Proxy Based Compression of Depth Movies

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computer Science

by

Pooja Verlani
200607019
pooja@research.iiit.ac.in
http://research.iiit.ac.in/~pooja

International Institute of Information Technology
Hyderabad, India
June 2008

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Proxy Based Compression of Depth Movies" by Pooja Verlani, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. P. J. Narayanan

To my Loving Grandfather,
Late Shri T.C. Verlani

Knowing is not enough; we must apply. Willing is not enough; we must do.
-JohannWolfgangvonGoethe

Acknowledgments

I would like to thank Dr. P. J. Narayanan for his support and guidance during the past three years. I was fortunate enough to have him as my guide. I sincerely appreciate all his help, generosity, patience and deep insights over perspective solutions to the problems.

I would also like to thank Dr. C.V. Jawahar, Dr. Anoop Namboodiri and Dr. Jayanthi Siwaswamy for various references and guidance in different subjects related to the stream. I am grateful to P. Sashi kumar for the initial explanations of the concepts. It is under his guidance and by the reading material provided by him, that the beginnings of this thesis have emerged and I am very honored to have him as the co-author of my first paper on real-time rendering. I sincerely thank Naveen for all his support with the preliminaries of the *Proxy-based Compression of Depth Image* project. It has been a pleasure to learn from my seniors, Visesh, Paresh, Jagmohan, Pawan, Vamsi. Their clever comments and witty criticisms have many a times saved me from embarking on a wrong path. I would also like to thank my friends, Neeba, Anand, Avinash, Himanshu, Pradhee and Jyotirmoy for all the support during my Masters degree. The company of such intelligent, vibrant and creative lab mates, always ready to talk, argue, and shoot down ideas, was certainly one of the best things about my entire graduate degree.

Finally, I would like to appreciate the support and patience from my parents and sister. I owe deep gratitude to all my friends and family members.

Abstract

Sensors for 3D data are common today. These include multicamera systems, laser range scanners, etc. Some of them are suitable for the real-time capture of the shape and appearance of dynamic events. The $2\frac{1}{2}$D model of aligned depth map and image, called a Depth Image, has been popular for Image Based Modeling and Rendering (IBMR). Capturing the $2\frac{1}{2}$D geometric structure and photometric appearance of dynamic scenes is possible today. Time varying depth and image sequences, called Depth Movies, can extend IBMR to dynamic events. The captured event contains aligned sequences of depth maps and textures and are often streamed to a distant location for immersive viewing. The applications of such systems include virtual-space tele-conferencing, remote 3D immersion, 3D entertainment, etc. We study a client-server model for tele-immersion where captured or stored depth movies from a server is sent to multiple, remote clients on demand. Depth movies consist of dynamic depth maps and texture maps. Multiview image compression and video compression have been studied earlier, but there has been no study about dynamic depth map compression. This thesis contributes towards dynamic depth map compression for efficient transmission in a server-client 3D teleimmersive environment. The dynamic depth maps data is heavy and need efficient compression schemes. Immersive applications requires time-varying sequences of depth images from multiple cameras to be encoded and transmitted. At the remote site of the system, the 3D scene is generated back by rendering the whole scene. Thus, depth movies of a generic 3D scene from multiple cameras become very heavy to be sent over network considering the available bandwidth. This thesis presents a scheme to compress depth movies of human actors using a parametric proxy model for the underlying action. We use a generic articulated human model as the proxy to represent the human in action and the various joint angles of the model to parametrize the proxy for each time instant. The proxy represents a common prediction of the scene structure. The difference between the captured depth and the depth of the proxy is called as the residue and is used to represent the scene exploiting the spatial coherence. A few variations of this algorithm are presented in this thesis. We experimented with bit-wise compression of the residues and analyzed the quality of the generated 3D scene. Differences in residues across time are used to exploit temporal coherence. Intra-frame coded frames and difference-coded frames provide random access and high compression. We show results on several synthetic and real actions to demonstrate the compression ratio and resulting quality using a depth-based rendering of the decoded scene. The performance achieved is quite impressive. We present the articulation fitting tool, the compression module with different algorithms and the server-client system with several variants for the user. The thesis first explains the concepts about 3D reconstruction by image based rendering and modeling, compressing such 3D representations, teleconferencing, later we proceed towards the concept of depth images and movies, followed by the main algorithms, examples, experiments and results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sensors for 3D scenes and objects are common today. These include multi-camera systems, laser range scanners, etc. Some of them are suitable for the real-time capture of the shape and appearance of dynamic events. A simple method for capturing 3D structure of a scene is as depth measurements taken from a point, called a *depth map*. A depth map is a two-dimensional array where the x and y distance information corresponds to the rows and columns of the array as in an ordinary image, and the corresponding depth readings (z values) are stored in the array's elements (pixels). Time varying captured sequence of depth maps from a camera is called a *depth movie*. Depth Movies captured from multiple views can help visualizing the 3D aspect of a typical scene.

