UCLA CSD TR030019

Integration of Real-Time Information into a Virtual Environment *

Scott Friedman Richard Muntz Matthew Yeo

Department of Computer Science
University of California, Los Angeles
{friedman,muntz,myeo}@cs.ucla.edu

ABSTRACT

Advances in sensor technology and wireless communication are enabling the real-time (or near real-time) collection of information about the physical world. This raises many interesting possibilities as well as challenges for acquisition and visual presentation of this data. We start with a fully interactive, photorealistic 3D model of an urban environment. A user "flies" through the model and relevant sensor data which impacts the rendering of the user's current field of view must be fused with the static model in a timely fashion. Within this environment we assume sensor data of various kinds may be available and the goal is to acquire this information and integrate it into the 3D model of the static environment as needed. One application scenario might be for emergency response.

We describe our testbed application which at present can track moving objects and integrate position data into the 3D urban simulation model. We discuss the tradeoffs between latency in the display of object positions in order to correct/smooth the noisy measurement data as well as a number of system resource optimization issues. The testbed is operational and actual experience with the system is reported.

CR Categories: H.5 [Information Interfaces and Presentation]: Multimedia Information Systems— [J.7]: Computers in Other Systems—Real-time, Command and Control

Keywords: interactive visualization, data filtering, sensors

1 INTRODUCTION

Advances in sensor technology and wireless communication are enabling the real-time (or near real-time) collection of information about the physical world. This raises many interesting possibilities as well as challenges for acquisition and visual presentation of this data. We start with a fully interactive, photorealistic 3D model of an urban environment. Within this environment we assume sensor data of various kinds may be available and the goal is to acquire this information and integrate it into the 3D model of a static environment as needed. One application scenario might be for emergency response. A central coordinator may, for example, be getting location information for various vehicles, personnel, and other assets as well as environmental sensor readings such as temperature, air pollutants, wind direction, etc. The visual display which incorporates the known static infrastructure as well as the real-time information provides obvious advantages. For example, the coordinating person can "see" the same context that a person on the scene may be seeing. Also, context is often important to leverage human cognitive powers and common sense. For example, the visualization could help the coordinator realize that a fire is approaching a hazardous chemical factory. Automating recognition of this situation requires an a priori specification of a very large variety of such dangers which is problematic. Even context hidden from an on-the-scene observer, for example an underground gas pipeline, could be observed in the simulation. These are examples of critical context information that is part of the static model. Sensors also raise the possibility of observation of information that is not directly available, even to those on the scene. For example, underground real-time measurements of contaminants, temperature, water, etc.

Figure 1: UCLA Campus Virtual Model.

Our testbed application at present is confined to the tracking of moving objects and the integration of a visualization of these objects into the 3D urban simulation model. Specifically we deal with actual GPS data gathered from transportation vehicles on the UCLA campus. Figure 1 shows a snapshot from a model of UCLA with two of the campus buses in view. One of the major challenges to be addressed is the imprecision of the GPS data. Due to the noise in the data, it cannot be directly used to position the object in the simulation. This raises questions such as (a) how to smooth or correct the reported values and (b) what the tradeoffs are with latency in the displayed location of the objects. The issue of smoothing or correcting the data is not just a signal processing problem since there is also an issue of compatibility with what is known (or assumed with high probability) such as that vehicles move on roads, that a vehicle has its wheels on the ground, etc. Latency is often a communications issue but here the problem can be that some position readings for a time t can only be corrected after the measurements through time $t + \tau$ are known. Another issue concerns optimizations of various resources in the system. (The system includes the devices on mobile objects that measure and transmit location data, the software for smoothing the raw location data,and the urban simulation.) Issues include how and in what form the object positions are updated in the simulation and also optimizations in which only objects within the viewing frustum of some observer need report their position. Such optimizations involve consideration of many factors

*This work was supported by NSF grants IIS-0086116, ANI-0085773 and EAR-9817773.

including communication delays, latency requirements of the data correction/smoothing algorithms, motion models/constraints for the objects being tracked, and also any constraints on the movement of the viewing frustum. Resource optimizations can also exploit the fact that the aesthetic visual requirements for an object distant from the viewer allow for less precise information. This can translate, for example, into less frequent reporting by an object which is visible but at considerable distance. Consider for example, a user at ground level observing vehicles going through an intersection versus a user at 2000 feet with a bird's eye view of campus. In the former case, there are few vehicles in view at any time but they need to be represented with fine-grained precision. In the latter case, most vehicles may be visible but a more gross representation of their movement will suffice. Since the user can "fly" from the intersection to the a 2000-foot high vantage point, the system should smoothly transition from the one extreme to the other.

