University of Cincinnati

Date: 3/17/2011

I, Chandra K Kuchi, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
Implementing Space and Time Non-linearity in Virtual Worlds

Student's name: Chandra K Kuchi

This work and its defense approved by:

Committee chair: Philip Wilsey, PhD

Committee member: Wen Ben Jone, PhD

Committee member: Carla Purdy, C, PhD
Implementing Space and Time

Non-linearity in Virtual Worlds

A dissertation submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the School of
Electric and Computing Systems
of the College of Engineering and Applied Science

Mar 17, 2011

by

Chandra K Kuchi
BE, National Institute of Technology Durgapur, 2006

Thesis Advisor and Committee Chair: Dr. Philip A. Wilsey
Abstract

3D virtual worlds provide rich environments for collaborative work and social networking. The design of these spaces is has largely been designed and limited by the classical linear time and space properties that we find in our own physical world. In some cases, some non-physical capabilities like teleportation are provided, but in general, most of the limits that we experience physically such as time linearity, gravity, orientation, and single location, are also experienced in these computer-based virtual worlds. This thesis, explores relaxing the linear constraints of time and space in a 3D virtual world. In particular, the thesis explore extensions that have been made to a Second Life (a 3D virtual world) viewer to bend the time and space barriers from conventional limits. The modifications are inspired by the concepts of picture-in-picture and DVR in modern television viewing systems. These extensions are called: multi-view which allows the viewer to monitor multiple locations in the virtual world (allowing a user to observe and switch quickly between multiple virtual locations), and pause-view which allows the user to pause and then continue (possibly in faster than real-time) a meeting stream (much like the pausing of a TV program made possible by digital video recorders). Providing faster than real-time playback allows the viewer to catch up (from a pause or late meeting arrival) so that they can, for example, participate in a closing Q&A session. The technical and functional modifications, benefits with emphasis on architectural details are described. Evaluation results of these new User Interface design are presented. Exciting possibilities for second life community due to the introduction of these new dimensions are discussed.
Acknowledgments

I would like to thank my adviser, Dr. Philip A. Wilsey from the bottom of my heart for his continued support throughout my thesis work. Without his support it would have been difficult to complete my work. It was not easy walk through the project. His moral support when things are not going as expected was beyond my expectations. His vision helped me to solve problems with relative ease. Beyond everything, his confidence and support towards his students is exemplary.

I would like to thank other members of my defense committee: Dr. Wen-Ben Jone and Dr. Carla C Purdy for their feedback to give better shape to my thesis document and for taking time to serve on the committee.

I would also like to thank my family and friends for their words of encouragement throughout my master’s work. I would like to thank my friends Narsimha, Rahul and Veera for encouraging me to pursue Graduate studies.
## Contents

1 Introduction .................................................. 1
   1.1 Hypothesis & Motivation ................................. 2
   1.2 Overview of this thesis ................................. 3

2 Background & Related Work ................................. 4

3 Multi-view and Pause-view ................................. 7
   3.1 Multi-view .............................................. 8
   3.2 Pause-view ............................................. 12

4 Multi-View Implementation & Examples .................. 17

5 Pause-view Implementation & Examples ................. 25

6 Analysis of Extensions ....................................... 32
   6.1 Analysis of User Experiences ......................... 32
      6.1.1 Speak the user language ......................... 33
      6.1.2 Minimize user memory load ..................... 33
      6.1.3 Provide Shortcuts ............................... 34
      6.1.4 Error conditions and Messages .................. 34
      6.1.5 Other heuristics .................................. 34
CONTENTS

6.2 Performance analysis .......................................................... 35

7 Conclusions and Suggestions for Future Work ......................... 36
List of Figures

3.1 Screen Capture Illustrating a Multi-View Session .......................... 9
3.2 Screen Capture Illustrating placement of secondary view in Multi-View Session . 10
3.3 Screen Capture Illustrating exchanged of primary and secondary sessions in a Multi-View Session .......................... 11
3.4 Screen Capture Illustrating a Pause-View Session .......................... 13
3.5 Screen Capture Illustrating a Pause-View Session when the player is paused . . . . . 14
3.6 Screen Capture Illustrating a Pause-View Session on resuming play .......................... 15
4.1 Rendering Engine (Observe the branching for that which runs in a separate thread) 20
4.2 Calculating Cameras for agents depending on agent context .......................... 21
5.1 Flowchart Depicting the Play/Pause Algorithm .......................... 27
5.2 Data Structure to Track Frames Captured in Pause-View .......................... 28
List of Tables

6.1 Performance Analysis ................................................. 35
Chapter 1

Introduction

The internet has and is revolutionizing the way we communicate. Using the web as our medium, online forums and social networking sites have been very effective tools for communication among large groups of people. Most recently, Virtual Worlds have proved to be good additions to these online social media. Virtual worlds, as their name suggests are pseudo worlds which offer rich digital content intended for rich interaction experience between a large group of individuals. According to wikipedia [22]: “A virtual world is a genre of online community that often takes the form of a computer-based simulated environment, through which users can interact with one another and use and create objects.”.