The views captured from different viewpoints find potential applications in Image Based Rendering (IBR). IBR aims at capturing 3D environments using a number of cameras that recover the geometric and photometric structure from the scene. IBR renders novel views using the captured views, to give the impression of a 3D scene with only a few acquired views around the scene. These views can be transmitted to a remote location for tele-immersive environments.

Tele-immersion is a new medium of human interaction that creates the illusion that a user is in the same physical space of the other participants, although in reality other participants may be miles away. This technology combines the concepts of scene capture and virtual reality with collaboration technology using the different media.

1.1 Image Based Rendering

Image based rendering methods take a set of 2D images of a scene and generate its novel views from different camera positions. IBR tends to derive a representation from the captured 2D streams and uses this for rendering. The principal advantage of IBR is that the representation need not be as comprehensive as a graphics model. To be able to look into various directions in a 3D scene using IBR, we do not need to take photographs of the scene from all directions and produce a panorama. To reduce the number of images necessary for novel views, IBR aims at deriving geometric representations of the scene through image correspondence, interactive photogrammetry, or active sensing, and then render this geometry from the desired novel viewpoint with colors projected on from the original photographs. Image based rendering techniques fall into two main categories: geometry based and light field based. Light field techniques do not necessarily need geometry information. They think of the scene as a space of rays, a portion of which are recorded by cameras. By re-sampling the recorded rays according to the geometry of the virtual camera we can reconstruct the image. Geometry information helps to improve the quality of the image, although it requires considerable preprocessing. Geometry based approaches transfer input images

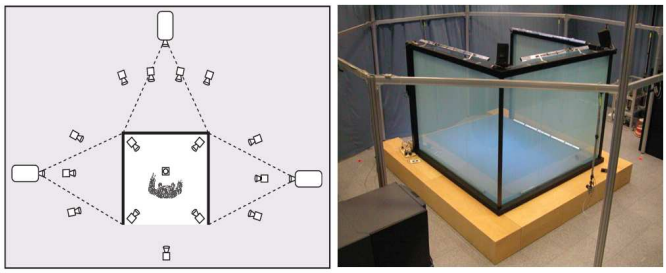

Figure 1.1: A view of Blue-C capture system, [31]

to the virtual camera through the use of scene geometry, which can be in the form of per-pixel depth or polygonal models.

Depth or disparity is powerful information that can be used like geometry for image-based rendering. Depth is calculated from stereo vision or using other sensors like range-finders or sonars and then combined with color information provided by images to form an image-based model. Depth image based rendering (DIBR) can produce novel views from different (new) viewpoints, based on a single two-dimensional (2D) image and its corresponding depth map. DIBR and IBR can effectively be used to render the scene structure at remote locations, a popularly known concept of tele-immersion.

1.2 Capturing a dynamic scene

With the growing need for capturing, transmitting and rendering dynamic scenes, 3D visualization and tele-immersion systems, different camera-hardware setups have been built by various labs in recent past. In order to capture dynamic scenes and visualize them at the remote location, we need to capture multiple views of the scene from different cameras around the space. Issues that relate to multicamera systems are the calibration and the synchronization of the cameras. Typically, multicamera calibration is based on solving the correspondence problem for multiple cameras to estimate their parameters. For example, Blue-C [31], an immersive display system, also acquires 2D streams of a scene in a similar manner as shown in Figure 1.1.