This paper is organized as follows. In section 2 we discuss the overall architecture and software components. In particular the urban simulation, the vehicle GPS units, error characteristics of the data, and some characteristics/assumptions based on the objects being tracked, which in the testbed, are campus transit buses. In Section 3 we present the algorithms used for cleaning/smoothing the reported GPS readings, the performance considerations and resulting protocols for communication to the urban simulation of object motion, and methods for minimizing the communication required to and from objects based on visibility. In Section 4 we compare our environment and objectives to the most closely related work, and we offer conclusions in Section 5.

2 Overview of System Architecture

The architecture of our testbed is composed of four principle components: sensor platform, data repository, data filter, and visualization tool. The relationship between these elements is fairly straightforward and this section will describe the details of these components as raw data flows from the sensors through the system until it is finally visualized on a user's display. All of the components except for the data filter have been in use for some time within a variety of other research projects, so we will focus here on the specific features related to this component of the system.

All of the data used for this project is derived from sensor platforms installed within seventeen transit buses operated by UCLA's Fleet and Transit Services. The platform itself is essentially a small embedded computer with a variety of sensor input capabilities. This computer is connected to a wireless CDPD modem which provides continuous IP connectivity for the platform. While the sensor platform can receive data from a variety of sensor devices on board the buses, here we are concerned only with the location, speed and direction of the bus. For this project we are using a GPS sensor alone to locate bus positions. We are currently working to fuse additional sensor data available on the bus with the GPS for more accurate raw position reporting. For instance an electronic compass would provide more precise heading information while very accurate speed information can be derived from the bus's transmission. It is important to note that many other types of objects (people, cars, etc.) that we might be interested in visualizing may not have additional sensor data available. With this in mind, we explore the issues related to accurately and in an aesthetically acceptable manner, visualizing objects using GPS data.

Collecting position data from a GPS device is simple enough as the data is most often reported as a series of ASCII strings or sentences. These sentences follow a protocol and syntax defined by the NMEA-0183 standard[1] that, when parsed, provide the GPS device's solution to position, heading, and speed, among other things defined in the standard. As the GPS device continually produces position information at one second intervals, the sensor platform collects this data and converts it into a binary packet for reporting to the data repository.

Data packets are transmitted over a wireless CDPD[2] connection to a data repository which acts as a collection point for all of our sensor platform data. Unfortunately, maintaining any wireless connection is not always possible and during the construction of our testbed we have experienced problems with connectivity. The two most serious have been "dead spots" where the signal from our wireless provider is either very weak or non-existent and, second, having our data connections bumped in favor of voice traffic. Whatever the cause, we implemented a mechanism for buffering data on the sensor platform until the CDPD connection could be restored. Fortunately, in most instances the buses move quickly back into areas of connectivity and fill-in the repository with any missing data from their buffer. While not experienced during this project, another possible cause of missing data is loss of the GPS signal itself. When this happens the! sensor platform cannot report updates to its position at all. Here, again, is a reason to consider implementing additional sensor capabilities. With compass and speed information available a dead reckoning could be computed from the last known position in place of the GPS signal until it becomes available again.

As the data repository receives updates from the sensor platforms over the network they are stored in a database. While the database provides us with an archive of mobile sensor data for other projects as well. It also has a practical purpose for a visualization system like the one we are describing; it allows us to support a playback mechanism for user evaluation of real-time events. While a commander could clearly use a system like this for real-time coordination of units in the field, an off line playback capability allows for analysis, training, and debriefing. Another benefit of an intermediate repository is that applications can all use a common mechanism for accessing the data, whether real-time or archival. Real-time applications incur a small increase in latency since sensor platforms need to add data into the repository, and application clients must request and retrieve that data, but we believe that the overall benefit of this cost is worthwhile.

