Adaptive Scene Synchronization for Virtual and Mixed Reality Environments

Felix G. Hamza-Lup[1] and Jannick P. Rolland[1,2]
[1]School of Electrical Engineering and Computer Science
[2]School of Optics-CREOL
University of Central Florida
fhamza@cs.ucf.edu, jannick@odalab.ucf.edu

Abstract

Technological advances in virtual environments facilitate the creation of distributed collaborative environments, in which the distribution of three-dimensional content at remote locations allows efficient and effective communication of ideas. One of the challenges in distributed shared environments is maintaining a consistent view of the shared information, in the presence of inevitable network delays and variable bandwidth. A consistent view in a shared 3D scene may significantly increase the sense of presence among participants and improve their interactivity. This paper introduces an adaptive scene synchronization algorithm and a framework for integration of the algorithm in a distributed real-time virtual environment. In spite of significant network delays, results show that objects can be synchronous in their viewpoint at multiple remotely located sites. Furthermore residual asynchronicity is quantified as a function of network delays and scalability.

1. Introduction

Advances in optical projection and computer graphics allow participants in virtual environments to span the virtuality continuum from real worlds to entirely computer generated environments, with the opportunity to also augment their reality with computer generated three-dimensional objects [1][2]. These objects can be created in real-time using dynamic texture projection techniques on refined 3D models [3]. In the case of remotely located participants, the distribution of three-dimensional objects allows efficient communication of ideas through three-dimensional stereo images that may be viewed in either a mixed reality or an immersive reality configuration. When designing a distributed application that takes advantages of virtual and mixed reality, large amounts of data may need to be distributed among the remote sites. This distribution must occur in real-time in order to ensure interactivity. For an effective collaboration, all the participants must be able to see the effects of the interaction at the same time. Every time the scene changes, new objects appear, or objects change their position and/or orientation, all participants must perceive these changes simultaneously, i.e. the dynamic shared state has to be consistent for all the participants.

This paper presents a scene synchronization algorithm that will compensate for the network latency, ensuring that all the participants to a distributed stereoscopic visualization session see the same pose (i.e. position and orientation) for the virtual objects in the scene. The algorithm ensures optimal synchronization of the scenes, by adapting to the variations in the network delays among the participating nodes.

The paper is structured as follows. Section 2 discusses related work. Section 3 describes the adaptive scene synchronization algorithm employed to compensate for the network delays. Section 4 presents the integration of the algorithm in a framework and a method for synchronization assessment. Section 5 focuses on the experimental results. Finally, Section 6 concludes the paper and identifies areas of future research.

2. Related work

From the distributed systems perspective, research in synchronization has been focused on time synchronization. The NTP [4] (Network Time Protocol) represents a way to keep the clocks of several nodes across the Internet synchronized. Miniaturization and low-cost design has led to active research in synchronization in large scale sensor networks [5].

Synchronization is a critical paradigm for any distributed virtual or mixed reality collaborative environment. Maintaining the consistency of the dynamic shared state in such an environment is one of the most difficult tasks. Previous work points to the need of synchronizing shared viewpoints. One of the first and most intensive efforts in building a networked simulation was the SIMNET project started in 1983 followed by Naval Postgraduate School's NPSNet [6] a few years later. Dead reckoning algorithms were employed to maintain a fairly consistent shared state. In some recent

IEEE Virtual Reality 2004 March 27-31, Chicago, IL USA
0-7803-8415-6/04/$20.00©2004 IEEE.

work, Schmalstieg and Hesina (2002) presented Studierstube, which uses a distributed shared scene graph to keep track of the actions applied on the shared scene by multiple participants [7]. The authors show that multiple concurrent operations on the objects in the scene may lead to inconsistent views. As communication delays increase, the inconsistency among remote participants grows even further. Therefore, synchronization is a key factor for maintaining a consistent view among the participants to a distributed mixed reality application.