1.3 Depth Images as Scene Structure

The depth map is a two-dimensional array of depth values, with location (i, j) storing the depth or normal distance to the point that projects to pixel (i, j) in the image. So, depth maps contain distances to points organized on a regular, 2D sampling grid. Each depth map can be considered as an image. The corresponding location (i, j) of the image stores the color from that ray. Both image and its corresponding depth map are acquired from the same point of view of a camera. Computer vision provides various methods to compute such structure of points visible in a view, called the $2\frac{1}{2}$D structure, using different clues from images. Motion, shading, focus, inter-reflections, etc., have been used to this end, but stereo has been most popular. Traditional stereo tries to locate points in multiple views that are projections of the same world point. Triangulation gives the 3D structure of a point after identifying it in more than one view. Volumetric methods map each world voxel to the views in which it is visible. Visual consistency across these cameras establishes the voxel as part of a visible, opaque surface. Recovering such geometric structure of the scene from multiple cameras can be done reliably today using stereo. Range scanners using lasers, structured lighting, etc., can also be used to detect structure. Figure 1.2 and 1.3 shows images and depth

maps from different viewpoints, with the points that are closer shown brighter than the farther ones.

The depth map gives the Z-coordinates for a regularly sampled X-Y grid coinciding with pixel grid of the camera. Combined with camera calibration parameters, this represents the 3D structure of all points visible from the camera location as a point cloud. Grouping of points into higher level structures such as polygons and objects is not available and doesn't have to be inferred.

To capture depth from various viewpoints, a setup of **m** cameras around the scene can be used, in a similar fashion as discussed in section 1.2. The depth and texture from one viewpoint represent local, partial structure of the scene, i.e., parts visible from a point in space with a limited view volume. The entire scene space can be captured using multiple, distributed depth maps and textures. It is possible to merge these partial models into a single global structure using methods like, mesh stitching volumetric merging etc.

1.4 Depth Movies

Time varying sequences of the Depth Images are called *Depth Movies*. Real-time capture of Depth Movies is possible today. Depth movies are feasible scene representation for capturing and streaming data for true 3D teleconferencing.

A depth movie is a sequence of aligned combination of depth map and image. Each pair corresponds to a time-instant and the sequence progresses across time. Depth movies can be thought of as having 3 channels:

1. **D Channel:** Consists of a depth map sequence D_t with $D_k[i, j]$ giving the depth or distance at pixel (i, j) at time instant k.

2. **I Channel:** Consists of an image sequence I_t with $I_k[i, j]$ giving the colour at pixel (i, j) at time instant k.

3. **C Channel:** Consisting of a sequence C_t of time varying calibration parameters of the scanner or the camera. C_k gives the 3×4 matrix that maps a world points $\mathbf{P}$ to an image point $\mathbf{p}$ using $\mathbf{p} \approx C_k\mathbf{P}$. In practice, the sampling of C_t along the time axis is sparse as the calibration parameters typically change slowly, if at all.

Optionally, the C_t and D_t channels can be combined into a **P Channel** of 3D point sequence S_t with $S_t[i, j]$ giving the 3D coordinates (x, y, z) of the points projecting to the pixel position (i, j). We use the explicit notation using $\mathbf{D}, \mathbf{I}$, and $\mathbf{C}$ channels for its brevity. The $\mathbf{C}$ channel can be compressed considerably as the calibration parameters of a camera is likely to change very slowly, if at all, as noted above. The calibration parameters carry the information about the geometry of depth-capture which could be useful for the applications that use the depth maps.

The depth movie datasets have depth images for each of the frame in the sequence captured from multiple views. Since the multi-stream depth images are huge in size, they need a compressed representation for 3D teleconferencing systems.

1.5 Tele-immersion

Tele-immersive environments are now emerging as the next generation of communication medium to allow distributed users more effective interaction and collaboration in joint activities. A basic scheme for Tele-immersion is as shown in Figure 1.5. Tele-immersion systems can have one of the two basic approaches. The first approach considers that tele-immersion is three dimensional, avatar-like

Figure 1.2: Texture (top) and depth map (bottom) of a scene at different time instant and from different viewpoint. [1(