The data filter acts as a client to the repository to retrieve location information for each of the buses. The filter continually requests the most up-to-date raw position information for each of the buses currently being filtered. Filtering here implies a series of steps that are performed to interpret the raw GPS data in terms of knowledge of the the real world. For the data filter this knowledge is encoded in a data structure representing the location of the roads and the location of relevant entities along the roads such as intersections, crosswalks and bus stops. The data filter uses this information to first determine which road segment the bus is on and then through additional corrections, determines the bus's location along the road. Once the filter has made a determination of where it believes the bus really is, it can make some final aesthetic adjustments before sending commands to visualize the bus.

The visualization system we are using is the Urban Simulator, an application developed by the Urban Simulation Team at UCLA[3]. The Urban Simulator is capable of rendering in real-time very large urban environments. The environment we used for this project is a model of the UCLA campus and the surrounding parts of Los Angeles. The model itself is several hundred megabytes in size and roads, buildings, etc., are accurate to within about a foot in location and dimensions. It is within this virtual world that the buses are visualized using the position data provided by the data filter. The models of the buses themselves are loaded along with the static urban model, and their locations are updated by the data filter through an API that the Urban Simulator exposes through the network. The

[1] NMEA - National Marine Electronics Association

[2] CDPD - Cellular Digital Packet Data

[3] `http://www.ust.ucla.edu`

user navigating the environment sees the buses realistically moving about the model as they would in the real-world.

3 Algorithms and Optimizations

3.1 Filter Module: Smoothing Data

As mentioned above, the data filter processes the raw GPS data to attempt to map the raw data into a more accurate reconstruction of the actual movements of the vehicles. Knowledge of the roads on which the vehicles travel is used in conjunction with several assumptions regarding the nature of the vehicles' movements, e.g., that they travel on the correct side of the road. However, it is impossible to correct a single point obtained from the GPS device using only the above information; coordinates of the points before and after it are needed as well. For this reason we maintain a buffer of points, which introduces some latency between the time when a GPS reading is received and the time when we can inform the simulator of an accurate, filtered version of that point reading. We also want vehicles to be displayed along a smooth, continuous path, the final vehicle position is calculated as the interpolation of several consecutive points in the buffer[4]

When a new point is received from the vehicle, it is placed at the end of the buffer, and all other points in the buffer move one step closer to the front of the buffer. Thus, the oldest point in the buffer is at the front, while the newest point is at the end. As new points enter the buffer, more is learned about the movement of the vehicle, so the position of older points in the buffer can be refined. However, we do not want to change the position of any points that have been used in an interpolation to produce the vehicle's final position, since this could lead to discontinuities in the vehicle's movements. Therefore there is a window of interpolated points at the front of the buffer on which no further calculations are performed.

The first and most basic step in refining a raw GPS device reading is to snap it to the known system of roads on which the vehicles travel, where the network of roads is represented as a directed graph. However, simply snapping the GPS data to the closest point on the graph does not take advantage known characteristics of vehicle motion; namely that a vehicle's direction of motion is very close to the direction of the street on which they are traveling and that vehicles follow a continuous path along the street graph. Since we know both the motion and previous position of the vehicle from stored points in the buffer, we can use this information to more accurately determine to which road (that is, to which edge of the graph) a point should be snapped. Segments that have a direction close to that of the vehicle's motion are given preference, as well as are those that are close to the road on which the vehicle was previously traveling. Since these calculations only require knowledge of past points, they are performed when the point is first received from the GPS and placed at the end of the buffer.

Snapping points to the road is a method of correcting lateral errors in vehicle positions, but it can neither correct nor detect errors along the direction of the road. This does not create a serious problem until the bus turns, at which point the vehicle may either "overshoot" the turn if the GPS coordinates are ahead of the actual position of the vehicle in the direction of travel, or "undershoot" the turn if the GPS coordinates are behind the actual position. If the vehicle overshoots the turn, it will first appear to pass an intersection, then make a sharp turn and drive across a corner to one of the cross streets that it just passed (see Figure 2). If the vehicle undershoots a turn, it will start to turn before the actual intersection and "cut" the corner (see Figure 3. While both of these situations produce undesirable results, they require very different solutions.

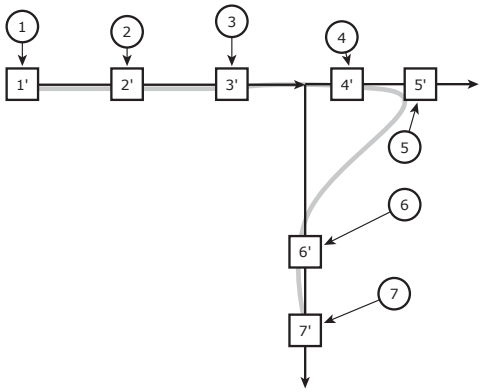

Figure 2: Overshoot Before Correction.