There are several factors that affect the synchronicity of a distributed virtual reality (VR) or mixed reality (MR) system, including network delays and variable bandwidth. Various distributed VR systems address these two factors in different ways. In the DEVA3 VR system [8], each entity is composed of a single "object" that represents what the entity does, and a set of "subjects" that represents how the entity looks, sounds, and feels. The object position is updated only when the subject has moved a certain distance from its previously synchronized location. Another synchronization approach is available in the MASSIVE-3 [9], a predecessor of the HIVE VR system. The updates in MASSIVE-3 are effective combinations of centralized updates and ownership transfers. In this approach, where the updates are centralized, the problem is system scalability. Other factors that affect the synchronicity of a distributed VR or MR system, besides the network delays, are differences in the hardware architectures over the system's nodes, hardware buffering, and software system delays [10]. Assuming similar rendering hardware for the system nodes, the most relevant factor is the network latency. The increase of LAN bandwidth and the decrease of queuing time facilitate the design of efficient synchronization algorithms that take into account the network characteristics. To the best of the authors' knowledge, algorithms for dynamic shared state maintenance in a virtual or a MR environment that take into account the network characteristics, have not been investigated.

3. Preserving the consistency of the dynamic shared state

The first step in a distributed VR or MR application is to ensure that each participant has the appropriate resources to render the virtual components of the scene. If the distributed application must ensure a real-time behavior, the appropriate resources must be available at specific time instances.

The virtual 3D objects in the scene usually have a polygonal representation. This representation allows for quick rendering; however, the polygonal representation might require significant storage space. Distributing these 3D objects, in real-time, on a local area network to a large number of clients is not possible. Our approach is to asynchronously download these models locally at each node before the interactive simulation starts.

Section 3.1 categorizes VR and MR applications based on the update frequencies, while the adaptive scene synchronization algorithm is introduced in Section 3.2.

3.1. Continuous vs. Discrete updates

Data flow in a distributed system, can be categorized as continuous or discrete. A discrete flow means that there are time intervals when the network infrastructure is not used. Similarly, when a VR or MR scene is modified, the modifications of the virtual components of the scene might have a continuous or a discrete pattern. For example, consider a scene that contains a virtual object, and whose position and orientation is given by a tracking system with a refresh rate of 120 Hz. An event-based mechanism will not fit this application since it would trigger an event every 8.33ms (1000/120). A distributed system built to fit such a model would have to continuously broadcast the tracking data to all its nodes.

On the other hand, if we consider that a participant changes the position and orientation of the object from a graphical user interface using a mouse, the participant is going to perform a sequence of actions (rotations and translations) on the object with a much slower rate. The *fastest* human-computer response time includes perceptual, conceptual, and motor cycle times, which add up to an average of about 240ms [11]. Moreover, some actions will generate continuous predictive movements. For example, the participant might spin the object for an indefinite period of time with a specific velocity around a specific axis. A distributed system built to fit such a model would have to discretely distribute the participant's actions to all its nodes. The event-based approach is more feasible in this case. The scalability of the system is also improved, because the system nodes use the network only when updates on the shared scene are necessary.

3.2. Adaptive scene synchronization algorithm

The adaptive scene synchronization algorithm assumes an event-based mechanism, triggered either by the participant actions on the shared scene or by a sensor (e.g. a tracking system) whose update cycle time is comparable or higher than the network latency. Such assumption is generally true as 100 Mbs local area networks (LANs) and optical routing are becoming increasingly available, decreasing delays and increasing bandwidth.

To control the position and orientation of the objects in the shared scene, each 3D object has a control packet associated with it. The control packet contains information about the position and orientation of the object, as well as information regarding the actions associated with each object: rotation, translation or scaling. The small size of the control packet (i.e. several KB) ensures a very low propagation delay, which allows the development of scalable, distributed real-time applications on local area networks. As the control packets flow through the LAN, the adaptive scene synchronization algorithm uses their information to synchronize the shared scene among different participants. The information carried by the control packets is distributed to each participating node allowing them to compensate for the network delays. These delays are called here *drift factors* since they cause a position/orientation drift of the virtual objects seen by the remote participants.

From the system architecture point of view we assume a client-server design. The node running the server process has the capability to interact with the virtual objects in the scene. The server acts as a data provider by pushing scene updates to the clients whenever the scene changes. The client nodes pull information from the server about their individual communication delays. The determination of the time intervals at which the measurements are triggered is described in Section 3.3.

