

High Resolution Video Playback in Immersive Virtual Environments

Han Suk Kim*
Computer Science and Engineering
University of California San Diego

Jürgen P. Schulze†
California Institute for Telecommunications and
Information Technology
University of California San Diego

ABSTRACT

High resolution 2D video content in High Definition or higher resolutions has become widespread and video playback of such media in immersive virtual environments (VE) will be a valuable element adding more realism to VE applications. This kind of video playback, however, has to overcome several problems. First, the data volume of video clips can reach up to hundreds of gigabytes or more depending on the length of the clips, and the data has to be streamed into virtual reality (VR) systems in real-time. Second, the interactivity of the playback screen in 3D virtual environments requires efficient rendering of each video frame. Interactivity means that the plane of the video playback screen needs to rotate, translate, and zoom in and out in 3D space as the viewer roams around in the VE. This also means that the video is not necessarily parallel to the display screen but will need to be displayed as a general quadrangle.

In this work, we propose an efficient algorithm that utilizes mipmapped data, that is, multiple levels of resolutions, to provide an efficient way to interactively play back high resolution video content in VEs. In addition, we discuss several optimizations to sustain a constant frame rate, such as an optimized memory management mechanism, dynamic resolution adjustment, and predictive prefetching of data. Finally, we evaluate two video playback applications running on a virtual reality CAVE system: 1) high definition video at 3840 x 2160 pixels and 2) 32 independent 256 x 192 pixels video clips.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality H.5.1 [Information Interfaces and Presentation (e.g., HCI)]: Multimedia Information Systems—Video

1 INTRODUCTION

High resolution 2D video content has become widespread nowadays due to advances in digital video recording technologies. Many videos are stored at high resolutions, e.g., 4K (up to 4096x2400 pixels) or HD video (1920x1080 pixels). The virtual reality community has long tried to embed high resolution video playback as realistically as possible in immersive virtual environments (VE). Possible applications can be a virtual theater, a virtual office, video conferencing, video surveillance, or video commercials.

However, due to the large volume of data in high resolution media, which easily exceeds hundreds of gigabytes, it is not an easy task to play back these videos in real-time. Constant I/O operations are required to load data in a timely manner and video rendering systems need to manage the limited memory resources. Furthermore, in a VE the video image is not displayed as a rectangle, but as its projection on the screens, which is a general quadrangle in an arbitrary orientation. Existing video conferencing systems can

*e-mail: hskim@cs.ucsd.edu

†e-mail: jschulze@ucsd.edu

stream video of people in conference rooms to remote locations. Part of this technology is applicable to video playback in VEs, but there are many differences in the system requirements.

The contribution of this work is to provide an efficient algorithm and its implementation that can handle high resolution video data so that the video streams can be displayed in VEs. Our main idea of handling high resolution video in VEs is to use mipmapping [8], on top of which we added various optimizations to allow for constant frame rates under varying viewing conditions and rendering rates determined by the rest of the 3D virtual world that needs to be rendered. In our approach, the preprocessed image data has several levels of detail (LOD), that have different downsampled resolutions of the same data, and choosing the best possible LOD can reduce the amount of memory required. In addition, we added advanced memory management algorithms, such as an out-of-core algorithm and tiling, to efficiently utilize memory. The performance of the overall system has been further optimized by having mechanisms of dynamic LOD adjustment and predictive prefetching of data.

For the playback of most video clips it is important that the video plays at a constant frame rate, the rate it has been recorded at or rendered for, the *video frame rate*. This is particularly important if there is a corresponding audio track to be played back in sync with the video. However, in interactive computer graphics applications, the rate at which the images get displayed on the physical screens, which we call the *image frame rate*, is not uniform and it is independent of the video frame rate. Our system provides a constant video frame rate despite a varying image frame rate.

2 RELATED WORK

Mipmapping, which is the level-of-detail (LOD) approach we employ in this project, was first introduced by Williams et al. [8]. Pre-computed texture data at multiple resolutions, downsampled from the original image, greatly eased the tight texture memory resource budget. The concept of Clipmaps [5], which are virtual mipmaps, extended the mipmap concept to load arbitrarily large images into graphics memory by splitting the images up into smaller pieces. It is very close to our work, except for the fact that clipmaps only considers a set of 2D textures of one image frame. OpenGL Volumizer [3] implements clipmaps for volume data sets and supports animation, but it does not have a mechanism to maintain a constant video frame rate, and it is designed for 3D volumetric data, not 2D video.

3 IMPLEMENTATION

3.1 Mipmap Generation and Tiling

In our approach, we pre-process the video frames in an off-line step to create the mipmaps. We implemented a tiling [6, 7] technique in which the atomic unit to read and write data is a texture of the same size.

3.2 Mesh Generation

The first step of rendering a video is to subdivide the playback screen into a set of tiles, which we call the mesh. The mesh is comprised of multiple tiles of different mipmap levels. The goal of subdividing the screen is to allocate the best possible mipmap level to each region with a limited number of tiles. Another goal is

to distribute limited resources of disk read to the region as fair as possible.

The mesh generation algorithm is based on a quadtree and the view frustum culling test for each tile reduces the cost of quadtree traversal significantly by pruning unnecessary nodes. During the traversal, in order to assign priorities to each tile, a cost function, $cost(b)$, is given as $area(b)/distance(e, b)$. $distance(e, b)$ measures the distance between the viewer's location and the center of tile b . The viewer's location is given by the head tracker in the VE. The tile that has the highest cost is selected and subdivided during the traversal. Another variable, $tileLimit$, controls the number of tiles to be rendered on the physical display screen and it guarantees that the rendering system does not render more tiles than can be rendered in the allotted time for the video frame.

