
1.5      **if** $(r_t > r_{t-1})$ **then**
1.6          $R_t \leftarrow$ **Retrieve**$(\{(O_t, r_{t-1}, r_t), (N_t, 0, r_t)\})$;
1.7      **else**
1.8          $R_t \leftarrow$ **Retrieve**$(\{(N_t, 0, r_t)\})$;
1.9  **else**
1.10     $R_t \leftarrow$ **Retrieve**$(\{(Q_t, 0, r_t)\})$ ;
```

Algorithm 1 shows the steps of data retrieval. Let Q_t and Q_{t-1} be the query frames at time t and $t-1$, respectively. First, the algorithm finds the overlapping region O_t between Q_t and Q_{t-1}. The region N_t of Q_t, which is not overlapping with Q_{t-1}, is also determined. Then, the function ***MapSpeed-ToResolution*** converts the speed at time t, s_t, to the resolution at time t, r_t. This function is application depended and using a set of quality of service parameters should be adjusted by the vendor. The function ***Retrieve*** is used to retrieve objects from the server. It takes a set of parameters, where each element of this set consists of a group of parameters, i.e., a region, a lower limit, and a upper limit for the resolutions of that region. If the required resolution r_t is greater than r_{t-1} (the resolution of the previous query frame), then the client needs to retrieve additional object details for the region O_t, which are necessary to convert the objects within O_t from resolution r_{t-1} to resolution r_t. Also, objects for the non-overlapping region N_t are to be retrieved with resolution r_t. If $r_t \leq r_{t-1}$, then objects for the region O_t are already available at the client with required resolution; so only objects for the non-overlapping region N_t are retrieved with resolution r_t. Finally, if there is no overlap between Q_t and Q_{t-1}, then all the objects that falls inside Q_t are retrieved with resolution r_t.

Using the above multi-resolution continuous data retrieval scheme, the data transmission costs can be significantly reduced by selecting objects with appropriate resolutions according to the speed of a client. However, the client can still incur a high-latency while retrieving data because it has to communicate with the server for each of the query frames. To overcome this problem, we propose a buffer management scheme at the client that reduces latency.

V. Motion-Aware Buffer Management

Our buffer management scheme uses a state estimation based motion-prediction scheme to determine the probability distribution of the data to be accessed. We also continue to exploit the multi-resolution nature of the system.

A. Cost Model

We propose a cost model for a multi-dimensional non-uniform motion of a client. Our approach follows a similar model [15] that was designed for one-dimensional non-uniform motion. The cost model for the buffer management scheme has two major parts: (i) latency and (ii) data transfer costs.

Lets assume that the client pose a continuous window query to the server. Also assume that the data space is divided into grid-like blocks. When the client visits a new region (i.e., not found in the local buffer), it retrieves a number of blocks from the server and puts it in the local buffer. Thus, the client does not need to contact with the server as long as it remains in the buffered region. The latency in our system can be reduced by lowering the cache misses (i.e., when the data is not found in the local buffer) to a small value. Also, the client has a limited buffer and some of the pre-fetched data may not be actually accessed by the client. Thus, the buffer management scheme for a client should also avoid the retrieval of redundant data to minimize the wasted bandwidth.

Let T_q be the total duration of a continuous query, M be the average number of local cache misses during the total duration of that query, and $N(j)$ be the number of blocks needed to be retrieved at jth local miss. Let C_c and C_t be the connection establishment and the data transfer costs, respectively, for a unit block of data with size B from the server to the client. Then the total data transfer costs for a continuous query from a mobile client can be determined as follows:

$$C = \sum_{j=0}^{M} (C_c + C_t \times B \times N(j)) \tag{1}$$

The data transfer costs will be less for smaller values of M.

In [15], a pre-fetching model is described for a one-dimensional setting, where a client can move to the left with probability p_l or to the right with probability p_r. Assuming the client can buffer $a-1$ blocks, where $a-1>1$, then, the client should buffer $(n-1)$ blocks on the left and $(a-n-1)$ on the right so that the average residence time $T_{a,n}$ that the client spends in the pre-fetched blocks is maximized. Since $M = T_q/T_{a,n}$, the average cache misses M will be minimal when the average residence time is maximized. For this, a position n_{opt} in the space that maximizes $T_{a,n}$ can be obtained according to [15] as follows:

$$n_{opt} = \frac{\log(\frac{(\frac{p_l}{p_r})^a - 1}{a.\log\frac{p_l}{p_r}})}{\log\frac{p_l}{p_r}} \tag{2}$$

Since the movement of a client cannot be restricted to one dimensional space, we extend the above model and partition the plane around the client into equally sized sectors. Each sector represents one of the k possible directions of the client (see Figure 4(b), where $k = 4$). Let p_i be the probability that the client will move in direction i. The client needs to determine the optimal assignment of the available buffer such that the client's average time spent inside the buffered regions is maximized. Let n_i be the number of blocks that need to be assigned in direction i, and $n_{(i,i')}$ be the summation of all the blocks that need to be assigned for the directions from i to i'. Thus, we have $n_{(1,k)} = a-1$. Then the buffer assignment process for different directions can be described as follows.