Virtual worlds have become synonymous to 3D virtual environments, where users take the form of avatars and interact with one another using text based chat system or voice support. Users navigate in these virtual worlds by walking, flying, teleporting and interacting with other avatars. Each virtual world is organized into multiple distinct locations, called regions and there are multiple virtual worlds. Some examples of virtual worlds are: Second Life [10], Metaverse [8], and Open-Simulator [20]. These virtual worlds are logical extensions to social networking spaces such as facebook and myspace. Due to their rich digital content and immersive feel they can represent real worlds very effectively and they have drawn attention from serious applications like edutainment,
online conferencing, virtual meetups and so on.

1.1 Hypothesis & Motivation

As stated above virtual worlds are organized into multiple grids known as regions. Each of these regions may represent a community or a Special Interest Group (SIG). These regions are used for holding fun events, academic sessions, conference meetings, and so on. The importance of online virtual worlds for professional work also cannot be ignored. For example, Intel’s ScienceSim [18] was developed for the purpose of scientific collaboration. Likewise Bluegrass [2] was developed by IBM to support spatially distributed teaming. Although they provide an environment akin to the real-world, the desire to provide real-world experiences has caused the designers to impose unnecessarily strict constraints from the physical world that are not strictly necessary in a virtual environment.

The principle appeal of virtual worlds is the “Sense of presence” experience that immersion into the virtual world can provide. In this case, “Presence” is defined as the subjective experience of being in one place or environment, even when one is physically situated in another [23]. Much like our physical reality, virtual worlds restrict these sense of presence experiences to a single region. Multitasking or need to be present at multiple spaces at a given instance of time is essential in an effective workspace. Multitasking impacts the sense of presence. Presence is governed by Social Richness, Realism, Transportation, Immersion, Social Actor in Medium and Medium as Social Actor [14]. The factor most impacted by multitasking is immersion. When user log’s into a different region to access different set of data, his presence is compromised by his required departure from the previous region. Similarly when an external, physical event occurs the user must exit (or ignore) the presence in the virtual world to engage the external event. This thesis contends that the user experience in a virtual world can be substantially improved by relaxing some of the artificial constraints from the physical world that have been carried into these virtual worlds. In
particular, this thesis explores the relaxation of the time and space linearities in a Second Life client to offer new opportunities for the user experience.

1.2 Overview of this thesis

The remainder of this thesis is organized as follows:

Chapter 2 will provide some background and surveys some work related to the topic of this thesis.

Chapter 3 provides a detailed overview of the extensions integrated into the Second Life client. These extensions are called Multi-view and Pause-View.

Chapter 4 will give in detail explanation of the implementation of Multi-View with examples of its use.

Chapter 5 will give in detail explanation of the implementation of Pause-View with examples of its use.

Chapter 6 provides a brief analysis and User Experiences with Multi-View and Pause-view.

Finally, Chapter 7 contains some conclusions and suggestions for future work.
Second life [13] is an Online Virtual World developed by Linden Labs. The Second Life Viewer is the client most commonly used to view and interact with this world. Second Life is divided into many regions which are characterized by specific themes or groups, for example it contains a beginners area, a region for the University of Cincinnati, regions for other universities and companies, and so on. Users enter these regions and interact as they would in a normal physical world; the visual experience of viewing the area and others (through their avatars) is maintained and communications occur through multiple media including voice and text messaging. Teleportation is the mechanism used for navigating between these regions. Second life, as the name suggests, provides a parallel to real life, another virtual life. In second life users are assigned to characters known as agents (or avatars) as they are popularly known. An avatar can interact with other avatars in virtual world. An avatar can be human controlled or bots programmed for specific purposes like customer service.

In Second Life users can own virtual property (real estate in Second Life), that they buy from Linden Labs. This virtual property can be modified with the construction of buildings and outdoor areas. This area can be open to all or restricted to allow access only to approved viewers. Avatars can create, edit, and exchange objects. Objects are the representations of real world materials.
such as furniture, electronic devices, musical instruments, abstract art and so on. These objects enhance the users understanding of the world and gives direct manipulation capabilities to user. Due to their rich digital content, their interactive communication environment, and their low cost of access, virtual worlds are increasingly being used for purposes beyond social networking.

Related Work

There have been a few previous attempts to provide access content from multiple regions in a virtual world simultaneously. One of them explored the possibility of accessing content by a notification system. The method consists of deploying auditory bots\footnote{7} named Marvin across the regions from which real time feed is needed. Though this method has vital advantage of saving on space screen, it proves to be the detrimental for presence. It does not provide the immersion experience which is vital for a “sense of presence.” In addition, the audio cues of multiple regions can get inter-mixed and easily confuse the user. Thereby causing the user to miss chance encounters or some other important notification cue and eventually actually reducing the users’ virtual world experience.

