View-Dependent Real-Time 3D Video Compression for Mobile Devices

Shu Shi Klara Nahrstedt Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave., Urbana, IL 61801, USA
{shushi2, klara, rhc}@uiuc.edu

ABSTRACT

3D video is an emerging technology that promises immersive experiences in a truly seamless environment. Currently, 3D video systems still require excessive bandwidth and computation power provided by gigabit switches and multi-core workstations machines. In order to extend the experience to mobile devices, we present a view-dependent compression methodology that shows great promise in making 3D video a reality on resource-constrained mobile devices. Using our technology, we are able to achieve a software-only rendering on a Nokia N800 PDA with only wireless network transmission. We believe that with the use of newer handhelds and improvements to our compression techniques, we will be able to deliver full-motion 3D video soon.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information System**]: Video

General Terms

Design, Measurement

Keywords

3D Video, Compression, Viewpoint

1. INTRODUCTION

In recent years, 3D video has appeared in many research projects [9, 8, 6]. As a result of the development and convergence of computer vision, graphics, multimedia and related fields, 3D video is different from conventional 2D video. The 3D information contained for each video pixel offers not only the 3D depth impression in presentation but also the freedom of watching the video objects at any viewpoint and merging video with different virtual graphic environments. This new media expands the user's sensation and interactive experience far beyond what is offered by 2D video.

MM'08, October 26–31, 2008, Vancouver, British Columbia, Canada.

Figure 1: TEEVE in Nokia N800

In our TEEVE (Tele-immersive Environments for EVEry-body) project [9], 3D video streams are captured, transmitted and rendered at real-time to build a tele-immersive environment. Depth image, a point-based 3D video representation, is used as the data model of TEEVE. Multiple depth image streams generated by 3D cameras from different viewpoints can be used to reconstruct the 3D scene. As a result, TEEVE enables users who are physically apart in distant places immersed into a shared virtual space and displayed on the same screen as if they are in the same room. Researches [10] showed great potentials of 3D video tele-immersion in different applications such as videoconferencing, remote education, medical service and so on.

In order to increase the mobility of the system and exploit new features, we are currently exploring TEEVE on the mobile platform. For the first step, we try to receive and display 3D video streams with a Nokia's PDA (Figure 1). Imagine such a scenario: we don't have to stay in the lab room, but use our own PDA to watch 3D video. We can customized the video background and select the preferred viewpoint. Furthermore, the motion and position sensors in the PDA may detect our movements and adjust the viewpoint automatically. However, unlike earlier researches on powerful desktop servers, 3D video application for mobile devices faces more challenges. One of the major bottlenecks is the limited bandwidth. In TEEVE, the basic bandwidth of Gbps level is required to transmit multiple depth images [11]. But the wireless network environment for the PDA can only provide a few Mbps bandwidth. The situation may be further exacerbated when more devices are transmitting data at the same time. Therefore, the compression of depth image streams is crucial to the 3D video application on mobile devices.

Compression techniques on depth image have been studied from many aspects. ATTEST [8] suggests using the standard video codec H.264 to compress depth data. The proxy-based compression proposed by Penta and Narayanan [7] and the skeleton-based compression introduced by Lien [5] both use pre-defined models to reduce data redundancy. Yang [11] proposed two general schemes for intra-stream compression and Kum [4] studied the inter-stream compression using reference stream. However, none of these techniques is suitable for the mobile application.

This paper proposes a new view-dependent approach of 3D video compression scheme to solve the mobile constraints. The main idea of scheme is to compress and transmit only useful data to mobile devices. Given a specific viewpoint, occluded and invisible objects in the 3D video are omitted when displaying the video. These objects are unnecessary for compression and transmission. If only a few hundred pixels are used to display on the screen of the mobile device client, why do we need to transmit thousands of original 3D points? The simple optimization is to pre-render the 3D video on powerful servers and transmit 2D images to mobile devices and display them directly. It doesn't only reduce the bandwidth requirement, but also simplifies the computation of rendering on mobile devices.

The rest of the paper is organized as follows. Section 2 gives an overview of the compression framework. Details and important steps about the compression scheme are discussed in Sections 3. Section 4 evaluates the performance. Section 5 summarizes related work and the final section concludes the whole paper and suggests future work.