First, we partition all the direction probabilities into two groups such that $p_l = \sum_1^{\lfloor\frac{k}{2}\rfloor}(p_i)$, and $p_r = \sum_{\lfloor\frac{k}{2}\rfloor+1}^{k}(p_i)$. Then we compute $n_{(1,\lfloor\frac{k}{2}\rfloor)}$ using equation (2). Hence, $n_{(\lfloor\frac{k}{2}\rfloor+1,k)} = n_{(1,k)} - n_{(1,\lfloor\frac{k}{2}\rfloor)}$. Similarly, we can calculate $n_{(1,\lfloor\frac{k}{4}\rfloor)}$ by using the equation (2), where $p_l = \sum_1^{\lfloor\frac{k}{4}\rfloor}(p_i)$, $p_r = \sum_{\lfloor\frac{k}{4}\rfloor+1}^{\lfloor\frac{k}{2}\rfloor}(p_i)$ and $a-1 = n_{(1,\lfloor\frac{k}{2}\rfloor)}$. This partitioning process continues until there is a single direction in each partition. After computing $n_{(1,2)}$, equation (2) is used to calculate n_1, where $p_l = p_1$ and $p_r = p_2$. Then, we have $n_2 = n_{(1,2)} - n_1$. Similarly, the values $n_3, n_4, ..., n_k$ can be calculated for other directions that maximize the average residence time $T_{a,n}$. Therefore, we have $n_1, n_2, ..., n_k$ portions of the total buffer assigned for the directions with probabilities $p_1, p_2, ..., p_k$, respectively.

Using the above process, we find a set of values $n_1, n_2, ..., n_k$ for a particular ordering of k directions. There can be $k!$ possible orderings of the directions and different orderings of directions may result in different values for $n_1, n_2, ..., n_k$. Among all possible orderings, we select the result with the maximum average residence time. However, our results show that this step can be omitted as the ordering only slightly affects the average residence time.

Similarly, we can extend the model described in [15] and calculate the number of blocks $N(j)$ to be retrieved at jth local cache miss for k directions.

Based on the above formulation, we can now decide what portion of a given buffer should be allocated for which direction. For this, we need to first determine the probabilities of the client to visit neighboring regions.

B. Motion Prediction

We use the *Kalman filter* [21] to predict future positions of a client from its recent locations and compute the future error covariances to determine the confidence level of those predictions. Based on the covariances and predictions, we calculate the probabilities of the client to visit neighboring regions.

The state s_t of a moving client at time t is defined by using the positions of this client from the h most recent timestamps, i.e., $s_t = [p(t), p(t-1), ..., p(t-h)]^T$, where $p(t)$ is a position vector. Thus, the prediction of a future state at time $t+1$ is $s_{t+1} = As_t$, where A is the transition matrix, called a one-step predictor. A can also be used for multi-step predictions. For example, a state at time $t+i$ can be calculated as $s_{t+i} = A^i s_t$. The transition matrix A can be calculated by using the recursive least-squares estimation method [22] as well as other methods [23] from recent states.

After predicting the future state, we estimate the expected confidence of a predicted state by calculating the error covariances of that state. If $\hat{s}_t$ is the predicted state and s_t be the true state, the error of the prediction is $e_t = s_t - \hat{s}_t$. Similarly, the prediction error for the state $t+i$ is $e_{t+i} = A^i(s_t - \hat{s}_t)$. From this, the covariance matrix P_t, which is a measure for the uncertainty of the predicted state $\hat{s}_t$, can be defined as $P_t = E[e_t e_t^T]$. Then, we estimate the probability of the predicted state $\hat{s}_t$ conditioned on all prior estimates. Hence, the probability of a state can be estimated using a normal probability distribution [21].

$$P(s_t) \sim N(E(s_t), P_t) = N(\hat{s}_t, P_t) \tag{3}$$

The predicted value $\hat{s}_t$ is the mean for the probability distribution and the variance for this prediction is obtained from the singular values of P_t.

Since the probability function is continuous, each point in the data space can have a distinct probability value. Rather than calculating the probability of each possible point location, which is a very costly operation, we divide the total space into grid cells and then calculate the probabilities for different blocks that can be visited by a client.