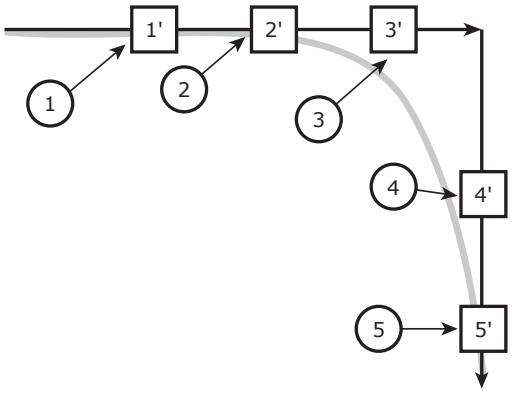

Figure 3: Undershoot Before Correction.

A point that overshoots an intersection cannot be detected by looking only at the earlier points stored in the buffer, Therefore it can not be resolved immediately when a point is first received. This is due to the fact that such a point will be encouraged not to turn by the snapping algorithm mentioned above. The line segment past the intersection has direction similar to the direction of motion of the vehicle up to this point, while the cross street is most often perpendicular to the motion of the vehicle. Since we cannot rely only on past data, we must correct over shoots based on new data that can alert us of the problem; namely points that are snapped to the cross street. We detect overshoots when as a point is about to move into the window of interpolated points at the front of the buffer, since this is when the most relevant data is available. The distance in terms of number of graph edges between the point being examined and points later in the buffer is computed. This distance is compared with points closer to the front of the buffer and the points later in the buffer. If the latter distance is smaller, then the point being examined is not along the path connecting points before it and points after it, so it should be corrected. It is snapped to the road using the snapping algorithm above, but this time only those segments between the older points and the newer points are considered (see Figure 4).

One large difference between over shoots and under shoots is that over shoots can be detected qualitatively, while under shoots

[4]Some computations on the raw GPS data are aesthetically neutral and some are made to provide a more realistic visual experience. Which is which is clear from the descriptions in this section. For some purposes, those calculations made for aesthetic reasons may not be appropriate, since these modifications to the data tend to eliminate abnormal conditions, e.g., when a vehicle leaves the road.

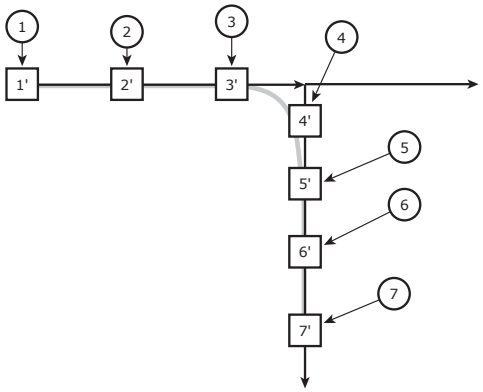

Figure 4: Overshoot After Correction.

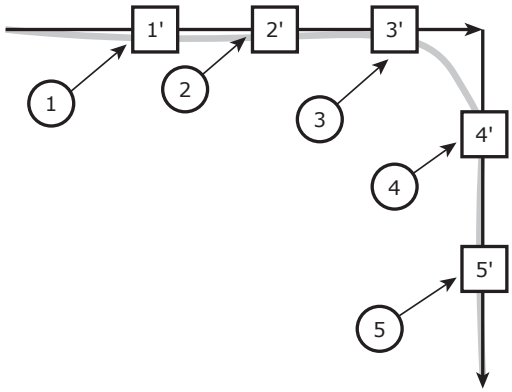

Figure 5: Undershoot After Correction.

are merely a quantitative problem. That is, we want to smooth the data to provide a smooth, round turn, but we do not want it to cut it so much that the vehicle appears to drive over the curb or through buildings. For this reason, instead of explicitly detecting when an under shoot has occurred, we try to resolve the main cause of under shoots in general, which requires the correction of errors parallel to the motion of the vehicle. Since errors in the GPS data are reasonably constant over time, we can use past errors as a good approximation of current errors. Thus, we can use the current lateral error as an approximation of what the parallel error will be after the vehicle makes a 90-degree turn. An average error vector is maintained, which is simply the average difference between the raw data and the calculated vehicle position for some past number of points. This vector is added to raw data before the original snap to the road is done, and the occurrence of under shoots is significantly reduced (see Figure 5).