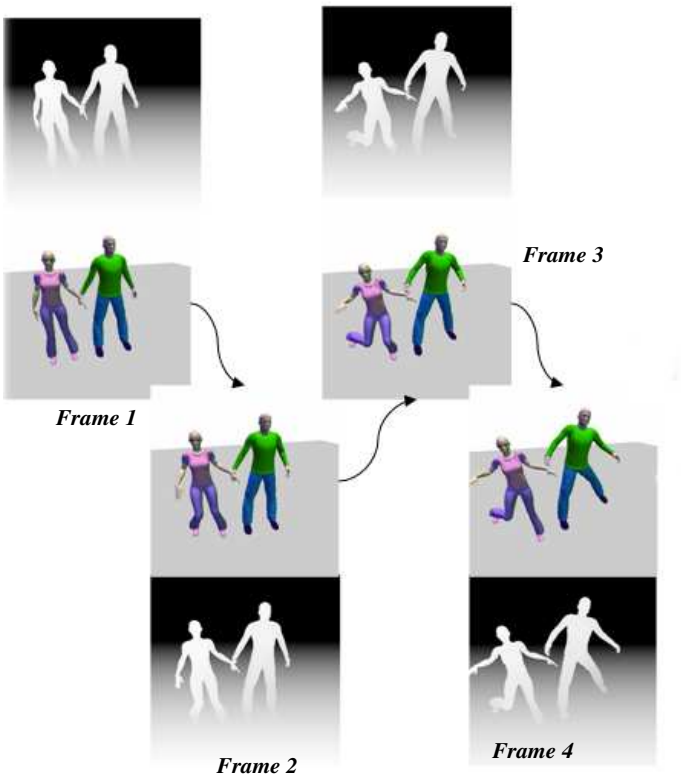

Figure 1.3: Frames of a Depth Movie, showing texture movie along with it.

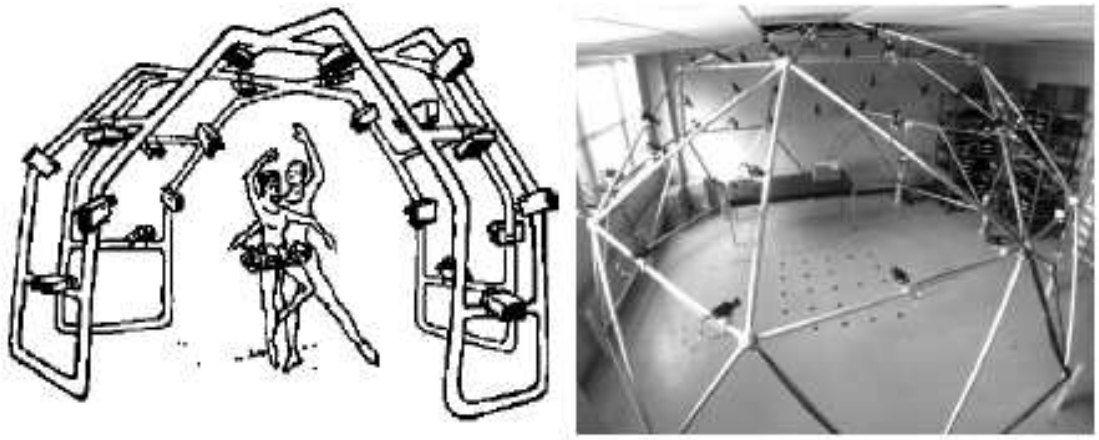

Figure 1.4: Conceptual and actual Dome to capture dynamic events from Virtualized Reality, [39, 64]

model using graphical representation of the participants as animations. The second approach treats tele-immersion as a representation of virtualized reality based on 3D reconstruction of real people and the rest of the scene. The second approach is less restrictive and is widely being researched by various research groups.

The three major components of a tele-immersion system are scene acquisition, 3D reconstruction, transmission, and rendering. Figure 1.5 shows a block diagram of these components to each other and the overall system. For effective interactive operation, these three components must accomplish their tasks in real-time. Accordingly, a tele-immersive environment requires several basic components, including a 3D camera array and sound system, a communication network, and a rendering system. Recently, several researchers have developed and experimented with individual components or tele-immersive environments with partially integrated components. We have concentrated our thesis on the use of depth movies, their efficient compression for a real-time transmission through network and reconstructing it back at the client's remote site.

1.6 The Problem

Depth maps are bulky as they store high precision depth values. As a result, multistream depth movies require effective representations and compression methods for transmission and 3D playback at the remote site. The server at the capture site is linked over a network to a client at the rendering site. Video compression is suitable for texture images. Compression of multistream depth movies of human actors is the focus of this thesis. The **D** channel of the depth movie is a video of depth values and it may appear that compression schemes like MPEG would work well. MPEG compression is psycho-visually motivated and gives less emphasis to the high frequency components. However, the high frequency regions of depth images represent occlusion boundaries which are critical for depth maps, especially as it has to be rendered at the remote client. Lossy compression of depth movies is at the cost of either changing distance or shape of the scene components. Thus, developing methods for lossless compression and effective transmission of these becomes the key challenge. We present algorithms to compress depth movies involving human actors using a common parametric proxy model.