A more recent effort to support multiple region view paradigm is developed by Craven \textit{et al}\footnote{3}. The authors developed a multimodal display approach to support non linear drama, where in actions in one region affects the play in a different region in real time. In this method interactivity among multiple regions is achieved by deploying multiple views on multiple browsers which are projected on to large screens. The actors receive updates from different regions on the screens and perform their acts. This method is simple and can be theoretically be extended to multiple regions, but it is not scalable as it demands more hardware and increasingly larger computer monitor display space as the number of regions grows. Furthermore, the user will be restricted by the field of view, which may cause severe strain to keep track of multiple views as time progresses.

Browser based Virtual Worlds such as Metaplace\footnote{15} do offer support to access multiple re-
regions at same time in their standard viewer. However, these worlds hold much less content compared to the larger, richer environments such as Second Life. Furthermore, work is required to achieve spatial correlation between two regions on adjacent grids in a Collaborative Virtual Environment [1]. IBM has especially been exploring multi-region correlation to support the usage of Virtual Worlds in distributed work environments and meetings [2,19]. Multitasking [21] is a paradigm where users are inherently able to concentrate on multiple tasks. Unfortunately the available immersive virtual worlds do not yet allow for the same kind of multitasking available in the real world [16]. Immersive Virtual Worlds limit the user to a single context, not allowing him to utilize their complete potential.

Lastly, the MASSIVE environment [5] attempts to support modifications to the temporal linearity in a virtual environment by supporting a record/replay function. These recordings are represented as temporal links which specify the space and time coordinates of the recordings. The record and replay of temporal links are equal to the shift in the temporal coordinates of the spatial non linearity we have been discussing. No support for online pausing and faster-than-real time playback is supported.
Chapter 3

Multi-view and Pause-view

Multi-view and Pause-view are the extensions developed to seamlessly support multitasking in virtual worlds. In our physical workplaces it is common to work on more than one task at any time, which may demand for communicating with teams at different locations. Virtual teams, as they are commonly known, are the teams that are separated by time and geography but have a demand for co-presence. Traditionally interactions between virtual teams is achieved with traditional interfaces such as email, online chat, phone and so on. However, the same level of simplicity has not been transported to interactions using virtual worlds. For the user to collaborate with multiple teams either the teams must gather at one place causing overcrowding or user should login into multiple spaces. In virtual worlds and in their browsers the capability to sign into multiple regions is not yet incorporated. Therefore, the user must login into different browsers with different user names and keep jumping between them. Further complicating the matter, several virtual world browsers do not support opening multiple browsers on a single computer. This limitation of single view is quite artificial as it limits user to one virtual world instance at a given time; with our design we intend to work around this incapabilities of virtual world browsers by inducing non linearities in space and time coordinates.


3.1 Multi-view

Multi-view is the name given to the extension that embeds multiple region views in a single browser. It is similar to the picture-in-a-picture capabilities of modern day televisions. The main virtual world stream from the region where user is active is shown in the main view while information and updates from the secondary region of interest are shown in an inset window (Figure 3.1). It is superior to multi tabbed browsing in a way that, all the text and notifications are channelized into single stream. Users can always have eye on state changes in a session without loosing focus on the current session. Multi-view can be extended to n instances with n of them being inactive. Additional details on this are presented in Chapter 4.

A static view of the modified SLViewer that illustrates several features of multi-view is shown in Figure 3.1 (the illustration is synthetic in that the 4 menus shown would not ordinarily be simultaneously present onscreen). The current design supports a primary view (the main view of Figure 3.1) and a secondary view (labeled 1 in Figure 3.1). The secondary view can be positioned at specific locations on the screen (labeled 2 in Figure 3.1); note the placement of secondary view in two different positions in figures 3.1 and 3.2. Figure 3.3 illustrates swap of the primary and secondary views and the command to achieve the same is shown in original figure (labeled 3 in Figure 3.1). In addition, a keyboard shortcut triggers an immediate and rapid view swap without the popup menu. Control of the SLViewer avatar remains mostly as before and all of the primary operations apply to the primary view (e.g., the teleport option of label 4 in Figure 3.1 applies to the primary view). Sound is also tied to the primary view. However, we have extended the text window to capture (and tag) text from the secondary view (labeled 5 in Figure 3.1) as well as text from the primary view. Of course, the secondary view can be opened/closed under user control.

The essential use of Multi-view can be explained with the following e-commerce example. Consider user X wants to shop for an item Y sold by two stores Z1 and Z2. While Y has some additional accessories to be attached as A1, A2, and A3. In order to get best deal user in real time
Figure 3.1: Screen Capture Illustrating a Multi-View Session
Figure 3.2: Screen Capture Illustrating placement of secondary view in Multi-View Session
Figure 3.3: Screen Capture Illustrating exchanged of primary and secondary sessions in a Multi-View Session
runs from z₁ to z₁ and back, while in virtual worlds he teleports between these regions. The multi-view capability enhances this user experience by allowing him to bring these two places within a single context and rapidly (with one keystroke) swap focus between them. As another example from a more professional setting, one can think of the user X as an individual who wants to attend two simultaneously offered technical sessions (Z₁ and Z₂) at a conference or workshop.