The interaction with the virtual components of the scene is done through a graphical user interface (GUI). Commonly the interaction with the mixed reality scene leads to position, orientation and scale changes of the virtual components, assuming the components are representations of rigid objects. For this reason we have designed a GUI which allows the participant to change the virtual object's position and/or orientation with a specific velocity as seen in Fig.1. Each new event triggered from the graphical user interface that changes the object position or orientation is considered an action applied on that object. Each action has a velocity associated with it.

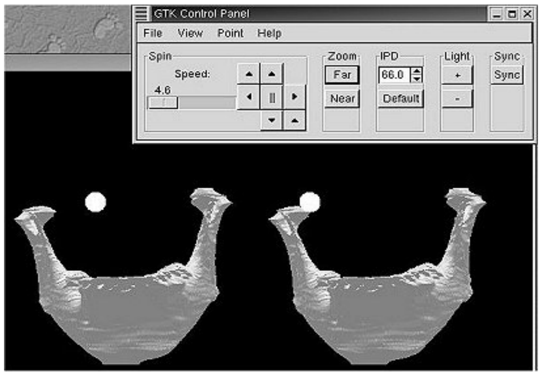

Fig. 1 Graphical User Interface. The white sphere represents a mouse driven 3D pointer. All the participants are able to see the pointer as the participant running the server process points to different locations in the 3D scene.

For simulation purposes, we ignore the acceleration component. From the computational complexity point of view, second-order polynomials that include object acceleration will not affect the computation speed, however higher order polynomials might delay the entire system. Moreover, when object acceleration changes frequently, it is better to ignore the acceleration estimate than to produce an inaccurate prediction before the next update is received.

Let's define the *drift value* for a particular object *i* present in the shared scene and a particular client node *j* as the product between the action velocity for the object and the network delay from the server to that node. If we denote M_τ as the number of virtual objects in a shared scene of N_τ participating nodes, both at a given time τ, a drift matrix $D(M_\tau N_\tau)$ associated with the distributed system at a particular time τ may be defined as:

$$D(M_\tau, N_\tau) = S \cdot T^t \qquad (1)$$

where S and T are both column vectors, S containing the action velocities for each object currently in the shared scene, and T the network delays from each participating client node to the server. T^t represents the transpose of T. The action velocity is extracted from each object's control packet, while the network delay is measured by each client node using an adaptive probe that computes the round trip time from the node to the server. S is stored locally at each node and updated when the scene changes.

A decentralized computational approach strips the drift matrix in N column vectors, called *drift vectors*, which contain the drift values of all the objects in the scene for a particular node. The drift vectors are updated when a new 3D object is inserted or removed from the shared scene by adding or respectively removing the entry associated with the new object from all nodes. The drift vectors are also updated when the participants perform actions on the objects in the shared scene. Whenever an action is applied to an object (e.g. a rotation), a control packet associated with that object is broadcasted to all the nodes. The information from the control packets is the first component used for synchronization. The second component accounts for the packet propagation and packet queuing delays. At specific intervals, each node "pings" the server to estimate an average network delay and computes the drift vectors associated with the objects in the scene as the product between the propagation delay and the objects' actions velocities. Each delay measurement between a node and the server triggers the node's *drift vector* update.

A sketch of the Adaptive Scene Synchronization algorithm is now described. The *ComputeNodeDelay()* function returns the delay associated with the connection between a specific node and the server. The *UpdateDrift()* function updates the drift values for the objects in the scene on each node. Three Boolean variables are used: *changedScene* that accounts for the changes in the scene, *newClientRequest* which is set if a new client has joined, and *trigger*, used in tracking the network behavior as described in Section 3.3. Finally the functions *ReceiveChanges()* and *BroadcastChanges()* ensure correct scene updates among the nodes of the system and the server. Each node's scene is synchronized with the server scene. Hence, a consistent dynamic shared state is maintained over all the participants.

```
Algorithm: Adaptive Scene Synchronization
Output: Synchronized shared scenes for a distributed
interactive VR/MR application.
Client side:
  Initialization:
    $T_n \leftarrow$ ComputeNodeDelay()
    $S_n \leftarrow$ UpdateAction();
    $D_n \leftarrow$ UpdateDrift()
    UpdateLocalScene();
  Main:
    if (trigger)
      $T_n \leftarrow$ ComputeNodeDelay()
      $D_n \leftarrow$ UpdateDrift()
    end if
    if (changedScene)
      $S_n \leftarrow$ ReceiveChanges()
      $D_n \leftarrow$ UpdateDrift()
    end if
Server side:
  for ever listen
    if (newClientRequest)
      SendToClient($S_n$);
    end if
    if (changedScene)
      BroadcastChanges();
    end if
  end for
```