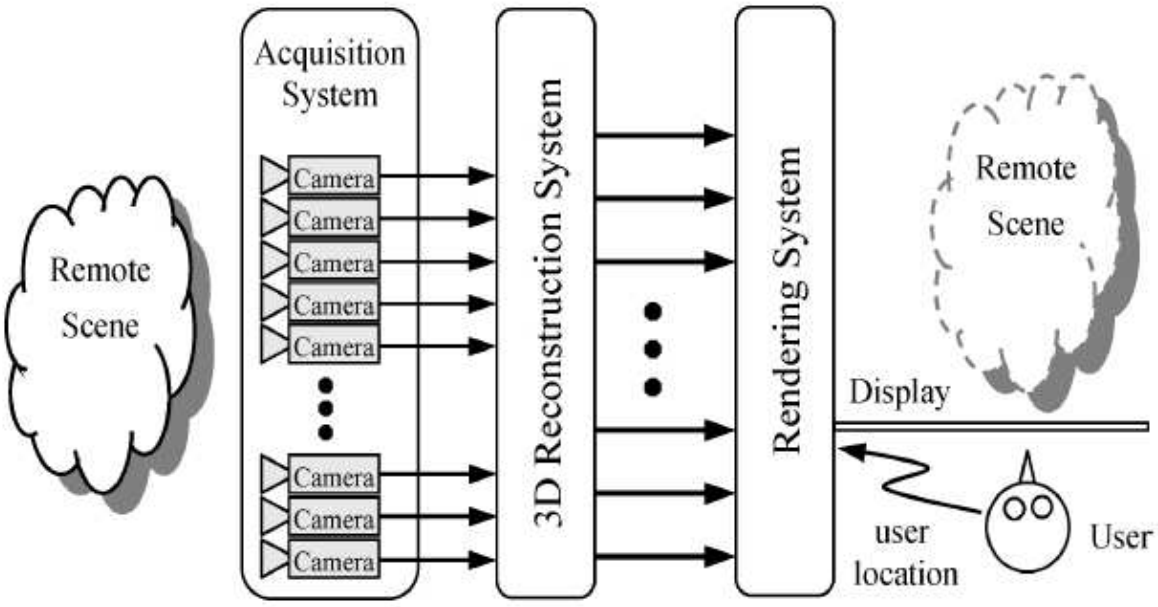

Figure 1.5: Components of a Tele-immersion system showing the communication link between two remote sites. [44]

1.6.1 Varying depth maps Proxy Based Compression

A general world scene can be assumed to comprise of moving humans, rigid static and dynamic objects, non-rigid objects, non-rigid dynamic objects, etc. We consider scenes with humans as the main subjects for this thesis. **m** cameras are set around the scene for capturing these scenes. Depth movies of **n** frames are generated from each view. For each of the **n** frames we have **m** depth-maps and **m** textured-images for the **m** camera views. We use a generic articulated human model as a proxy and its various joint angles as parameters for each frame representing the common prediction of that scene at that instant. The time varying parameters approximate the underlying geometric structure of the action such that it is independent of the viewpoint. The proxy depth map is captured by projecting the depth on the **m** cameras. The difference between the captured depth map and the proxy depth map, known as "residue", exploits the inter-view spatial coherence. Differences in residues across time are used to exploit temporal coherence. Intra-frame bitwise coded frames and difference coded frames provide high compression and facilitate random access in a depth codec, on the same lines as a MPEG codec.

1.6.2 Server-Client System

Figure 1.6 presents the overview of compressing a scene using parametric proxies. The input to the system is a 3D scene comprising multistream depth maps and texture maps of a human performing some action. A standard articulated human model is used as the parametric proxy. The first task is to fit a proxy to the point cloud for each frame. This proxy is known as the prediction of the human in the scene. The depth maps of the fitted proxy for each frame are captured. The prediction error between the original and the fitted proxy model is calculated by subtracting the predicted proxy from the input depth maps. These "residues" are encoded and sent to the client. At the client's side, the parameters are applied on the articulated human to capture the proxy