### 3.2 Pause-view

The *Pause-view* extension incorporates a DVR like concept of streaming media to virtual worlds. Hollan *et al.* [6] argues that for any media to act as a successful medium of communication they should be capable of offering something more than presence. While virtual worlds may serve as alternative to real world conferences, they have much more in offering to user/agent. All users in a room may not assimilate the content at similar rate. The reasons behind this might be varied, ranging from need for attention towards different task to varied levels of comprehension achieved by the different participants of a diverse audience.

As the information served to user in real time can be quantified into frames, it is quite easy to control the rate of the information flows. We can embed user configurable stream processing speed play controls into virtual worlds that might open new avenues in collaboration techniques in virtual worlds. Machinima is quite popular in Second Life; primarily, because it has to capability of recording and replaying a world. With pause-view, we have effectively implemented temporal play controls into the Second Life client so that it can be used serious applications of learning, conferences, and many important events. *Pause-view* is similar to digital video recorder in satellite television; the user is able to pause (or record) a live broadcast and resume play at their convenience. Advanced controls for recording and replaying a specific portion of the stream without tight coupling to temporal coordinates may increase the usability of virtual worlds as collaboration tools.
Figure 3.4: Screen Capture Illustrating a Pause-View Session
Figure 3.5: Screen Capture Illustrating a Pause-View Session when the player is paused
Figure 3.6: Screen Capture Illustrating a Pause-View Session on resuming play
Pause-view allows the user to pause a live event streaming from the Second Life server. The user can then resume playback at faster than real-time and once the playback catches up to the original source stream, the system returns to normal (real-time, interactive) operation. Pause-view can be combined with multi-view and is triggered on the primary view. If pause-view is combined with multi-view, the current state (paused/active) of a stream is maintained in that view even as the user switches between the primary and secondary views. That is, a paused primary will remain paused if it is swapped with to the secondary view.

A static view of the modified SLViewer that illustrates features of pause-view is shown in Figure 3.4. Label 1 shows the pause and play buttons. Using these buttons user can pause and resume the stream of a virtual world as in normal media player. While label 2 shows the fast forward button which helps user to catch up with the real world updates, by viewing stacked up stream as quickly as he can. When the user pauses the main view, a message is displayed in the main view as depicted in Figure 3.5. On resuming play, user is provided with the player speed and remaining time (the red text in Figure 3.6).
Chapter 4

Multi-View Implementation & Examples

There are two viable approaches for implementing multi-view and pause-view into the Second Life experience. In particular:

Server side: Changes include support for multiple region login from a single browser and modifying the message template to support multiple region updates.

Client side: Changes include letting multiple users log in from single browser and creating parallel messaging system to support these agents.

Modifications on the server side are relatively complex for various reasons. Most significantly the source code for the server is not available. Thus to work server side, the modifications would have to be implemented in an alternative code base such as opensim [20]. Furthermore, without the ability to duplicate the content on the current Second Life servers, the user experience and demonstrations of multi-view would be severely compromised. Lastly, any server side modifications would also need to be complemented with client code revisions. Fortunately the modifications for both extensions can be achieved strictly on the client side with minimal drawbacks. The most significant drawback impacts multi-view and it is the need for the user to have multiple login accounts, one for each parallel session desired in pause-view. While less than ideal, this drawback
will not prevent the effective demonstration of the multi-view capability and once incorporated into the client viewer, login data can be captured, saved, and automatically used to streamline the users’ immersive experience.

In the remainder of this chapter, we will discuss the client side modifications for multi-view. The modifications for pause-view will be discussed in the next chapter.

The main modifications that are required to achieve the multi-view extensions in the Second Life client viewer are:

1. Support for multiple user authentication from a single browser.
2. Extending the Message System to support secondary users communications.
3. Effective context switch to handle updates from the parallel sessions.
4. Changes to the Rendering system to properly handle and display all user updates.
5. Modification to the Display code to project both views on the common display.

From the perspective of the actual source code, these modifications translate into the following code revisions:

- Create secondary threads (one for each additional view).
- Create new frame buffers.
- Expand Messaging system.
- Register the secondary users.
- Create context manager to manage multiple users.
- Setup secondary cameras and projection/model view matrices.
- Image Decode and Texture Fetch.

Each of these steps are explained in detail below:
Spawn a thread: Generating a scene in second life is computationally expensive process. The Frames Per Second (FPS) rate of the SLViewer on regular computing machine ranges from 7-14, any deterioration below this number would result in a degraded experience for user. To maintain same performance levels while rendering multiple scenes, we created a thread to handle tasks related to secondary view. This thread handles both logging in and rendering process. It is spawned after primary user registration. This approach keeps the design elegant and modular, separating the primary viewer stream flow from each stream for an alternative view.

Support for threads is readily provided in second life framework through llthread which is developed on top of apr_thread_t offered by the apache portable run time library. These threads are invoked at the end of a rendering cycle. The llpnpthread class represents secondary view rendering thread. Image and texture fetches for this view are managed by already existing image and texture fetch threads. As detailed later in this section, messaging systems for image decode and texture fetch are different and messages are routed to corresponding simulators with the help of context manager.