3.3. Fixed threshold vs. adaptive threshold

As a result of the network jitter, the round trip times among different nodes vary. To achieve the best synchronization possible among collaborating nodes, delay measurements must be triggered whenever significant variations appear. These data are necessary to obtain an estimate of the average delay for each node (i.e. participant) that joins the distributed application. An average round trip time can be obtained by sending "ping" messages to the newly arrived node when it joins the group. Half of this delay represents an average delay from the node to the server.

The adaptive synchronization algorithm uses two approaches to trigger the information collection. In the first approach, at regular time intervals, using ICMP, a node opens a raw socket and measures the round trip time to the server. We call this the "*fixed threshold*" approach. However, gathering all this data implies additional delays at the client side and additional network traffic. The time intervals at which these measurements are triggered impact the real-time behavior and the scalability of the algorithm.

An alternative approach consists of adaptively triggering the round trip measurements for each node, based on the delay history, which better characterizes the network traffic and the application. In this approach, a fixed threshold is initially used at each node to build the delay history denoted H_p. The delay history is a sequence of p delay measurements h_i where $i=1,p$ (e.g. in the implementation we have chosen p to be 100).

Let σ and h_{mean} be the standard deviation and the mean of H_p , respectively. Let h_0 be the most recent delay, i.e. the last number in the H_p sequence, and γ_0 the current frequency of delay measurements, expressed as the number of measurements per second. The adaptive strategy is to decrease γ_0 by 1 unit if $h_0 \in [\, h_{mean} - \sigma ,\ h_{mean} + \sigma]$ and to increase γ_0 by 1 if h_0 does not belong to this interval.

4. DARE

The adaptive scene synchronization algorithm has been embedded in DARE, a Distributed Artificial Reality Environment, which is developed at the ODALab (*http://odalab.creol.ucf.edu/dare*). DARE [12] is a framework which uses virtual environments and distributed systems paradigms to improve human-to-human interaction enhancing the real scene that a person sees with 3D computer generated objects. Applications built using this framework range from distributed scientific visualization to interactive distributed simulations and span the entire virtuality continuum [13].

4.1. System components

The first collaborative environment that we have developed based on DARE consists of several sites located on a local area network. From the hardware point of view each site consists of at least one head-mounted display [14], a Linux based PC and a quasi-cylindrical room, called Artificial Reality Center (ARC) [2].

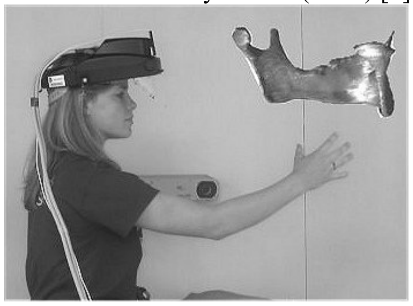

Fig. 2 Remote participant and a 3D virtual jawbone

As participants wearing head-mounted displays enter the ARC, they gradually start immersing themselves in virtuality. Initially, the scene is augmented with floating objects as seen in Fig.2.

These virtual objects augment the participant's reality and they may appear to multiple participants wearing head-mounted displays as illustrated in Fig.3.