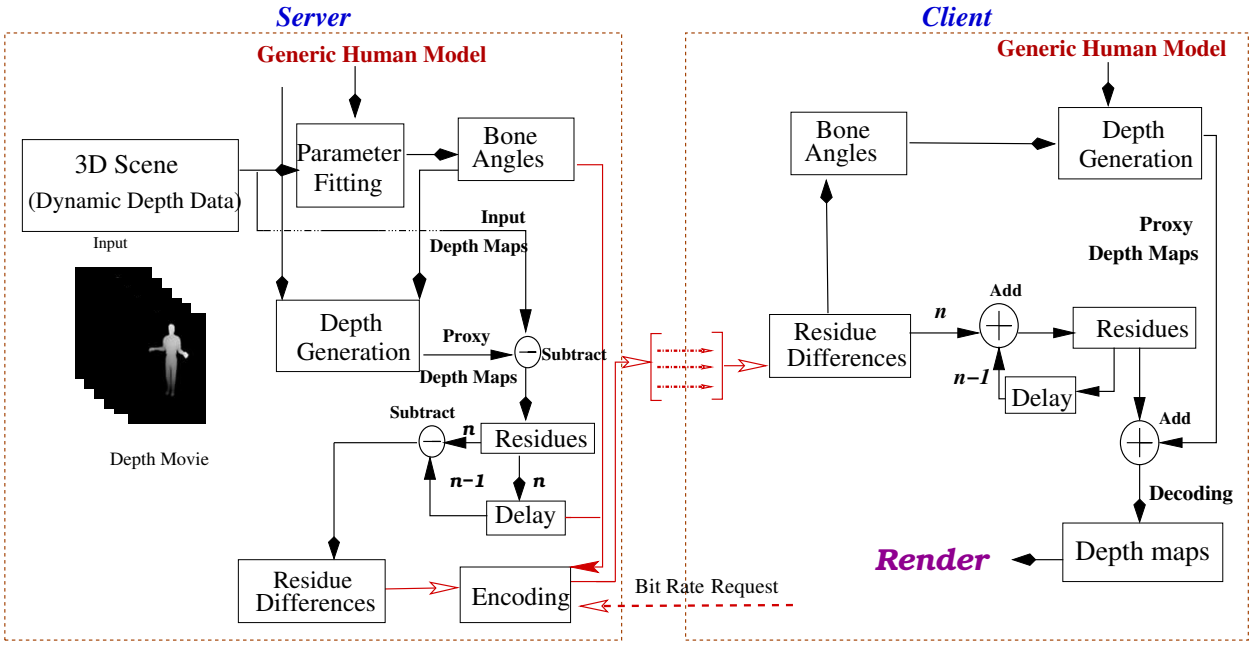

Figure 1.6: Basic scheme for our procedure

depth maps. Residues are added to these depth maps to get the real depth for rendering.

A compressed packet is sent to the client as per the demands of compression factor and quality, and the available network bandwidth. At the client's end, the original $2\frac{1}{2}$D representation is recovered from the proxy parameters and is applied on the common proxy available at the client's site. The residues are decoded and the $2\frac{1}{2}$D representation is used for rendering the 3D scene in real-time. Thus, client controllable compression can be adopted by the system and rendering be done in real-time at the client. This makes the technique ideal for 3D teleconferencing and remote immersion systems.

We used three variations of the experimental datasets: Real data, MOCAP data with meaningful actions, and synthetic action. Different strategies and implementations have been done to carry out these variations of the datasets. The second and third datasets are produced using POSER to give an animated human feel.

1.7 Applications

Our compression scheme is useful to a server-client based tele-immersion. Tele-immersive environments have potential to significantly change educational, scientific, corporate and manufacturing standards. Some of the possible applications can be remote education, long distance corporate meetings, virtual experiment labs, surveillance, etc. Tele-immersion can promote the concept of virtual classrooms. Where students can sit miles away from the professors but still get the feel of the class and grasp concepts while asking doubts in real-time. A good example of such a system is "Electronic Books for Tele-immersion Age" lead by Brown University and University of North Carolina, that provides surgeons to train for different surgeries and operations remotely. Tele-immersion can improve the everyday graphical display environments, and 3D tele-immersion capabilities that allow distant people to feel as though they are together in a shared office space. One such work can be the "Office of the Future" project being worked on at University of North Carolina (Chapel Hill) [71] .