Figure 4(a) shows the current position l and query frame Q_t at time t of a client. Lets assume that the next query frame Q_{t+1} can be predicted at l^1, l^2, and l^3, with probabilities 0.5, 0.3, and 0.2, respectively. Similarly, we can calculate the probabilities of blocks to be visited by Q_{t+2} at time $t+2$. By iterating this process, the surrounding blocks are assigned different probabilities. Figure 4(b) shows the probabilities of different neighboring blocks to be visited by the client. Based on these probabilities, we can approximate the probabilities of a client moving in different directions. Figure 4(b) shows the partitions of blocks into four directions as shown by the dotted partition lines. A block that intersects a partition line is assigned to one of the two partitions that owns the maximum region of that block. If a block is equally owned by

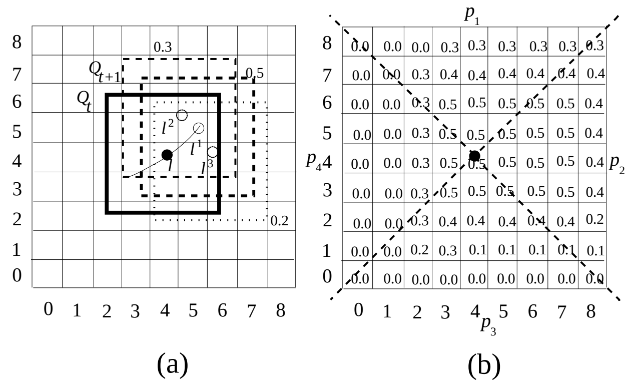


Fig. 4. (a) Motion prediction for the next timestamp and (b) probabilities of visiting different cells.

two partitions, we resolve the dispute by assigning alternative blocks in different partitions. For example, blocks (5,5), (6,6), (7,7), and (8,8) intersect the partition line between direction 1 and 2. In this case if the blocks (5,5) and (7,7) are assigned for the direction 1, then the blocks (6,6) and (8,8) are assigned for the direction 2; or vice versa. Finally, the probability of the client going in a specific direction can be obtained by summing up all the probabilities of cell blocks for that direction and then normalizing the sum by dividing it with the total sum of the probabilities.

After obtaining the probabilities in different directions, we apply the method described in Section V-A to determine what portion of the available buffer is assigned for each direction.

The summary of the process of buffer management is: (i) estimate the client's path and probabilities of surrounding cell blocks to be visited, (ii) select the list of blocks to be put into the buffer from each of the directions, (iii) retrieve objects from the server for the predicted blocks which are currently not in the client's buffer.

Our buffer management scheme also utilizes the motion of clients to adapt a multi-resolution strategy. The idea is that a client moving at higher speeds buffers more objects with lower resolutions than that of a slowly moving client. The process is similar to our motion-aware continuous data retrieval strategy and thus is not given in detail for the brevity of the current presentation.

VI. Indexing Wavelets of 3D Objects

In this section we propose a novel technique for indexing the wavelets of 3D objects. We have utilized one of the important properties of wavelets known as support regions that ensure optimal data retrieval for a window query. We will first propose a straight forward approach for indexing the wavelets. Then we give the formal definition of support regions of wavelets and discuss some properties of support regions. Finally, we propose our technique to efficiently index the wavelets that can reduce the I/O costs significantly.

An R-tree [16] can be used to index the positions of wavelet coefficients and the associated values resulted from the decomposition process described in Section III. The position of a wavelet coefficient is represented by a point vertex (x, y, z) in a three-dimensional space and the value of the wavelet coefficient is a numerical value w. A 4D-R-tree can index a wavelet coefficient, where the first three dimensions represent the position and the fourth dimension stands for the value.

Lets assume that a window query $Q(R, w_{max}, w_{min})$ with the region of interest R, where w_{max} and w_{min} are the upper and lower bounds of the coefficient values (w) for the required level of objects' details within the region of interest, respectively, is submitted from a client. For this, all the coefficients (vertices) that fall inside the query rectangle are retrieved first. However, these coefficients are not sufficient for the required visualization inside the query rectangle, because the coefficients that are associated with the neighboring vertices of these already retrieved coefficients also contribute to the visualization for the region of interest R. Therefore, after retrieving initial sets of coefficients, we compute a bounding region that encloses all the neighboring vertices and re-execute the query for the extended region. The problem with this access method is as follows. This access method is not optimized for the retrieval costs because it cannot avoid multiple retrievals of wavelet coefficients and hence incurs high I/O costs. Moreover, additional information, neighboring vertices, are also needed to be stored for each of the vertices. To overcome these limitations and to facilitate optimal data retrieval, we plan to utilize the support regions of wavelets.

A. Support Regions of Wavelets

The support region of a wavelet represents a region of the object to which the wavelet contributes during reconstruction of the 3D surface. For example, the wavelet coefficient associated with the vertex v_4 in Figure 1(c) has the value $v_4 - v_{4'}$ and the support region as the polygon $(1, 4, 2, 5, 6)$. If w_i^j is an ith wavelet coefficient of level j mesh M^j, then the support region r_i^j of this wavelet coefficient is the region of M^j to which the wavelet coefficient contributes during the reconstruction of the next level finer resolution mesh M^{j+1} from M^j.