2. OVERVIEW

2.1 Data Model

The data model of 3D video assumed in this paper is the depth image. The depth image comprises of an image map and a depth map. The image map is a 2D image recording RGB color value and the depth map records pixels' depth value for each pixel. 24 bits are used for color value and 16 bits for depth value. The depth image stream contains depth image frames taken by the 3D camera with fixed time interval and camera parameters. All cameras capturing depth image streams have to be synchronized and calibrated under a global coordinate system. Parameters for camera synchronization and calibration are stored in the stream header and used to reconstruct the 3D coordinate information for each pixel in the depth image.

2.2 System Model

The proposed compression is a closed loop with feedbacks (Figure 2). The compression framework has two parts: the server and the client. The server receives multiple depth image streams from the network or reads them from a file. 3D points are reconstructed and rendered on the server side with the respect of the client's viewpoint information. The points visible to the client are maintained while invisible points are discarded or highly compressed. The compressed stream is transmitted to the client through a wireless network. The mobile device client decompresses the stream and renders the 3D video. Meanwhile, the client keeps the server updated with its own viewpoint information as soon as it is changed.

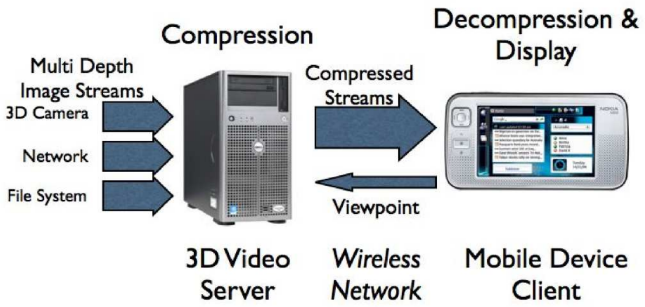

Figure 2: Compression Framework

3. COMPRESSION SCHEME

3.1 Stream Structure

The compressed stream has a layered frame structure. Each frame consists of three layers: a viewpoint layer, a basic layer and an enhanced layer. The viewpoint layer stores the viewpoint information, including the position of the viewer, orientation, view range angle, and display window size. Viewpoint information is received from the client. When the client receives the compressed stream, it compares the information in the viewpoint layer with its current viewpoint. Difference indicates the change of the viewpoint. The basic layer stores a depth image. The image map of the depth image is a pre-rendered view of the 3D scene while the depth map of the depth image contains the depth information with which 3D points can be reconstructed. The enhanced layer includes a set of 3D points with 3D coordinates and color information.

3.2 Basic Flow

The compression flow on the server side has three main steps (Figure 3). In the first step of 3D points reconstruction, depth images of the same time frame in different streams are used to reconstruct 3D points. Each pixel in the depth image corresponds to a point in 3D space. In the second step of 3D points projection, a projection plane is generated according to the viewpoint information of the client. Then all 3D points are projected to the projection plan. The Points invisible to the client can be filtered out. All visible points are collected to generate the basic layer. Invisible points are grouped, compressed mapped to the enhanced layer in the third step of 3D points downsampling. Both layers are combined to form one frame of the compressed stream for transmission. Viewpoint information used in compression process is also added as the viewpoint layer.

The decompression process on the client side also has three steps (Figure 4). The first step is parsing. Stream received from the server is parsed to extract the viewpoint layer, the basic layer and the enhanced layer. If the viewpoint information in the stream remains the same as the current viewpoint, only the basic layer is useful and the image map in the basic layer needs to be parsed and then directly displayed on the screen. Otherwise it indicates that the viewpoint of the client has been changed. Then both the basic layer and the enhanced layer are useful. Position coordinates of all 3D points contained in both layers need to be calculated in the step of 3D point reconstruction and rendered according

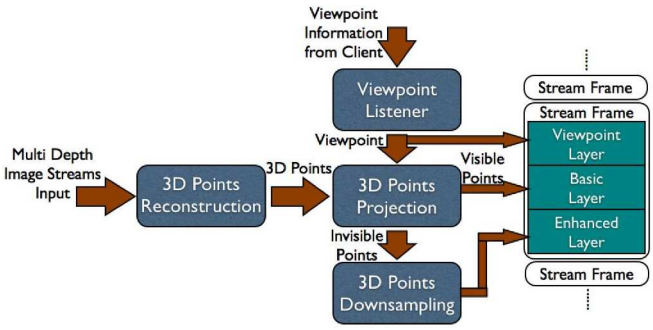

Figure 3: Compression Flow

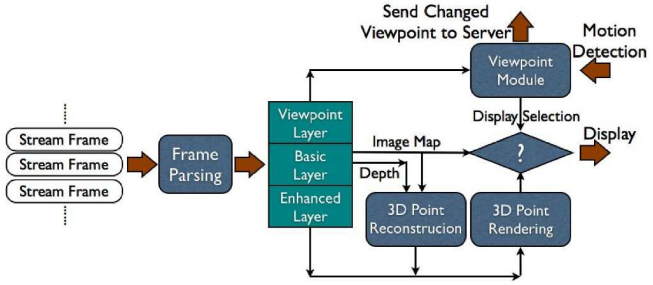

Figure 4: Decompression Flow

to the current viewpoint in the step of 3D point rendering. Meanwhile, the client must inform the server with the updated viewpoint information.