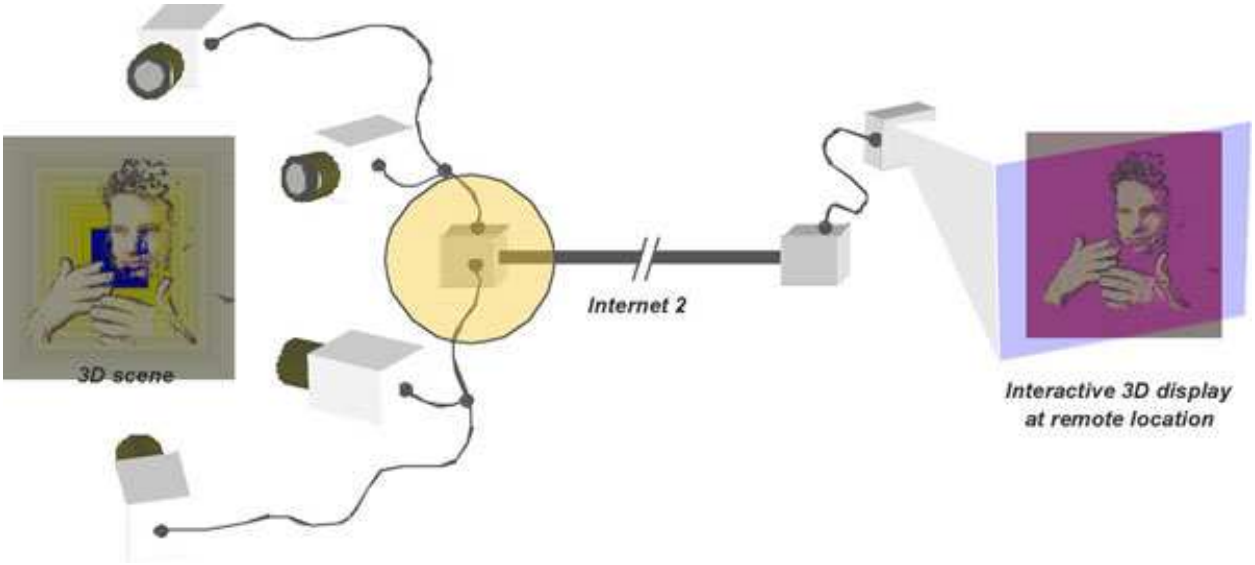

Figure 1.7: A systematic diagram of a teleconferencing system developed at UIUC.

The concept can also be used in research labs where extensive research is carried on by different research groups that are miles apart but can together work on experiments and derive new analogies by sharing a virtual research space. This way more idea and research brains can be put to a better use for innovations required in daily life. Also, an improved remote surveillance concept can be achieved using tele-immersion and 3D tele-conferencing. An example of tele-immersive system as used by UIUC is shown in Figure 1.7.

1.8 Contributions

This thesis contributes towards efficient representation and compression of depth movies or time-varying depth maps, using a parametric proxy, which has not been studied much. This thesis extends the idea proxy-based compression of multiple depth maps to multiple time-varying depth maps. We also analyze the different options for compression of such data and show results on real and synthetic data. We achieve impressive compression at acceptable quality levels on many synthetic and real data.

1.9 Organization of the thesis

The main focus of the thesis is efficient compression of depth movies. We have proposed a parametric proxy based compression method for compressing depth movies. Different experiments, analysis and variants of the parametric proxy compression has been presented in the following chapters.

- Chapter 2 presents a detailed review of the previous work. A review of different works on image based modeling and rendering using depth images has been presented. We also present the various methods initiated by different researchers on compressing depth images and multiview reconstruction of such scenes. A number of concepts relating parametrization of a 3D scene and human model for the articulation for the parametric model have also been discussed.

- In chapter 3, we give a brief review of the methods used for compressing depth images with proxy-based compression.
- In chapter 4, we introduce the concept of depth movies and their capture process. We present our experimental work for compressing such depth movies using the pre-existing methods like JPEG, MPEG, quadtrees, etc. We draw different analogies to promote the need and requirement for the parametric proxy compression to be discussed in the next chapter.
- In chapter 5, we define and discuss our proxy based compression method for depth movies. We have defined the parametrized articulated proxy model along with the fitting tool to get the proxy for each frame of the depth movie. We have drawn a server client model using the various compression algorithms for encoding the multiview depth movies.
- In chapter 6, the experimental setup is explained in detail. The results on these datasets are presented through graphs and tables. For MOCAP datasets, results are included by varying noise levels and proxy details. Finally, we analyze our results for the real-time tele-immersion.
- In chapter 7, we draw a conclusion from the work and give application specifications of the work.