Create an off-screen frame buffer: Second life is built on a single opengl buffer. The Scene is rendered into this buffer and then projected onto the view port. In order to render multiple views, multiple buffers are required; the images of these buffers are projected as subimages to the main view. We used the render to texture method \[24\] to project these subimages.

Off-screen rendering and binding concept render to texture is used to post the secondary view. In this technique the scene is rendered into an offscreen framebuffer known as frame buffer object (fbo). This fbo’s are in turn bound to a texture which are created by the user. Hence objects that are rendered into framebuffer are populated onto the texture. Texture thus generated is rendered into main framebuffer as a subimage. The Second life viewer framework has APIs to create off-screen framebuffers and use them to render a texture. This texture can be rendered as a sub-image at any place in the primary window by specifying the co-ordinates.
In our work we have provided shortcuts to move this screen easily onto any one of the four corners. Future work may involve developing mouse click interface to move and resize this view to the required size. The performance impact incurred using this technique is small and further discussed in Chapter 6.

**Expand Messaging System:** Second life works in many client to one server model. The server that sends messages necessary to render the scene are known as the Simulation server or simulator.

![Diagram of Rendering Engine](image)

**Figure 4.1:** Rendering Engine (Observe the branching for that which runs in a separate thread)
According to the Second Life wiki on the server architecture [11]: “Each simulator process simulates one 256x256 meter region. As the viewer moves through the world it is handled off from one simulator to another. It handles storing object state, land parcel state, and terrain height-map state. It performs visibility calculations on objects and land and transmits the data to the client. It transmits image data in a prioritized queue. Chat and instant messages are processed here.”

The communication of Second Life messages a transmitted between the server and client is by UDP using circuits. Each UDP circuit has a unique identification number. However this identification number is not sent with all the messages between server and client. Hence it would be difficult to establish multiple circuits without making changes to both client and server. Because we decided to restrict our modifications to the client side only, we solved this problem by implementing a parallel messaging system on the client side. In particular, the tight coupling of the client code

![Diagram](image_url)

Figure 4.2: Calculating Cameras for agents depending on agent context
to gMessageSystem (the current messaging system in SLViewer) in the messages pertaining to regions and agent updates, also forced us to use a parallel messaging system.

Though the underlying message system is separate, the front end presents and displays the text messaging system as unified across all multi-view sessions. Thus, text messages and notifications are displayed in the same console area, but for the messages from the secondary views are shown with a prefix (from parallel universe, Figure 3.1). This keeps things simple and aids the user in quickly processing messages from the multiple sessions.

Although we generally modularized the code based around the messaging systems, the agent and object creation is modularized around region of existence due to the non-availability of message system parameters. Apart from this, the Second Life universe consists of event polling mechanism that takes care of teleport start, update, finish, and enabling neighboring region simulators. A parallel event polling system was also instantiated after the creating the message system. Changes to the message system are easily the biggest and central change/challenge we faced in the entire architecture.

Register Secondary User: Second life necessitates the presence of a user in order to stream the updates of a user into the Second Life client viewer (SLViewer). Thus for successful demonstration of the multi-view paradigm a user must be present at all the places of interest (sessions). This requirement is not possible to achieve with a single user in the current Second Life configuration. This fact creates the second major change/challenge in this work.

Second Life has 5 different types of back end servers to handle different tasks. The Login Server handles user login, and checks for the user home based on last login or url specified while logging in. It informs the simulation server about the possible user arrival [11]. It does not allow the user to connect to multiple simulation servers. As mentioned earlier it is a many to one relationship between a client and server. Therefore, in order to receive updates from a secondary region we must register a secondary user. The secondary user and corresponding region updates are tied
to the parallel messaging system described above. When the user credentials are validated, this messaging system is instantiated. For this extension and demonstration, we have hard coded the user first name, last name and password of the secondary user. As a future work, secondary user credentials can also be taken as input from the login screen.

**Setup secondary camera and projection/model view matrices:** The camera is the eye of third person or god view into the agent/user’s world. It is synonymous to somebody recording our actions and displaying it to the outer world. The default view in second life is third person view, this is the most common view in role playing games. It helps in observing the surroundings with ease and gives panoramic view of the world. The agent’s view to the world is synonymous to the camera view [9] and each agent needs a camera to represent him. In technical terms, the camera is the heart for performing culling, sorting and rendering on agent’s view. The camera has many properties attached to it, world coordinates, region coordinates, angular velocity, point of interest to name a few.