It is important to note the following property for support regions. Let $W_1 = \{w_1, w_2, ..., w_n\}$ be a set of n wavelet coefficients that covers the region R_1 and let $W_2 = \{w_1, w_2, ..., w_m\}$ be another set of m wavelet coefficients that covers the region R_2, where $m \leq n$. Here, $W_2 \subseteq W_1$ and $R_2 \subseteq R_1$. If a new wavelet coefficient w_k, $k \geq n$, is added to both W_1 and W_2, and the regions affected in R_1 and R_2 by the support region of w_k are $R_1^{'}$ and $R_2^{'}$, respectively, then $R_2^{'}$ is also a subset of $R_1^{'}$ that is $R_2^{'} \subseteq R_1^{'}$. This observation is trivial from the following set of simpler observations. Let each wavelet coefficient w_i represent a region r_i. Then $R_1 = \cup_{i=1}^{n} r_i$ and $R_2 = \cup_{i=1}^{m} r_i$. When w_k is added to the set W_1, the affected region is $R_1^{'} = R_1 \cap r_k$. Similarly, $R_2^{'} = R_2 \cap r_k$. Since $R_2 \subseteq R_1$, then $R_2^{'} \subseteq R_1^{'}$.

For example, Figure 5(a) shows the support regions r_1, r_2, r_3 for the wavelet coefficients w_1, w_2, w_3, respectively, and the region affected by the support region r_4 of w_4 is shown as a filled rectangle, while Figure 5(b) shows the support regions r_1, r_2 of wavelets w_1, w_2, respectively, and the portion of original region affected by the support region of w_4 is

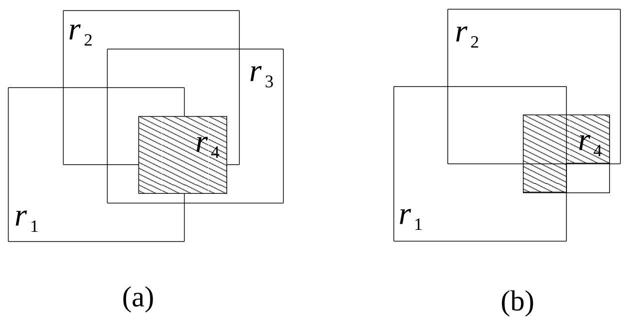


Fig. 5. Regions affected by the support region r_4 of a wavelet

shown as filled polygons. This property of a support region of a wavelet helps us design an efficient index that facilitates the efficient retrieval of data for a given query frame.

B. An Efficient Index

We use a 4D R-tree to index wavelet-based representations of 3D objects. We utilize coefficient values and support regions of wavelets for indexing the data. The first three dimensions represent the three axes x, y, z that are used to index the Minimum Bounding Box (MBB) of the support region of each wavelet, and the fourth dimension represents the normalized value of the wavelet coefficients w. Hence, each wavelet coefficient represents a region in the x, y, z, w plane. The minimum bounding rectangles (MBRs) for the support regions of wavelets representing the vertices 4, 5, and 6 are shown as $[A_1, A_2, A_3, A_4]$, $[A_1, A_2, A_5, A_6]$, and $[A_1, A_2, A_5, A_6]$, respectively, in Figure 6(a). Similarly, MBRs of level two wavelets are shown in Figure 6(b), where $[A_7, A_8, A_9, A_{10}]$ is the MBR for the support region of vertex 7. These examples show that the bounding rectangles can enclose two-dimensional support regions. For 3D objects, the support region of each wavelet coefficient is enclosed with a MBB.

Figure 7 shows an example for representing wavelet coefficients in a node of a 3D R-tree for 2D objects. In this example, x and y dimensions are used to represent MBRs of wavelets, and w corresponds to the coefficient value for the wavelet. Figure 7 shows MBRs with dotted boundary lines for three different w values 0.0, 0.3, and 0.7.

For a window query $Q(R, w_{max}, w_{min})$, when a client needs to retrieve objects with the finest resolution, then it sets $w_{max} = 1.0$ and $w_{min} = 0.0$. It means that it retrieves all the wavelet coefficients irrespective of their values, whose MBRs intersect with the query rectangle R. On the other hand, if a client needs to retrieve objects with the lowest resolution it sets $w_{max} = 1.0$ and $w_{min} = 1.0$. In this case, it only retrieves the wavelet coefficients having value 1.0 required for the lowest resolution representation of objects.

In addition, the client can set any appropriate values for w_{max} and w_{min} to support progressive retrieval of objects. Lets assume a scenario where a client has all the coefficients having values greater than 0.7 for a given query rectangle R. If the client needs objects with the finest resolution for the query, it just needs to send a query $Q(R, 0.7, 0.0)$ as shown with a box in Figure 7.