One other anomaly that can be caused by even a slight parallel error is the appearance of vehicles stopping at inappropriate places, for example in the middle of an intersection or on top of a crosswalk. When the GPS data shows a vehicle stopped at one of these places, we can guess that most of the time the vehicle is actually stopped at the edge of the region, at a stop sign or stop light. While there are situations in which a vehicle could stop in one of these areas (e..g., when waiting to make a left turn) data for these situations looks much like parallel error, so we treat all cases as if the vehicle is stopped at the edge of the region.

Detection of this sort of anomaly requires the detection of two conditions: when the vehicle has come to a stop, and when the vehicle is in a region where it should not be stopped. Since we have knowledge of the streets on which the vehicle is traveling, we also keep a list of points on the street graph that correspond to points where a vehicle might stop, as well as the width of the associated region in which a vehicle should not stop. Detecting when a vehicle has stopped is a matter of comparing consecutive points in the buffer to see if they are below some minimum threshold distance. However, in order to correct the situation, we must be able to detect stops as soon as the vehicle enters the region, even if it does not stop for several more seconds. When a point enters the window of interpolated points in the buffer, it is compared to the list of stops to see if it is in a region in which it is normal to stop. If it is, the points later in the buffer are checked to see if the vehicle stops. If it does, all points between when the vehicle enters the region and when it stops are snapped to the edge of the region. When the vehicle starts moving again, the next few points are brought closer to the edge of the region to give the vehicle the appearance of a smooth start up, rather than jumping to the spot at which the GPS data showed it stopping in the first place.

3.2 Optimization of Communication

In the current testbed implementation we have at most 17 vehicles being tracked at any one time. With this small number of objects reporting their position once per second, communication requirements and processing at the filter do not stress the system. However, this naive approach of having all objects report continuously is not optimal since not all vehicles are within view all the time. In order to scale the system to situations where there are a large number of objects being tracked, we analyze how we might control reporting by vehicles to closely approximate the intervals when they are in view. Our objective is to have the position data for vehicles at the urban simulation in time to maintain a smooth, seamless view. To begin, we need some notation for critical system parameters. Let

- V = the maximum latency required for a vehicle to transmit a point, or a buffer of points, to the filter. This includes any processing that may be done at the vehicle between when a point is recorded and when the vehicle is ready to transmit that data.
- F = the maximum latency required by the filter. In order to calculate the final position for time t, the filter must have all points in the interval $[t,\ t + F]$.
- M = the maximum latency of a message passed from the filter to a vehicle. That is, a message sent by the filter at time t will be received by the vehicle at least by time $t + M$.
- S = the maximum latency of a point sent from the filter to the simulator. That is, a point sent by the filter at time t will be displayed on the screen at time $t + S$.
- A = the maximum latency required to send frustum information from the simulator to the filter.
- T = the maximum total latency of the system. That is, at time t, the vehicles are displayed at their positions as of $t - T$. Since a point must be transmitted by the vehicle, stored in the filter, and transmitted to the simulator, we can see that $T \geq V + F + S$.

Figure 6 shows the communication timeline under normal streaming, in which all vehicles are transmitting their positions once per second[5] For the rest of this discussion, let us say that the

[5]We use maximum latencies in all discussions since, to avoid jitter in the display, we need to allow for this maximum time. Of course data can be delivered early and delayed appropriately by the processing algorithms.

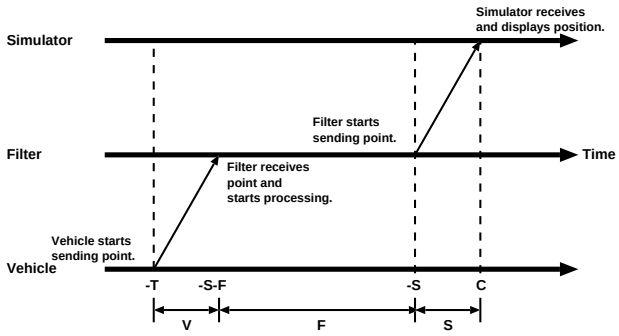

Figure 6: Continuous Streaming from Vehicles.

viewer is looking at a certain point on the screen at time $t = 0$. At time $t = -T$, the vehicle records this point and transmits it to the filter. At time $t = -T + V$, this point is received by the filter and stored in the filter's buffer. Over the interval $[-T+V, -T+V+F]$, the filter receives new points from the vehicle, stores them in its buffer and performs smoothing calculations on them. At time $t = -T + V + F$, the filter transmits the smoothed version of this point to the simulator and at time $t = -T + V + F + S$, or since $T = V + F + S$, time $t = 0$, this point is displayed on the screen to the user.