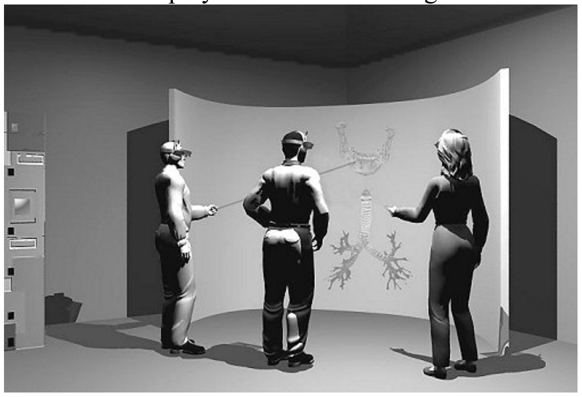

Fig. 3 Local collaboration

Participants can also interact with the 3D models. Using a three-dimensional graphical user interface they can point in the virtual space to different parts of the virtual objects and they can manipulate them.

Several ARC rooms can be interconnected on a LAN allowing remote stereoscopic visualization. People located in these rooms, as shown in Fig.4, can visualize and manipulate virtual objects from a shared scene.

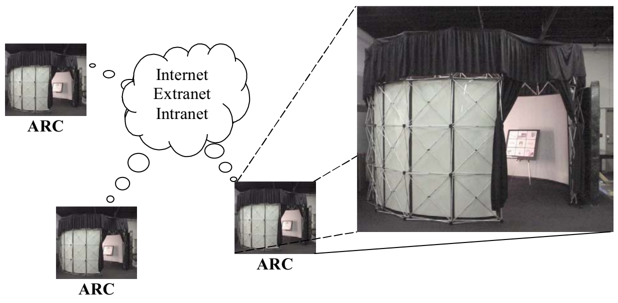

Fig. 4 NOE's ARCs (Networked Open Environments Artificial Reality Centers)

4.2. Method for synchronization assessment

To assess the efficiency of the synchronization algorithm we measured the amount of drift between the orientations of a 3D object at remote sites. The measurements were done pair-wise between the server node and each of the client nodes. Each node pair shared the same virtual 3D scene; one acted as a server and the other as a client. As described in Section 3.2, a GUI was available at the server site, which allowed the participant at that node to change the object position applying rotations around the Cartesian axis. This participant generated events from the GUI, and each time an event was generated, the position/orientation of the virtual object is recorded at every participating node. Because of the network latency, different vectors at each node described the orientation of the object. The rotations can be easily expressed using the quaternion notation.

Let q_s express the rotation of an object at one node (e.g. server node) and let q_c express the rotation of the same object at another node (e.g. client node). Both nodes render the same virtual scene and the displayed object should have exactly the same position and orientation. To quantify the difference between the orientations of the object on two different nodes we can compute the correction quaternion q_E between the nodes every time the participant triggers a new event. The correction can be expressed as follows:

$$q_s = q_E q_c \qquad (2)$$

And thus

$$q_E = q_s q_c^{-1} \qquad (3)$$

The quaternion q_E may be further expressed as

$$q_E = (\omega_E, \vec{v}_E) = (\cos(\frac{\alpha}{2}), \sin(\frac{\alpha}{2})(x\hat{i}, y\hat{j}, z\hat{k})) \quad (4)$$

where

$$\alpha = 2\cos^{-1}(\omega_E) \qquad (5).$$

The angle α represents the drift between the orientations of a 3D object as seen by the two nodes.

5. Experimental setup and results

To evaluate the performance of the algorithm, we first calculated the network latency using a latency measurement probe on a 100 Mbps LAN. The average round trip time for this setting was 1.5 ms.

To investigate the effect of the network latency, given that the drift value for an object is the product between the action velocity and the network latency, as defined in Section 3.2, we repeated the experiments at different action velocities.

To prove the scalability of the system, regarding the number of participants, two sets of experiments were performed. The first set contained two nodes, one acting as a client, the other one as a server. The second set contained 5 nodes, one acting as a server and the other 4 as clients.

5.1. Two nodes setup: network latency analysis

Running the distributed visualization with and without the synchronization algorithm, we can assess the effectiveness of the algorithm. Fig.5 provides a plot of the drift angle (α) for various action velocities before synchronization. The actions in this case are random rotations of a virtual object around its coordinate axis with the angular velocity of: 10, 50 and 100 degrees per second. The plot shows that as the action velocities increase, the drift also increases as expected and the magnitude of the drift reaches after 24 actions over 140 degrees for high action velocities. Overall, the drift increases in time as more and more actions are applied on the object. The sudden drops in the drift are caused by the compensating factor of the random rotations (e.g. clockwise followed by counterclockwise rotations of the object around the same axis). The drifts created will compensate each other to some extent.