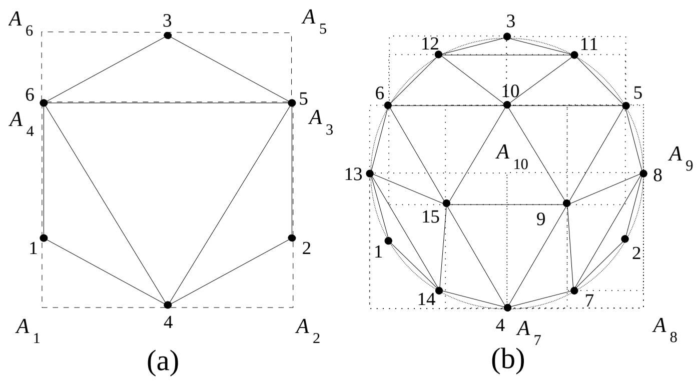


Fig. 6. Minimum bounding rectangles of support regions of (a) level 1 wavelets of Figure 2(a), and (b) level 2 wavelets of Figure 2(c).

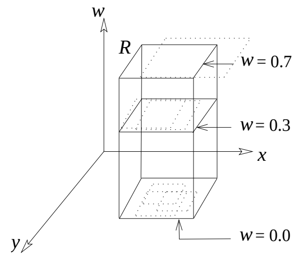


Fig. 7. Wavelet coefficients and support regions in our index

In general, let $Q(R, w_{max}, w_{min})$ be the query with the region of interest R, where, w_{max} and w_{min} are the upper and lower bounds of wavelet coefficients for the required resolutions, respectively, and $0.0 \leq w_{min} \leq w_{max} \leq 1.0$. Then, our motion-aware access method gives the minimum number of wavelet coefficients necessary for the query Q. Let $R_1 = \{r_1, r_2, ..., r_n\}$ be the set of support regions of n wavelet coefficients retrieved using our motion-aware index for the query Q. The set contains all the support regions that fall inside or intersect with Q. Assume that another method gives a set of support regions $R_2 = \{r_1, r_2, ..., r_n\} - \{r_k\}$ of $n - 1$ wavelet coefficients for the query Q. Here, $1 \leq k \leq n$, and hence, $R_2 \subset R_1$. In this case, the wavelet coefficient w_k associated with the support region r_k is absent. Therefore, objects inside the region $R \cap r_k$ lack some details that would otherwise be contributed by the wavelet coefficient w_k. Hence, any method that retrieves less data than that of our method gives is not sufficient.

VII. Experimental Study

We present our experimental results in this section. Since we have a suit of components to optimize the processing of continuous window queries on 3D object databases, we first evaluate each of the components independently. Then we combine all components and compare the system performance of our motion-aware schemes with naive schemes.

A. Experimental Setup

We have set-up our experiments based on a realistic augmented-reality city tour. We have created tours augmented with 3D objects (e.g., representing old buildings in cities) that

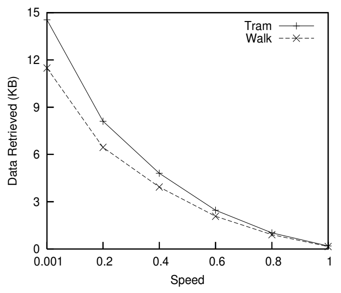


Fig. 8. Effect of speed on data retrieval

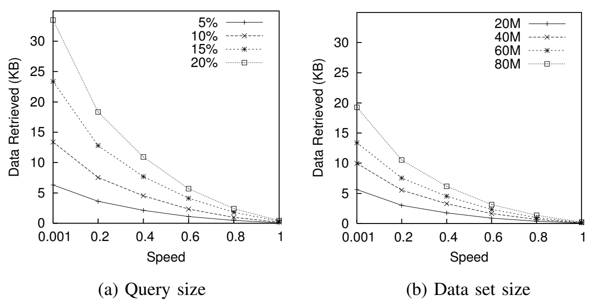


Fig. 9. Effect of query size and data set size on data retrieval

are distributed uniformly throughout the data space. We vary the data set sizes as 20MB, 40MB, 60MB, and 80MB by placing 100, 200, 300, and 400 objects in the data space. The default data set size in our experiments is 60MB. Also, we have collected and approximated the head movements of 10 tourists in two different settings: (i) tram tours, (ii) pedestrian tours. As a client moves from a given starting point towards a destination, it connects to the server through wireless links to retrieve the 3D objects. In the experiments, the bandwidth and the latency of the wireless links are 256Kbps and 200ms, respectively. We also vary the length and the width of the query frame by taking 5%, 10%, 15%, and 20% of the length and the width of the total data space, where 10% is the default for our experiments.

We assume that the speed of the client reveals the detail of information that the client is willing to consume; thus, in our experiments the speed is expected to be inversely proportional to the value of the wavelet coefficients retrieved. All coefficient values are normalized to the range [0.0,1.0]. When the speed is at a normalized maximum (i.e., 1.0), only the coefficients which have the highest geometric influence need to be retrieved. Since all the vertices in the coarsest version of an object have coefficient values 1.0, these vertices are retrieved for the fastest clients. If the speed is very slow (i.e., close to 0.0), all the coefficients between 0.0-1.0 are retrieved, leading to all the objects being retrieved with the highest resolution.