When choosing an appropriate communication strategy, we must examine certain constraints imposed by the system. In order to display a point on the screen at $t = 0$ (which corresponds to the vehicle's position at $t = -T$), we can see by working backwards that, regardless of the strategy used, the filter must send this point to the simulator no later than $t = -S$. Thus, assuming negligible computation time by the filter, the vehicle must transmit any data that the filter might need no later than $t = -S - V$. As stated above, in order to calculate the position for the point at $t = -T$, the filter needs all points from $t = -T$ to $t = -T + F$. Since $T + F = -S - V$, the vehicle will necessarily have all needed data by the time that it needs to transmit to the filter. This assumes that the vehicle has enough memory to store the data in this interval. Still working backwards, we see that the filter must have some idea of what will be visible at $t = 0$ no later than $t = -S - V - M$. At this point, there are two basic alternatives. The filter can either have some idea whether or not this vehicle will be visible at $t = 0$, and explicitly tell the vehicle to send its position if it will be visible, or the filter can transmit its estimation of the viewing frustum at $t = 0$, and let the vehicle decide if it should transmit its position.

We are interested in minimizing the network traffic by having vehicles stream their position data in intervals that closely approximate (but subsume) the intervals in which that data is required by the simulator. Thus if a vehicle is visible during interval (t_1, t_2), that vehicle needs to stream its position data in an interval that subsumes $(t_1 - T, t_2 - T + F)$. Note that on the trailing edge of the interval, an extra F seconds of data are required by the data filter smoothing algorithm. Here, in consideration of space limitations, we will be concerned with the leading edge of the intervals when a vehicle needs to stream position information. The trailing edge is less critical and also yields to similar analysis.

3.2.1 Vehicles Do Culling

For each vehicle to do culling, it must have some information to compute a bounding box (or sphere) of where the viewing frustum might be at a future time. To begin, we assume that the data filter periodically sends the vehicles viewing frustum and user motion data (velocity, acceleration) and perhaps other information useful in computing a relatively tight bounding box with period P.

Given the information in the last report from the data filter and the age of that report, let $MBF_V(LastReport, Age, T)$ be the minimum bounding box, at a vehicle, for the viewing frustum at time T into the future. Here we are not concerned with the exact nature of the information sent from the data filter or the algorithm for computing the MBF_V. We are interested in how to formulate the rules for using this information to ensure that all necessary data reaches the simulation in a timely fashion.

We assume that a vehicle knows its own position with no delay. Due to imprecision in the GPS data, its current raw position is expanded to a bounding box centered at that raw position. Let us denote this bounding box as MBL_V, which stands for Minimum Bounding Location at Vehicle. The vehicle should start sending data at a time t if there is a non-empty intersection of its current MBL_V and $MBF_V(LastReport, Age, +T)$.

A number of variations on this basic scheme can be considered. For example, if the data filter knows the prediction algorithm that the vehicles use, there can be a protocol in which, if the viewing frustum, deviates from the prediction, the data filter will send an earlier update. This allows for more aggressive assumptions to be used in the prediction in order to obtain tighter predictions since, if the assumptions are wrong, the data filter can send updated information that corrects the vehicle's predictions.

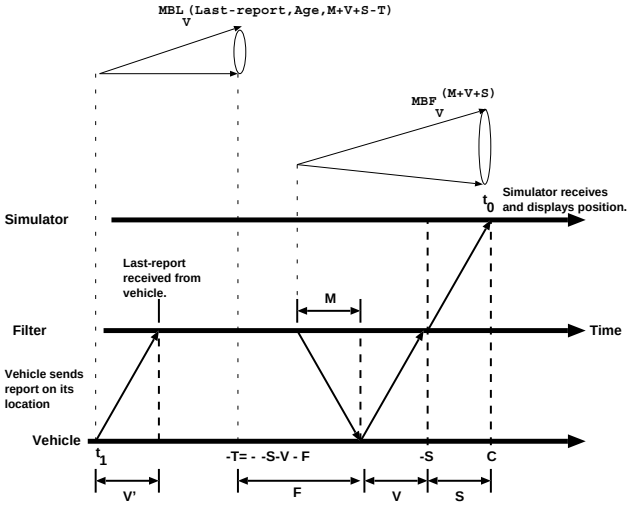

Figure 7: Filter Culling With Fast Communications.