In order to render views of an agent all this properties are to be populated into the camera object. For multiple views the properties of the camera must be changes based on the agent. Alternatively the same result can be achieved by treating the camera as another object of the user, thereby changing its property with an agent switch (swapping the primary view with one of the secondary views). This helps in minimizing the code changes for context switch and improving performance of system. In the current design, code pertaining to the camera is object oriented. However the camera is declared as an singleton object, hence it could be used only once. Our changes include removing the restriction of object being a singleton; making the camera a property of the user and changing the direct access of the camera to access from user. Changes done to the camera access is depicted in Figure 4.2. This model can be easily extended to support multiple views in the same browser beyond just one or two.
Create context manager to manage multiple/two users: The context manager is a pseudo system that will help rendering and system update to choose the correct parameters. As mentioned above, the agent, the message system and the camera are quintessential in SLViewer. As two views are rendered by the same graphics pipeline, there is always a need to select the context to generate necessary parameters for rendering. The working of the rendering engine was made simple by enabling/disabling the primary agent active flag for corresponding threads. While the indecision that occurred on applying context for network updates are resolved by either looking at region of the object or the ip address of the host of object. The clearing of flags for all kinds of updates are taken care of appropriately. In our modifications to the system, we maintain the basic setup and add this pseudo system to separate and yet coordinate the computations between the primary and secondary views.

Image Decode and Texture Fetch: A textured black and white world is better than plain one, and so is the colorful world with images all around works much better for eyes than a black and white world. Images and textures attains these effects and transforms a bland black and white world into rich colorful environment. Except for loading cache miss images and textures the current SLViewer works in place for secondary view too. However, the cached images are an important component for complete rendering. Cache misses are pooled in when network update messages are processed. At the end of network updates, this missed images/texture upload requests are sent in batches of 256 messages. As discussed in previous section, using the context manager we pooled in the missed image/texture id’s into a separate bucket; And upload request for these images are sent via secondary messaging system.
Chapter 5

Pause-view Implementation & Examples

Pause-view allows the user to keep track of all the active events in his Second Life world whenever he decides temporarily suspend his Second Life experiences to perform some other tasks. When the user pauses all the updates in the Second Life world must be captured and presented (later) in the same chronological order that they originally occurred. The user may also desire to view these updates at faster than real time play. It requires functionality akin to streaming media players such as youtube [25] or a digital video recorder, which buffers the media onto local disk for supporting asynchronous pause and play activities of local users.

The following strategies were explored to implement Pause-view:

**Video capture tools for recording and a built browser for playback:** We worked on exploring and applying open source software for recording and in building support of SLViewer for playback. Instead of reinventing the wheel of developing software for pause, record and replay in media applications we explored the use of existing tools. Many standard tools are available to record content of OpenGL based applications such as SLViewer. A comprehensive list of tools available for recording in Second Life are listed here [12]. After a literature review and numerous prototyping experiments, we decided to use glc for our capturing needs as glc is able to capture
CHAPTER 5. PAUSE-VIEW IMPLEMENTATION & EXAMPLES

any application that uses ALSA for sound and OpenGL for drawing [4]. For play back purposes, we intended to use builtin web browser in Second Life by embedding a media player into the browser and playing the recorded video from local disk.

GLC is very effective in capturing the content of Second Life. It captures video at a very high frame rate without any perceivable loss of content. This capability comes at a disadvantage of requiring a large disk to store the recorded content. Typically, a one minute pause demands a space of 400 MB for storing the recorded content, which raises concerns of scalability when the pause time increases. In addition, the web browser inside Second Life lacks support for JavaScript which is needed to embed and play media applications. Developing a JavaScript engine for this browser is a prohibitive task, as this request in itself is a big project.

Attempt 1: Video capture tools for recording and media player for playback: The above method can be deployed, using external media player such as vlc for play back purposes instead of the builtin web browser SLViewer. It requires the user to switch back and forth between the Second Life browser and media player. Though this method works, it is clumsy and defeats the purpose of project which space time multiplicity inside a Virtual World browser.

Attempt 2: Capture Frames for Recording and display in SLViewer for playback: The need for in place viewer to playback updates, which the previous two methods have failed to achieve has led us to prototype the following idea. The SLViewer has APIs that support capturing screen-shot’s and storing them to local disk at run time. We used these APIs to capture and encode each frame, and index them by timestamp. In playback mode, these pictures are decoded and displayed on top of normal Second Life world browser. The disadvantages with this method are very high processing power need to encode images and need for equally high diskspase for storing images. As a proof, though there was no frame loss with this method, frame rate has dropped down to 0.5 from an average of 9-11 fps in normal mode of operation.
CHAPTER 5. PAUSE-VIEW IMPLEMENTATION & EXAMPLES

Attempt 3: Capture network updates for recording and delay rendering for play back: The common problem in the previous methods is high storage space needed for storing updates. We solved this problem by storing the binary data that is required to render the scene from the current scene. This data is extracted from the network packets that come through UDP. We detail this approach in the remainder of this section.

The complete flowchart for the Pause-view implementation is presented in Figure 5.1. In this

Figure 5.1: Flowchart Depicting the Play/Pause Algorithm
CHAPTER 5. PAUSE-VIEW IMPLEMENTATION & EXAMPLES

Figure 5.2: Data Structure to Track Frames Captured in Pause-View

implmentation, network packets received by the system are stored for rendering when play is resumed. Hence we must be able to map each packet to the corresponding frames and so the data structure we use to map packet numbers to the frame id is shown in Figure 5.2.