B. Evaluation of Motion-aware Continuous Retrieval

In the first set of experiments, we show the effect of the speed of the clients on continuous data retrieval. We expect that the costs of data retrieval can be reduced significantly by selecting objects with appropriate resolutions according to the speed of clients. We measure the amount of data retrieved by clients traveling similar distances at varying speeds. Figure 8 shows the average amount of data retrieved by clients on trams and on foot at different speeds (normalized to 0.001-1.0). Since, the size of the objects with the highest resolution is higher than that of the objects with the lowest resolution, we expect that the data size required for clients moving with the highest speed should be significantly less than that of clients that move slowly (Figure 8). In Figure 9(a), we measure the average amount of data retrieved for tram tours by varying the length and the width of the query frame 5%, 10%, 15%, and 20% of the length and the width of the total data space. In Figure 9(b), we vary the data set size as 20MB, 40MB, 60MB, and 80MB and measure the average amount of data retrieved for tram tours at varying speeds. These figures show that the amount of retrieved data decreases significantly with the increase of speed for different query and data set sizes. We see that for large query frames and data sizes, absolute benefits of our multi-resolution technique are more pronounced.

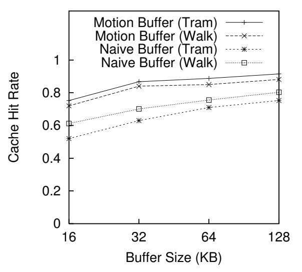
C. Evaluation of Motion-aware Buffer Management

In the second set of experiments, we compare our motion-aware buffer management technique with a naive approach where all the surrounding regions of a query frame are buffered with equal probabilities. We compare the cache hit rate (which is a measure of reduction in latency), and the data utilization (which is a measure of data transfer overheads due to pre-fetching) of the motion-aware buffer management scheme to the naive buffer management scheme.

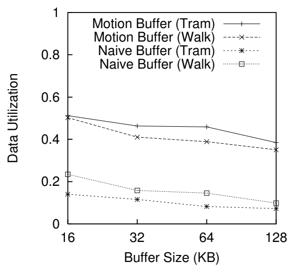
Effect of Buffer Size: The more buffer space a client has, the more data it can put into the buffer. However, to fill a large buffer, a client pre-fetches more data by predicting positions of the query frame far into the future. Hence, there is a chance that redundant data may be pre-fetched with very long term motion predictions. In this experiment, we vary the buffer size from 16KB to 128KB (Figure 10). The speed of the clients may also slightly vary at different parts of a tour for this experiment as the data uses a seed travel pattern collected from movements of real-world clients.

Figure 10(a) shows that the cache hit rate increases with the increasing buffer size because a larger buffer can hold more data. For a 16K buffer, the cache hit rate for tram tours and pedestrian tours are 75% and 72% for our motion-aware technique, whereas, for a 128K buffer, 91.5% (tram) and 88% (walk). (Tram tours give superior cache hit rates because these tours can be predicted more accurately than the pedestrian tours.) We also observe that the cache hit rate for the motion-aware approach is always better than the naive approach, i.e., on average 32% and 15% better for tram tours and pedestrian tours, respectively.

An efficient buffer management scheme should avoid pre-fetching data that will not be used by the client. From this point of view, the used portion of the total pre-fetched data is one of the major metrics for a good buffer management scheme. Figure 10(b) compares the data utilization of the motion-aware scheme and the naive scheme for both tram

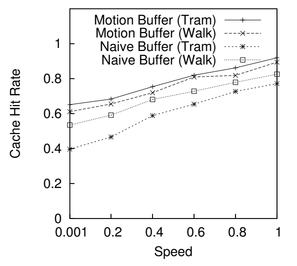


(a) Cache hit rate

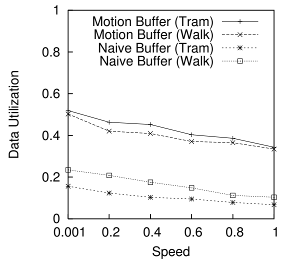


(b) Data utilization

Fig. 10. Effect of buffer size



(a) Cache hit rate



(b) Data utilization

Fig. 11. Effect of speed

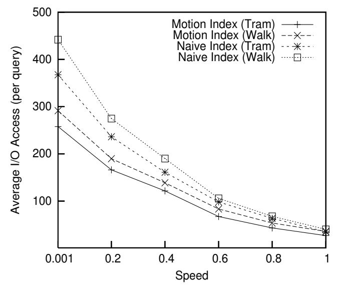
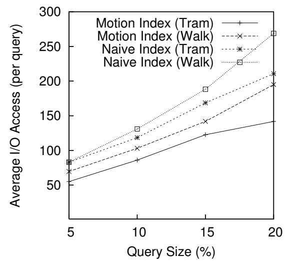
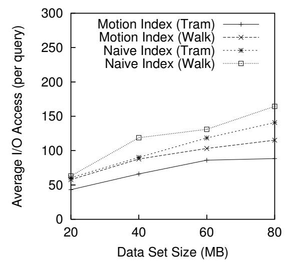


Fig. 12. Effect of speed



(a) Query size



(b) Data set size

Fig. 13. Effect of query and data set sizes

and pedestrian tours. The data utilization in our motion-aware scheme is 51% for trams and 50% for walking with a 16K buffer. The utilization drops to 38% (tram) and 35% (walk) for a 128K buffer. Hence, with the increase in buffer size the data utilization decreases as the client cannot make accurate predictions far into the future. The data utilization in the naive buffer management scheme is 14% (tram) and 23% (walk) for a 16K buffer, whereas, the utilization is 7% (tram) and 9% (walk) for a 128K buffer. Hence, the data utilization in our approach is always better than the naive approach (on average 3.5 times and 1.7 times for tram tours and pedestrian tours, respectively).