3.2.2 Data Filter Does Culling.

In this scheme, the vehicles send location information to the data filter periodically. The data filter determines when a vehicle should start streaming position data and sends a "start" message to the vehicle in time for it to send the needed data. The timing in this case is a bit more complex. Referring to Figure 7, a vehicle sends its position information to the data filter at time t_1 and this is received at the data filter at least by time $t_1 + V'$ where V' is the maximum latency that this message will experience. If the data filter sends a "start" message to the vehicle at time t, that message is received as late as $t + M$ at the vehicle. The vehicle can then send its position

data which arrives at the data filter perhaps as late as $t + M + V$ and at the simulator at $t + M + V + S$.

From Figure 7 we see that for the vehicle position data to reach the simulator at time t_0, the vehicle must send the data by $t_0 - S - V$. Since the "start" message has a maximum latency of M, the data filter must send the "start" message by $t_0 - S - V - M$.

We recall that M is communication delay from the data filter to a vehicle and that F is a delay required by the data filter smoothing algorithms. Therefore these are not control variables that can be tuned to minimize communication bandwidth requirements. There are two cases to consider: $M < F$ and $F < M$.

Case $F > M$**.** This case is illustrated in Figure 7. Note that at $t_0 - S - V - M$ the data filter must know that it is possible that the vehicle comes into view at time t_0. The data filter knows the current viewing frustum but must predict the minimum bounding box containing the frustum an interval $M + V + S$ into the future. We denote this as $MBF_F(M + V + S)$. At the same time the data filter must create a minimum bounding box for where the vehicle might be at time $M + V + S - T$ into the future. This minimum bounding box we denote $MBL_V(LastReport, Age, M+V+S-T)$. We note that the last argument here can be negative. (This is not the case illustrated in Figure 7.) If this value is negative but less than Age in absolute value, it is just the distance into the future that the $LastReport$ data needs to be projected. The more interesting case is when it is negative and greater in absolute value than Age. In this case, the projection is into the past or, if the $LastReport$ contains information about where the vehicle has been (which it knows rather precisely) this not an issue of projection into the future but the report may be an approximation of the past trajectory. Note also that prediction is only completely avoided if $M+V+S-T+Age < 0$ at all times. If P is the period between vehicle reports, and if V is the communication delay, then to avoid all prediction of vehicle position requires $M + V + S - T + P + V \leq 0$ since this is the worst case.

Case $M > F$**.** This case is illustrated in Figure 8. In this case, since $M > F$, at $t_0 - S - V - M$ the data filter has to compute $MBF_F(M+V+S)$, i.e., a minimum bounding box at $M+V+S$ into the future, knowing the current viewing frustum and velocity, etc. It also needs to predict from the last report from each vehicle, $MBL_F(LastReport, Age, M + V + S - T)$. If the intersection of these two bounding boxes is non-empty, then a "start" message must be sent to the vehicle. (Note that $M + V + S - T$ is always positive if $T = S + V + F$, its minimum value, and $M > F$.)

From this analysis, it would appear that the vehicle culling solution is more efficient in most cases. Since it does not place any constraints on the relative latencies of the system, it is more widely applicable. It also distributes more work from the single point at the filter to the individual vehicles. However, in systems that meet the constraints for the centralized solution, and in which the vehicles may not have the needed computational abilities to decide when they are visible, the second solution may be the only choice.