The SLViewer is a multi-threaded application. The main thread is responsible for rendering the scenes into the graphics window; the other threads are responsible for activities such as texture fetch, image decode, and so on. The main thread goes into an idle loop after each rendering cycle. In the idle loop, the main thread synchronizes application across threads; captures network updates that are used to update existing objects or create new objects in the agent’s view. The network update cycle involves the following steps:

1. Check Message
2. Receive Message
3. Validate Message
4. Decode Message
5. Update Network Statistics

While each of these steps has its own significance, the one that relates to the user-interface
update is Decode Message. During this stage, messages are matched with its template and blocks of data are read that will update a standard data structure in the SLViewer code. Post packet reading, a callback function that maps to the message template is invoked. The call back function updates the scene. It is possible to save the message once it is validated and execute the call back function corresponding to the message at a later instance of time, this translates into executing the update with a delay and it is a suitable solution for the recording and playback features of Pause-View.

As shown in Figure 5.1, when the SLViewer is in record updates mode (when the viewer is paused), step 4 of the idle network cycle is conditionally replaced with code that saves the content of network packets to local disk. These packets are indexed and grouped together by frame. As shown in Figure 5.2, each buffered data object contains the list of packets that corresponds to a particular frame. These buffered objects are stored in another list, which manages this updates in order.

BufferDataList is a FIFO data structure and updates are organized in their original stream order. In playback mode the buffered data is pulled from the front BufferDataList and is executed at the end of each rendering cycle (which is idle cycle for the application). Then callback functions for the data in these packets are executed. New incoming message packets are extracted during the current cycle are pushed to the back of list. During normal playback mode, in each rendering cycle one packet list is popped out while another list is pushed keeping the BufferedDataList size constant.

The playback feature of Pause-View also has a fast forward mode (ffm). If we assume that X is the desired fast forward speed, then X BufferedData objects are popped out of the BufferDataList and the corresponding call back functions are invoked. In this mode, the BufferDataList will be consumed at X times faster than normal mode and the system will ultimately return to real time play. When the BufferDataList is empty the flags will be reset and the player will return to normal operation. During playback and ffm the agent cannot perform any actions on the objects except for text chat. Sending of the text messages will be in real time while receiving will still continue to in
playback mode until the player catches up with real time feed.

This approach has many advantages, namely:

1. **Minimal Storage Consumption:** The average disk space required for storing 1 hr feed is 200 MB, which makes it nearly 100 times more efficient in comparison to earlier methods. This number may vary for regions which are extremely dynamic. However, given that the maximum packet size in Second Life is 80KB and the average packet size for the applications we tested is 35KB, it still fares very well in comparison to the other methods we explored. With it comes very high scalability.

2. **Finer level of control in play, pause of Multi-View paradigm:** Multi-view as described in the previous sections is about the ability to simultaneously view multiple regions in Second Life. Furthermore pause-view is about controlling the speed of streams in these worlds. Because the static picture capture tools such as glc capture the entire scene by operating on top of the SLViewer, they do not have ability to operate separately on each view when operating in multi-view mode. With our implementation, either, several, or all views can be independently paused and resumed. Thus if the action in multiple views becomes highly significant to the user, they can pause one of the views while the other plays out and ultimately return to playback the captured second view and not miss anything important in the second stream.

3. **Performance:** By saving only binary data to disk, we are writing a minimum amount of data to disk thereby reducing the amount of time to write to disk. Although this has some minor performance implications, the performance impact is nearly negligible compared to the other methods we explored (where frame processing rates dropped from 9-11 fps to 0.5-1 fps; in contrast to this solution’s 7-10 fps).

4. **Smooother Steps of fast forward speed control:** We group packets by the frames which they are about to update. Thus, the list of packets in bufferedData object translates to the difference
between updates of two frames. Hence by applying updates from X number of objects directly translates into X speedup. Thereby making it easy to apply any integer speedups capabilities to the playback mode.
Chapter 6

Analysis of Multi-View and Pause-view

In this chapter we study both the impact that multi-view and pause-view have on (i) the user experience, and (ii) the run time performance of the modified SLViewer.

6.1 Analysis of User Experiences

It is very difficult to quantitatively analyze user experiences. Neilsen [17] has quoted 9 heuristics for a successful User Interface design evaluation by users. The advantage of heuristic evaluation are it is intuitive, easier to train people for this kind of testing, relatively cheap, and can be used in all the stages of development. These advantages matched the needs of our user interface design evaluation process and so we followed them. The heuristics used for evaluation are:

- Simple and natural dialog.
- Speak the user language.
- Minimize user memory load.
- Be consistent.
- Provide feedback.
• Provide Clearly marked exits.
• Provide shortcuts.
• Good Error Messages.
• Prevent errors.

We review each of these heuristics for multi-view and pause-view below.