Effect of Varying Speed: Our buffer management scheme also uses a multi-resolution representation of objects. Figure 11 shows that, the cache hit rate increases from 64% to 91% (tram) and 61% to 89% (walk) with the increase of speed as more data can be buffered with lower resolutions. However, due to long distance predictions, we see that the data utilization is less at higher speeds than that of lower speeds. Our motion-aware approach achieves higher (on average 33% for tram tours and 10% for pedestrian tours) cache hit rates. We also have a higher data utilization (35%-51%) in comparison with that of the naive approach (7%-23%).

D. Evaluation of Motion-aware Indexing

In the third set of experiments, we evaluate our advanced indexing and access strategy that is built on wavelet-based multi-resolution objects (in comparison to the simpler index proposed in Section VI). We implement a $3D$ $(x - y - w)$ R^*-tree [24], where the page size and the node capacity are set to 4K and 20, respectively.

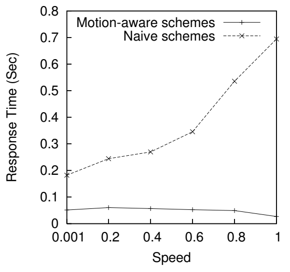
Effect of Varying Speed: First, we observe the effect of speed while retrieving multi-resolution objects from our index. Figure 12 shows that when the speed is high, i.e., in the range of 0.9-1.0, we require approximately 8-11 times less I/O costs than the costs for clients moving at the lower speeds (i.e., 0.001). This is because most of the wavelet coefficients have very small values and have almost insignificant geometric influence on the geometry of objects, i.e., leading to these coefficients not being retrieved for higher speeds. Our index structure avoids the retrieval of any extra data than required, whereas the simplistic approach requires a larger amount of retrievals. Figure 12 shows that our motion-aware access method incurs much less (21%-52%) I/O costs than the naive approach.

Effect of Query and Data Set Sizes: In this experiment, we vary the query size and keep the data set size at 60MB and the speed at 0.5 for each of the tours. Figure 13(a) shows that the data retrieval costs increase with the size of the query. Also, our motion-aware access strategy incurs on average 36% less I/O costs than the naive approach. The improvement is more prominent for larger query size which is up to 49%.

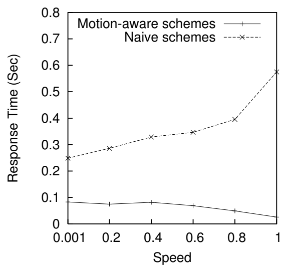
We also show the scalability of our index and access strategy by varying the data set size from 20M to 80M, Figure 13(b). In this case query size is fixed at 10% and the speed is 0.5. The results show that as the data size increases, the cost difference is more pronounced between our approach and the naive method. The improvement is 59% for the largest data set size of 80MB.