4 Related Work

Our work combines aspects from previous research in two areas: the visualization of real-time information, and the culling of moving objects from a virtual environment. However, our goal of presenting the user with an accurate and realistic view of events and their surroundings in real time presented additional challenges. There has been a great deal of research done on systems to collect and display real-time information to a user in a convenient manner [4, 2, 3]. However, most of these systems have one or more of the following simplifying characteristics: the raw data that is being displayed is accurate enough that it can be usefully displayed to the user with little or no correction, or the user interface for the

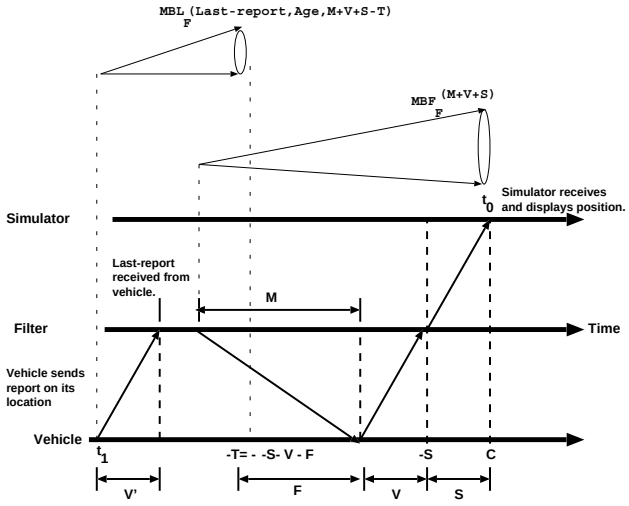

Figure 8: Filter Culling With Slower Communications.

system is abstract or coarse enough that the usefulness of the application is not affected by small errors in the data, e.g., a freeway map showing real-time travel speeds Our work differs in the fact that we display data that can have errors of up to 50 feet in a photorealistic environment where a user can detect errors of less than a few feet. While the idea of collecting complex data and showing it to a user in a way that can be easily understood and processed by the human mind has been explored previously, we have expanded greatly on ways of filtering information to remove errors introduced by the underlying hardware. The result is a system in which a user can visualize relatively accurate data in relation to a highly detailed environment. We have also explored and analyzed methods of culling unnecessary location data which would be required to scale the system several orders of magnitude.

Systems to display dynamic information to a user who can change his point of view freely are well documented [2]. It has been shown that in order for these systems to be scalable, objects that are not visible to the user should be culled in some manner so that intensive calculations are not done unnecessarily [1, 5]. However, most past work has dealt with objects whose motion was either prerecorded, generated by an internal motion model, or rendered at an abstraction level that eliminated the need for the level of accuracy needed for a realistic and accurate rendering. In our application, the data is near real-time, is relatively unpredictable, and requires accuracy and realism, Also, we have explored methods to distribute the job of culling to the distributed individual objects, while previous research has considered only performing this task at one central location.

5 Conclusions

As sensor technology advances, it will make much real-time information available. There are a number of technological challenges including the need to control access to much of this information to ensure privacy. Our purpose here was to explore algorithmic issues concerning visualization of sensor data (in our particular case, tracking information) in terms of (a) smoothing or cleaning of noisy data and (b) optimization of data communication based on culling of location information for vehicles that are not visible. Future work would include reduced precision in tracking objects that are

visible but at a far distance from the viewer. Another issue to be studied is scalability when the number of objects grows. The challenge is that the on-board computers of the objects should perform much of the work so that a bottleneck is avoided. This is particularly difficult in that the objects are moving (as well as the viewing frustum) and associations based on proximity are continually changing. Future work should also include consideration of multiple simultaneous users with different viewing frustums. This then becomes an "N-to-M" problem with N consumers (viewers) and M sources (vehicles) where the subset of sources feeding a destination is continuously changing and potentially overlapping with the subset of sources feeding other consumers.

REFERENCES

[1] Chenney and Forsyth. View-dependent culling of dynamic systems in virtual environments. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, April 1997.

[2] Hu, Mehrotra, Winkler, Ho, Gregory, and Allen. Integration of saturn system and VGIS. In *Proceedings of ARL Federated Laboratory Advanced Displays and Interactive Displays Consortium*, February 1999.

[3] Kawasaki, Murao, Ikeuchi, and Sakauchi. Enhanced navigation system with real images and real-time information. In *Proceedings of 8th World Congress on Intelligent Transport Systems*, October 2001.

[4] Long, Mantey, Rosen, Wittenbrink, and Gritton. REINAS: A real-time system for managing environmental data. In *Proceedings of Conference on Software Engineering and Knowledge Engineering*, June 1996.

[5] Shou, Chionh, Huang, Ruan, and Tan. Walking through a very large virtual environment in real-time. In *Proceedings of Very Large Databases*, September 2001.