3.3 Techniques

This subsection introduces some detailed technique issues in compression flow.

For the step of 3D points reconstruction in compression flow, reconstructed points are grouped in a set **P** for further compression. Redundancy exists because of the overlap in cameras capturing depth images. But no redundancy removal is handled at this step. There is also no size specified for each 3D point.

For the step of 3D points projection, we use techniques similar to conventional perspective projection and Z buffer testing in computer graphics. With the client's viewpoint information, all 3D points can be rendered on the server to generate the final view to display on the client. First, we create a projection plane matching the display window size and project all 3D points in the set **P** to the plane. Then, we record the projected pixel position and the distance value for each 3D point. Finally, for each pixel in the projection plane which has been projected by one or more 3D points, select the 3D point with the least distance value as the visible point. Pixels in the projection plane are directly exported as the image map of the basic layer, and recorded distances are exported as the depth map.

For the step of 3D points downsampling, it handles all unnecessary points. They won't be shown on the screen of the client device if the viewpoint is not changed. However, if the viewpoint on the client side has been changed and the server hasn't been updated quickly enough, these points may be helpful to approximate the scene from new viewpoint on the client side. Therefore, we still compress these invisible points. First, we divide the whole space into small cubes so that every point falls into one cube. It is similar idea to the block/macroblock concept used in 2D video and image processing. Second, points inside each cube are downsampled. One of the easiest methods of downsampling is to average all 3D points in the same cube. A new point with the average value of the coordinates and color is generated for each cube. New points are grouped as the enhanced layer. One important parameter here is the size of the cube dividing the space. As the cube size increases, more points fall into the same cube leading to a higher downsampling factor and larger compression ratio.

3.4 Further Improvement

Bandwidth requirements can be further reduced with some improvements. The viewpoint layer is not necessarily used in every frame. If the viewpoint information remains the same as the previous frame, it doesn't need to be added to the compressed stream. The basic layer can be further compressed. One purpose of using depth image as the format of the basic layer is that it may take advantage of previous compression techniques. However, decompression of these data may add the computational burden to mobile device clients.

The performance can also be enhanced in the step of 3D points projection on the server side. Since the projection used in the compression scheme is similar to graphics rendering techniques, using OpenGL functions and the graphic card pipeline may greatly accelerate the projection computation. It is especially crucial when one server needs to provide streams for multiple mobile devices clients.

The video quality may be improved by exploring better downsampling method. For example, for the cube with more points, the cubes may be further divided into smaller cubes and then average the points in each sub-cube.

4. PERFORMANCE EVALUATION

Experiments are carried out to test the compression ratio of the compression scheme. The source 3D video, which was recorded in previous TEEVE experiments, has 12 depth image streams. Table 1 compares the average total number of points of one frame before compression and the average number of visible points stored in the basic layer. **gluProject** is used for projection calculation. The data in the table indicates that the compression ratio increases as the display window size decreases. In our experiment, we use JPEG encoding tools to compress both image and depth maps of the basic layer to JPEG images. When the video is recorded at 10 frames per second and the display window size is set to 320*240, the bit rate of the 3D video is only 480Kbps while the original compressed depth image streams in TEEVE requires 56.6Mbps bandwidth. The Table 2 shows the average number of points after downsampling. The compression ratio increases greatly as larger cubes are used.