E. Overall System Performance

Finally, we compare the overall performance improvement of our motion-aware system with a naive non-multi-resolution technique. To obtain a naive system, we always retrieve objects with the highest resolution and we use an R^*-tree [24] to

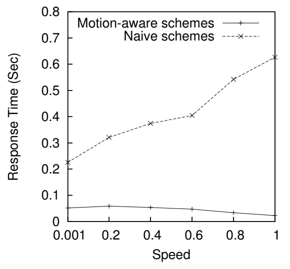


(a) Tram

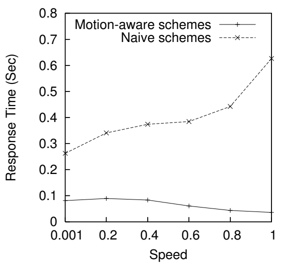


(b) Walk

Fig. 14. Query response time (uniform)



(a) Tram



(b) Walk

Fig. 15. Query response time (Zipf)

index objects without using multiple resolutions. We also use a simple Least Recently Used (LRU) scheme for caching. In these experiments, each client travels for the same duration of time at varying speeds. Hence, when a client is traveling at a higher speed, it covers a larger area in the city than that of a slowly moving client. We measure the average response time for queries (with size 5%) from the clients moving at different speeds. Experimental results in Figure 14 and Figure 15 show that, for both uniform and Zipfian data sets, the query response time for both tram and pedestrian tours in our motion-aware approach is on average 23 times less costly than that of the naive approach at high speeds (i.e., 1.0). The improvement is on average 3.5 times when the speed of the client is low (i.e., 0.001). Thus, these results reveal that the performance of the naive system degrades with the increase of speed, because a large number of objects need to be retrieved in a short period of time. However, the motion-aware approach can cope with the speed by retrieving lower resolution objects. The tram tours show a slightly lower response time than that of pedestrian tours because tram tours can be predicted more accurately.

VIII. Conclusion

In this paper, we introduce a motion-aware approach for continuous retrieval of 3D objects for mobile clients where regions of interests change continuously with the motion of the clients. First, we show that the speed of the clients can be used to reduce the data transfer costs significantly by facilitating incremental data retrieval in multiple resolutions. Then, to reduce the high latency in a wireless link for a large number of queries from a client, we propose a motion-aware multi-resolution buffer management scheme that pre-fetches and caches data based on a probability model derived from the motion of the clients. Also, we introduce an efficient index using wavelet-based multi-resolution data that can significantly reduce the I/O costs by giving the minimum amount of possible data to a query. Our experimental evaluations show that the system that adopts motion-aware continuous data retrieval schemes observe significantly less query response time than that of a system that does not consider the motion of clients.

Acknowledgment

We would like to thank the anonymous reviewers for their comments that improved our paper. This work is partially supported under the Australian Research Councils Discovery funding scheme (project number DP0880215).

References

[1] LifeClipper, *http://www.torpus.com/lifeclipper/*.
[2] Ofcom, *http://www.ofcom.org.uk/static/archive/Oftel/publications/research/2002/benchint1202_56.htm*.
[3] GoogleEarth, *http://earth.google.com/*.
[4] B. Gedik and L. Liu, "MobiEyes: a distributed location monitoring service using moving location queries," *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, pp. 1384–1402, 2006.
[5] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: scalable incremental processing of continuous queries in spatio-temporal databases," in *SIGMOD*, 2004, pp. 623–634.
[6] K. Mouratidis and D. Papadias, "Continuous nearest neighbor queries over sliding windows," *TKDE*, vol. 19, no. 6, pp. 789–803, 2007.
[7] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *VLDB*, 2002, pp. 287–298.
[8] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing," in *SSTD*, 2001, pp. 97–116.
[9] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD*, 2003, pp. 443–454.
[10] I. Lazaridis, K. Porkaew, and S. Mehrotra, "Dynamic queries over mobile objects," in *EDBT*, 2002, pp. 269–286.
[11] D. P. Luebke, *Level of Detail for 3D Graphics: Application and Theory*. San Francisco, CA: Morgan Kaufmann, 2003.
[12] H. Hoppe, "Progressive meshes," in *SIGGRAPH*, 1996, pp. 30–99.
[13] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, *Wavelets for Computer Graphics: Theory and Applications*. San Francisco, CA: Morgan Kaufmann, 1996.
[14] G. Cho, "Using predictive prefetching to improve location awareness of mobile information service," in *ICCS*, 2002, pp. 1128–1136.
[15] V. de Nitto Person, V. Grassi, and A. Morlupi, "Modeling and evaluation of pre-fetching policies for context-aware information services," in *Mobicom*, 1998, pp. 55–65.
[16] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.
[17] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *IEEE Visualization*, 1998, pp. 35–42.
[18] M. Kofler, M. Gervautz, and M. Gruber, "R-trees for organizing and visualizing 3D GIS database," *Journal of Visualization and Computer Animation*, vol. 11, no. 3, pp. 129–143, 2000.
[19] L. Shou, Z. Huang, and K.-L. Tan, "HDoV-tree: The structure, the storage, the speed," in *ICDE*, 2003, pp. 557–568.
[20] K. Xu, X. Zhou, and X. Lin, "Direct Mesh: a multiresolution approach to terrain visualization," in *ICDE*, 2004, pp. 766–772.
[21] G. Welch and G. Bishop, "An introduction to the kalman filter," *SIGGRAPH 2001 Course*.
[22] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris, "Online data mining for co-evolving time sequences," *Technical Report, Carnegie Mellon University*, May 1999.
[23] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *SIGMOD*, 2004, pp. 611–622.
[24] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: an efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331.