3.3 Data Loading and Prefetching

We implemented three optimization methods to load necessary texture data from disk to system memory and then to the GPU efficiently.

Prefetching One observation we made was that once viewers in the VE walk up to a video clip, they often stop to watch it. At that point, the tile mesh remains relatively unchanged. Thus, we can use prefetching to load mipmap tiles for future frames ahead of time.

Asynchronous I/O In order to accelerate the data transfer between main memory and texture memory, a separate thread is spawned and dedicated to asynchronous disk I/O operations. Every disk read request is sent to the I/O thread via a message queue and the I/O thread reads data whenever it finds a message in the queue.

Memory Pool and Cache The third optimization we implemented is to pre-allocate a pool of memory blocks so data loading can save time for allocating memory blocks. The pool of memory blocks consists of a list of blocks, each of which can store the texture data of one tile. The pool is initialized both in main memory and in texture memory.

3.4 Dynamic LOD Adjustment

The dynamic LOD adjustment algorithm monitors the image frame rate. If the frame rate is below the video frame rate, then the algorithm decreases the number of tiles to be rendered for the current frame. Thus, the rendering system can recover from a burst of unpredictable disk I/O requests, resulting in very low frame rates, and as the frame rate becomes stable, $tile\ limit$ is restored incrementally.

3.5 Synchronization

The time for rendering a frame changes over frames and this can cause two types of synchronization problems: synchronization 1) between frames and 2) between CAVE nodes. A synchronized clock across all nodes of our CAVE software offsets frame numbers so that frame number changes neither too fast nor too slow.

4 RESULTS

We have implemented the above described video playback algorithm for virtual environments running on PC clusters by writing a C++ plug-in for the COVISE software framework. We tested two different videos in our CAVE-like virtual environment, the StarCAVE [4]. It consists of five walls, each of which has three rear projected screens, and there is a top projected floor. Each screen is projected on by a pair of JVC HD2K projectors (1920 x 1080 pixels each), which are connected to an Intel quad core Dell XPS computer running ROCKS [2], with 4GB of main memory and dual Nvidia Quadro 5600 graphic cards. We distributed the video data to the local hard drives of the nodes. In our experiment we used two different video clip scenarios: one using a single 4k (3840 x 2160 pixels) clip showing the result of a tornado simulation created by

Resolution	High	Medium	Low
Image Frame Rate (fps)	1.79	17.27	60.09
Number of Tiles	151	24	6
Texture Size (bytes)	7,421,952	1,179,648	294,912
Texels/sec	4,428,431	6,790,840	5,907,087

Table 1: Performance comparison at three different video resolutions. (Screen resolution: 1920 x 1080 pixels)

NCSA [1], the other one consisted of 32 low resolution (256 x 192 pixels) video clips. Tiles of 128 x 128 texels were used for both data sets.

Table 1 compares rendering performance at three different resolutions. The first experiment was set to achieve as high resolution as possible. The total number of tiles loaded at every frame was 151, which corresponds roughly to the screen resolution. The average image frame rate was 1.79 fps. The medium resolution was about DV quality, which we can render at 17 fps. The low resolution setting used only 6 tiles to test the prefetching algorithm. It turned out that all the tiles could be prefetched before rendering of an image frame started. All three experiments were measured on one node but at full screen resolution and all the previous buffer cache blocks in main memory stored by the operating system were flushed to prevent disk caching effects from influencing the results.

The second video data was, however, small enough to be cached in the main memory of our system, and we deliberately allowed disk caching. The operating system cached the data blocks read from disk with a cache size of 2-3GB. Although our system does not create a memory pool large enough to store the whole video clip (1.3GB), after two or three playbacks, most data started to be loaded at a much faster speed.

5 CONCLUSION AND FUTURE WORK

We have shown the design and implementation of high resolution video textures in VEs. The main disadvantage of the current implementation is that data has to be preprocessed to generate mipmaps, which prevents it from being used for live video. Since the downsampling process, however, is highly parallelizable and modern GPUs provide hardware supported parallel computing (e.g., CUDA), real-time downsampling might be feasible in the future.

REFERENCES

- [1] National Center for Supercomputing Applications. <http://www.ncsa.uiuc.edu>.
- [2] Rocks Cluster. <http://www.rocksclusters.org>.
- [3] P. Bhaniramka and Y. Demange. OpenGL volumizer: a toolkit for high quality volume rendering of large data sets. *Proceedings of the 2002 IEEE symposium on Volume Visualization and Graphics*, Jan 2002.
- [4] T. A. DeFanti, G. Dawe, D. J. Sandin, J. P. Schulze, P. Otto, J. Girado, F. Kuester, L. Smarr, and R. Rao. The starcave, a third-generation cave and virtual reality optiportal. *Future Gener. Comput. Syst.*, 25(2):169–178, 2009.
- [5] C. C. Tanner, C. J. Migdal, and M. T. Jones. The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 151–158, New York, NY, USA, 1998. ACM.
- [6] W. Volz. Gigabyte volume viewing using split software/hardware interpolation. *Proceedings of the 2000 IEEE symposium on Volume Visualization*, January 2000.
- [7] D. Weiskopf, M. Weiler, and T. Ertl. Maintaining constant frame rates in 3D texture-based volume rendering. *Computer Graphics International*, Jan 2004.
- [8] L. Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, 1983.