The computational cost is also evaluated. A workstation with a 2.4Ghz CPU and 4GB memory acts as the server and runs the compression. A Nokia N800 PDA with a 320Mhz ARM CPU decompresses and displays 3D video. On the server side, depth image streams can be compressed at the speed of 6.8 fps. It can be further improved to balance the computation in dual cores. On the client side, the PDA may display the video at 14.5 fps if the viewpoint remains unchanged. When the viewpoint is changed, the frame rate

Table 1: Basic Layer Compression

3D Points Before Compression	Display Window Size	3D Points in Basic Layer	Ratio
	1024*798	26105	3:1
	800*600	17154	4.6:1
78962	640*480	11397	6.9:1
	320*240	3127	25.3:1

Table 2: Enhanced Layer Compression

3D Points Before Compression	Cube Size	3D Points after Downsampling	Ratio
	1*1*1	78917	1:1
	4*4*4	75891	1.1:1
78962	16*16*16	14699	5.4:1
	32*32*32	3956	20:1
	64*64*64	953	82.9:1

decreases significantly to less than 4 fps with the window size of 320*240. It is mainly because when the viewpoint doesn't match, 3D points need to be reconstructed and then rendered again. The experiment shows that on average, it takes about 120 ms for the changed viewpoint information updated in the server. As the 3D video plays at the speed of 10 fps, about 2 frames are compressed with incorrect viewpoint when no buffering issue is considered. Hardware modules for graphics acceleration may help solve the problem. Adaptive frame drop is another scheme to help maintain the QoS when decoding rate fluctuates.

The video quality remains the same when no viewpoint is changed. There is no big difference for the display view to be rendered on the mobile device or on the server (rounding error may cause a little difference). But the quality may degrade when the viewpoint is changed. The quality lost is related with the size of cube used in the step of points downsampling. If the cube size is set to 1, no point is downsampled and the video quality has no degradation. As the cube size increases, higher compression ratio can be achieved but more quality is lost.

5. RELATED WORK

Some researches on computer graphics compression and rendering are related to our work. Chai [1] proposed a depth image compression method using graphic techniques. He discovered a new mesh based representation of depth map. The triangulated depth map takes advantage of graphic card to accelerate rendering. Duguet [2] discusses point-based rendering on mobile devices. Since the display size of mobile devices is very small, point-based rendering techniques can be used to simplify mesh-based graphics rendering. Hoppe [3] studied the view-dependent refinement of progressive mesh rendering. The scene, which is out of the viewpoint, can be greatly simplified and transmitted progressively.

6. CONCLUSION

This paper proposes a new real-time 3D video compression scheme to allow 3D video playback on mobile devices. Due to the limited bandwidth and computational capabilities of mobile devices, the compression scheme is designed to transmit only useful data and reduce the decompression computation workload on mobile devices. This view-dependent compression scheme works only for 3D video compression but not 2D video or 3D graphics, because 2D video does not have the visibility problem and 3D graphics does not update the scene every frame.

7. ACKNOWLEDGMENTS

We thank Ellick Chan for his valuable comments on this paper. This work was supported by the National Science Foundation under Grant CNS 05-20182and CNS 07-20702. Any opinions, finding and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] B.-B. Chai, S. Sethuraman, and H. S. Sawhney. A depth map representation for real-time transmission and view-based rendering of a dynamic 3d scene. In *Proc. of 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT'02)*, pages 107–115, June 2002.

[2] F. Duguet and G. Drettakis. Flexible point-based rendering on mobile devices. *IEEE Trans. on Computer Graphics and Applications*, 24(4):57–63, 2004.

[3] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH*, pages 189–198, 1997.

[4] S.-U. Kum and K. Mayer-Patel. Real-time multidepth stream compression. *ACM Trans. on Multimedia Computing, Communications and Applications (TOMCCAP)*, 1(2):128–150, 2005.

[5] J.-M. Lien, G. Kurillo, and R. Bajcsy. Skeleton-based data compression for multi-camera tele-immersion system. In *Proc. of Advances in Visual Computing, Third International Symposium (ISVC'07)*, pages 714–723, November 2007.

[6] W. Matusik and H. Pfister. 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *ACM Trans. on Graph.*, 23(3):814–824, 2004.

[7] S. K. Penta and P. J. Narayanan. Compression of multiple depth maps for ibr. *The Visual Computer*, 21(8-10):611–618, 2005.

[8] A. Redert and et al. Attest: Advanced three-dimensional television system technologies. In *Proc. of 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT'02)*, pages 313–319, June 2002.

[9] Z. Yang and et al. Teeve: The next generation architecture for tele-immersive environment. In *Proc. of 7th IEEE International Symposium on Multimedia (ISM'05)*, pages 112–119, December 2005.

[10] Z. Yang and et al. Collaborative dancing in tele-immersive environment. In *Proc. of ACM Multimedia (MM'06)(Short Paper)*, pages 723–726, October 2006.

[11] Z. Yang and et al. Real-time 3d video compression for tele-immersive environments. In *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN'06)*, 2006.