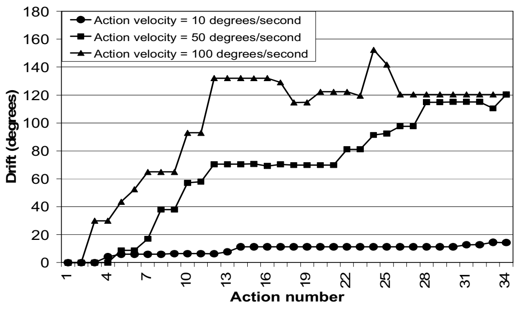

Fig. 5 The angular drift (α) without synchronization for different angular velocities

The synchronization module activation causes a significant decrease in the drift as shown in Fig.6.

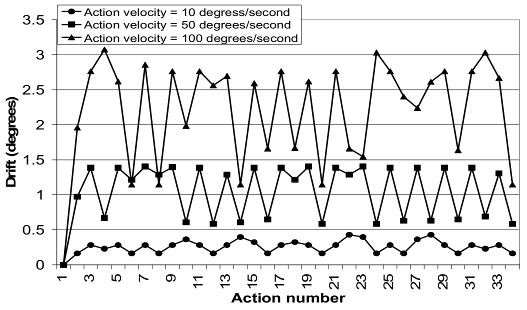

Fig. 6 The angular drift (α) with synchronization for different angular velocities

As the action velocity is increased, the drift oscillation amplitude also increases, however the drift value is maintained at an average of 2.4 degrees in the worst case, when the action velocity is 100 degrees per second. Moreover the average drift value has almost a constant value during the simulation.

Increasing the action velocity to 100 degrees per second on a network having 1.5 ms latency would be equivalent, in terms of drift magnitude, to running the simulation on a network having 15 ms latency using action velocities of 10 degrees per second. This proves that the synchronization algorithm is effective at maintaining the dynamic shared state of a distributed VR or MR application over nodes separated by network latencies of 15 ms.

The adaptive approach for triggering the network delay measurements, described in Section 3.3, has a positive impact on the scalability of the applications deployed on a stable network infrastructure. On the other hand, if the latency of the network infrastructure varies, the frequency of measurements triggered by each client node increases. If the number of participants also increases, the server might become ping flooded. Strategies in the category of ping flood protection might be employed in this case, which will limit the number of participants in the mixed reality collaborative simulation.

5.2. Five nodes setup: scalability analysis

To test the scalability of the algorithm, a five nodes setup was tested. This setup allowed 5 remote participants to be part of the distributed interactive simulation. One of the nodes runs the server process and the participant on this node is able to change the position and orientation of the virtual objects in the scene. In the current implementation, the other 4 participants do not interact with the scene. They are only able to visualize the virtual scene. During the simulation we monitor the orientation of one virtual object while the participant on the server node applies rotations on it with different speeds. The other four nodes run client processes and they are able to visualize the same virtual scene. Every new event generated from the server node triggers an orientation update on the virtual object on each node. At the same time, the current orientation is recorded in a file on each node.

From the hardware point of view, the nodes are heterogeneous. The network cards on all nodes support 100Mbps connections. Below is a table containing a brief specification of each node's hardware components.

Table. 1 Hardware systems attributes

Node no.	Arch	CPU (GHz)	RAM (MB)	Video card (GeForce)
1	Desktop	1.5 AMD	1024	4 Ti4600
2	Desktop	1 P3	1024	2 Mx
3	Desktop	1.7 P4	512	4 Mx 440
4	Desktop	1.7 AMD	1024	4 Ti4600
5	Laptop	2 P4	1024	4 Go440

In the first stage, the simulation was run without synchronization and at different action velocities. Fig.7 presents a plot of the angular drifts for different speeds for each client node. The legend for Node 2 applies to Node 3, 4 and 5. Node 1 is acting as a server and was used as a reference for the drift computation.