### 6.1.1 Speak the user language

The multi-view and pause-view extensions does not add any complexity to the natural flow in virtual world play. While multi-view develops a parallel flow to the already existing flow, pause view provides granularity on time coordinates. There are no change the language usage with the addition of this paradigms to second life. Hence our extensions perform equally well with original viewer on both these heuristics.

### 6.1.2 Minimize user memory load

Both multi-view and pause-view score high on these heuristic in comparison to the original viewer. The main purpose of multi-view is to provide synchronous feed to the user from multiple regions without losing information. In the other alternatives (described in Chapter 2) the user was required to keep regular tabs on multiple regions while switching between them. In those methods, the user must manually store and retrieve the previous state of the region. In contrast, multi-view removes most of the effort needed for the user to manage multiple session streams. It provides these multiple streams for the user in a convenient overlay of the Second Life viewer and combines the multiple views with easy to use rapid switching capabilities. Furthermore, the message streams coming from the multiple sessions are conveniently captured and displayed in a common area with message from secondary sessions tagged as such for fast user recognition.
Likewise pause-view extends the user experiences and offers users the opportunity follow a session stream at a pace suitable to their needs. Without the pause-view extension the user must be continuously alert throughout the session. Any lapse in concentration or a parallel work assignment will affect the user. With the help of pause-view, the user can time the information flow according to his convenience, hence reducing strain on users and enabling them to be more effective multitaskers.

6.1.3 Provide Shortcuts

We provided shortcuts in multi-view along with the menu-dialogs. This shortcuts are consistent with the already existing shortcuts in the SLViewer. With the help of these shortcuts, the user can switch between displays, change the placement of secondary view and play/pause the world. Hence, multi-view and pause-view satisfy these requirement as well.

6.1.4 Error conditions and Messages

After thoroughly testing the multi-view and pause-view, we confidently state that they are error free under normal conditions. Some of the abnormal conditions might be running out of disk space or graphics card compatibility issues. These exceptions are gracefully handled by the original Second Life viewer and we preserve them in the extensions. Both multi-view and pause-view are good on this heuristic evaluation appear to satisfy this heuristic.

6.1.5 Other heuristics

The multi-view extension provides an equivalent experience of having multiple Second Life viewers adjacent to each other. While using multiple instances of the SLViewer operating on one workstation console may provide simultaneous view of multiple regions in Second Life, they do not alter the user experience of each individual region or unify the users’ Second Life experience. In
conclusion, both the original viewer and our viewer with extensions perform equally well on most of the heuristics. However, the integrated multi-view and pause-view extensions provide a more coherent and integrated experience for the user.

6.2 Performance analysis

We have collected frame processing rate (fps) statistics to determine what impact the multi-view and pause-view extension have on the display performance of the SLViewer. In particular, the need to render multiple scenes from multiple regions in multi-view may have some negative impact on the overall display performance shown in Table 6.1. We have conducted these tests on the viewers running on a debian system with an nvidia 6800m graphics card and 256 MB RAM. These results show that our extensions have nominal impact on performance. On a more robust hardware platform, these impacts should be even less noticeable.

<table>
<thead>
<tr>
<th>Multi-View</th>
<th>Pause-View</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>11.6</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>9.4</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>11.3</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>9.2</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusions and Suggestions for Future Work

In this thesis we added extensions to the current virtual world browser to incorporate space and time non-linearities into virtual worlds. These changes are meant to provide a dynamic user experience in 3D virtual worlds. In particular, we have modified a second life client viewer, SLViewer, to incorporate multiple simultaneous views (multi-view) into the spaces of second life and we have also incorporated a pause/resume feature (pause-view) into the SLViewer. The multi-view extension provides a picture-in-a-picture like viewing experience and the pause-view extension supports a DVR like pause/playback experience. The possible design space of multi-view and pause-view were explored and various solution options evaluated. In the end, we decided that a client side implementation would provide the richest experience to evaluate the extensions. While implementing the client side multi-view and pause-view extensions, several possible design solutions were explored and contrasted. Ultimately many of the design choices were made to minimize the performance of the interactive performance. In the final design, we experienced minimal impact and retained virtually all of the design features that were originally envisioned for both extensions. We also conducted an analysis of our solution based on user interface design heuristics. We argue...
that bending the space and time boundaries in 3D virtual worlds opens up exciting opportunities to improve the social and collaborative work experiences.

Future Work

Though we are able to provide extensions with the help of minimal code changes to client side, server side changes might extend the possibilities even further. For example, the interoperability of the multiple regions of multi-view would be much easier. Scaling of the system from 2 views to n views can be done with trivial changes on server side in comparison to client side changes. This would present a more consistent view of presence in virtual worlds. Video on demand is another concept that can be easily extended to virtual spaces. Instead of client side recording of video, server side recording and managing the client side requirements would provide with an effective way to deal with space requirements.

In addition, as of now only the main viewer’s audio is streamed through the browser. When appropriate a functionality switch between audio streams should also be provided. Another exciting possibility includes seeking through the paused view with the help of event markers. Finally, adding event notifications to pause-view, so that the meeting source could notify a paused view of important events could further help improve the collaboration experiences.
Bibliography