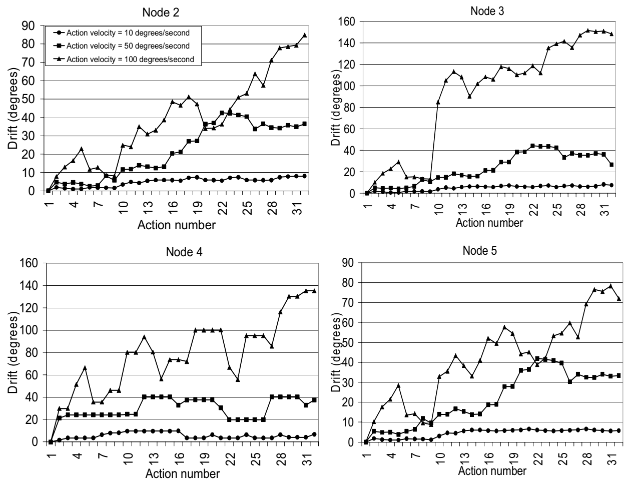

Fig. 7 The angular drift (α) without synchronization for different angular velocities on different nodes

As in the first set of experiments, the results show that the drift increases as the action velocity increases. The drift variation over different nodes is caused by the hardware heterogeneity of the nodes.

The second stage of the simulation was executed with the synchronization module active and at different action velocities. As the action velocity increases, it negatively affects the drift correction, however in all cases the average drift angle at 100 degree/second action velocity does not exceed 3.5 degrees, and over all the nodes the drift average is 2.9 degrees.

Fig.8 illustrates the drift variations over different nodes with the synchronization module active. The legend for Node 2 applies to Node 3, 4 and 5. Node 1 is acting as a server and was used as a reference for the drift computation.

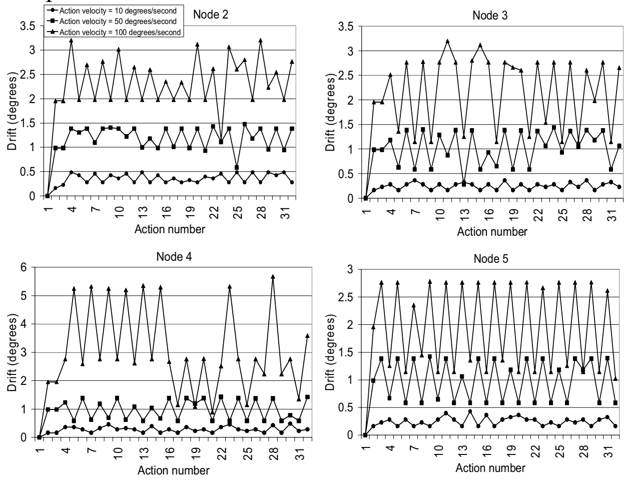

Fig. 8 The angular drift (α) with synchronization for different angular velocities on different nodes

The current client-server architecture on which the algorithm was deployed seems to introduce the disadvantage of a centralized approach. Scale is clearly limited by the capacity of the server, and centralized systems are often thought of as having a low degree of scalability. However, in our approach the majority of the computation is distributed among participating nodes. Each node renders its own scene and computes its own drift value. The only burden on the server node, which increases with the number of nodes, is the reply to each delay measurement message sent by a client node.

We can define a metric analyzing the relationship between the number of nodes in the system and the drift values. Provided that the algorithm is activated, let ψ_i be the average drift value over all the nodes, when i+1 nodes are in the system and the action velocity is set to 100 degrees per second, for example. In the case of a two nodes setup results show that the average drift is $\psi_1 = 2.4$ degrees while in the case of 5 nodes setup the average drift is $\psi_4 = 2.9$ degrees. An algorithm with low degree of scalability would have at least a linear increase in drift, i.e. $\psi_n = n\,\psi_1$, while a high degree of scalability would mean $\psi_n \approx \psi_1$. Using this metric in the 5 nodes setup, a low degree of scalability would translate to $\psi_4 = 4*\psi_1 = 9.6$ degrees. The experimental results show that $\psi_4 \approx \psi_1$. The algorithm scales well with the number of participants

6. Conclusion and future work

We have presented an adaptive synchronization algorithm that addresses the impact of network latency on shared scenes in distributed mixed and virtual reality applications. The fundamental property of our design is that the algorithm takes into account the network latency. Moreover, by taking into account the measurement history of the end-to-end network delays between the nodes and the server, the network jitter is taken into consideration. The decentralized computation approach for the drift values, carried out independently at each node, improves the system's scalability and its real-time behavior.

The proposed algorithm works under the assumption that the system is driven by events generated by a human actor or a sensor with low update frequency. As the widespread use of high speed optical networks and optical routing becomes increasingly common, the approach presented will be widely applicable. Future work will involve testing the current algorithm when high frequency update sensors are connected to the system (e.g. tracking systems and haptic devices).

If the network latency is high, disconcerting jumps in the object position and orientation will occur at some nodes. We are investigating the possibility of eliminating these jumps using interpolation.

7. Acknowledgements

We wish to thank our sponsors NSF/ITR: IIS-00-820-16, the Link Foundation, and the US Army Simulation, Training, and Instrumentation Command (STRICOM) for their invaluable support for this research. We also thank Eric Clarkson for stimulating discussions about quaternions, and Charles Hughes for helpful comments about the paper.

8. References

[1] Billinghurst, M., Kato, H., Kiyokawa, K., Belcher, D., and Popyrev, I. "Experiments with Face to Face Collaborative AR Interfaces", Virtual Reality Journal, 4(2), 2002.

[2] Davis, L, Rolland, J., Hamza-Lup, F., Ha, Y., Norfleet, J., Pettitt, B., and Imielinska, C., "Enabling a Continuum of Virtual Environment Experiences", IEEE Computer Graphics & Applications, 23(2) , 2003, pp.10-12.

[3] Neumann, U., You, S., Hu, J., Jiang, B., and Lee, J.W., "Augmented Virtual Environments (AVE): Dynamic Fusion of Imagery and 3D Models", IEEE Virtual Reality 2003, Los Angeles, 2003, pp. 61-67.

[4] Mills, D., "Internet Time Synchronization: The Network Time Protocol." in Global States and Time, in Distributed Systems, IEEE Computer Society Press, 1994.

[5] Elson, J., Girod, L., and Estrin, D., "Fine-Grained Network Time Synchronization using Reference Broadcasts", Fifth Symposium on Operating Systems Design and Imp. (OSDI), 2002.

[6] Macedonia, M., Zyda, M., Pratt, D., Barham, P., and Zeswitz, S., "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments", PRESENCE, 3(4), 1994, pp.265-287.

[7] Schmalstieg, D., and Hesina, G., "Distributed Applications for Collaborative Augmented Reality", Proceedings of IEEE Virtual Reality 2002, Orlando, Florida, March 24-28, 2002, pp. 59-66.

[8] Pettifer, S., Cook, J., Marsh, J., and West, A., "DEVA3: Architecture for a Large-Scale Distributed Virtual Reality System", ACM Virtual Reality Software and Technology, 2000, pp.33-40.

[9] Greenhalgh, C., Purbrick, J., and Snowdon, D., "Inside MASSIVE3: Flexible Support for Data Consistency and World Structuring", ACM Collaborative Virtual Environments, 2000, pp. 119-127

[10] Swindells, C., Dill, J., and Booth, K., "System Lag Tests for Augmented and Virtual Environments", Proceedings of the 13th annual ACM symposium on User Interface Software and Technology, 2000, pp. 161-170.

[11] Eberts & Eberts, "Intelligent interfaces: theory, research, and design", Hancock and Chignell (eds.), North Holland, Elsevier Science Publishers, 1989, pp. 69-127.

[12] Hamza-Lup, F., Davis, L., Rolland, J., and Hughes, C., "Where Digital meets Physical – Distributed Augmented Reality Environments", ACM Crossroads (online), 9.3, 2003.

[13] Milgram, P., Kishino, F., "A Taxonomy of Mixed Reality Visual Displays", IECE Trans. Information and Systems (Special Issue on Networked Reality), E77-D (12), 1994, pp. 1321-1329.

[14] Hua, H., Ha, Y., and Rolland, J.P., "Design of an ultra-light and compact projection lens", Applied Optics 42(1), 2003, pp.97-107.