Chapter 2

Related Work

2.1 Image Based Rendering

Image-based modeling and rendering techniques have gained much attention as an alternative to traditional geometry-based techniques for image synthesis. Instead of geometric primitives, a collection of sample images are used to render novel views. Shum et. al [77] reviewed earlier work on image-based rendering (IBR) which reveals a continuum of image-based representations [46] based on the trade off between the number of images needed and information of scene geometry. The various rendering techniques (and their associated representations) can be vaguely categorized into those with no geometry and those with implicit geometry.

On one end of the rendering spectrum, traditional texture mapping relies on very accurate geometric models but only a few images. In a general IBR system with depth maps, such as 3D warping [54], layered-depth images (LDI) [76], LDI tree [18], etc., the model consists of a set of images of a scene and their associated depth maps. When depth is available for every point in an image, the image can be rendered from any nearby point of view by projecting the pixels of the image to their proper 3D locations and re-projecting them onto a new view. Unlike for synthetic environments, estimating the depth information from real images has had limited success even for the state-of-art computer vision algorithms.

Some image-based rendering systems do not require explicit geometric models and instead require feature (such as points) correspondence between images. For example, Chen et. al [19] introduced *View Interpolation* that generates novel views by interpolating optical flow between corresponding points. On the other hand, *View Morphing* by Sietz et. al [75] generates in-between camera matrices along the line of two original camera centers, based on point correspondences. Computer vision techniques are usually used for generating such correspondences. At the other extreme, lightfield rendering uses many images but does not require any geometric information or correspondence. Levoy et. al [48] introduced *Lightfield Rendering* that generates a new image of a scene by appropriately filtering and interpolating a pre-acquired set of samples. *Lumigraph* by Gortler et.al [28] is similar to lightfield rendering but uses approximate geometry to compensate for non-uniform sampling in order to improve rendering performance. Shum et. al [78] devised the concentric mosaics representation which reduces the amount of data by capturing a sequence of images along a circular path. Lightfield rendering, however, tends to oversample to counter aliasing effects. Oversampling means more intensive data acquisition, more storage, and more redundancy. The optimal number of images required for unaliased rendering is critical to all IBR systems. Finding a solution to this problem is difficult as it involves unraveling the relationship among three elements: the depth and texture information of the scene, the number of sample images, and the rendering resolution. The

Figure 2.1: 3DTV :: Left: Array of 16 cameras and projectors. Middle: Rear-projection 3D display with double-lenticular screen. Right: Front-projection 3D display with single-lenticular screen.

solution should provide design principles for image based rendering systems in terms of trade-off between the images and the geometry information needed.

2.2 3D Capturing Systems

Various labs have used variants of camera arrangements for capturing 3D. Since 1995, CMU has a 3D room for capturing 3D information of any subject in the scene as shown in Figure 2.2. Also as shown in Figure 1.4, the "3D room" is a facility for 4D digitization i.e., capturing and modeling a real time-varying 3D event, into a computer. On the walls and the ceiling of the room 49 cameras are mounted, all of which are synchronized with a common signal. A PC-cluster of 17 computer systems digitizes all the video signals from the cameras simultaneously in real time as uncompressed and no loss full frame images with color. Narayanan et. al [64] designed the system that was initially based on multi-baseline dense depth map computation. Its recent version by Cheung et. al [20] is based on visual hull computation using silhouette carving and has been commercialized by Billinghurst et. al [10].

Some of the recent significant multicamera systems that relate to 3DTV, 3D reconstruction, telepresence and teleconferencing, acquire 3D scene sequences and send them over network, as these applications intrinsically require live video feeds. Examples are CMUs new 3D room [20], the view-dependent visual hull system at MIT [55], the multicamera systems at the Keck laboratory at the University of Maryland [9] (Figure 2.4)and the Argus system at Duke University. Kauff et. al [40] designed a teleconferencing system that captures a scene with four cameras mounted around a display. There are a few other hardwares setup for multiview captures. Like, Free-viewpoint video (FVV) captures using the notion of 3DVO (3D Video Objects), as shown in Figure 2.5, where a 3D Video object is captured in a relatively sparse dome view configuration. Similarly, 3D video recorder [95], where 2D video streams are recorded from several synchronized digital video cameras and are stored as pre-processed images to the disk. Blue-C [31], an immersive display system, also acquires 2D streams of a scene in a much similar manner as shown in Figure 1.1.

The Stanford multi-camera array [91] is an architecture specialized for facilitating the lightfield rendering. Mulligan et. al [61] introduced a system (Figure 2.3) which first pioneered the use of a large number of video streams to provide a real-time multiview reconstruction. In 2004, Matusik

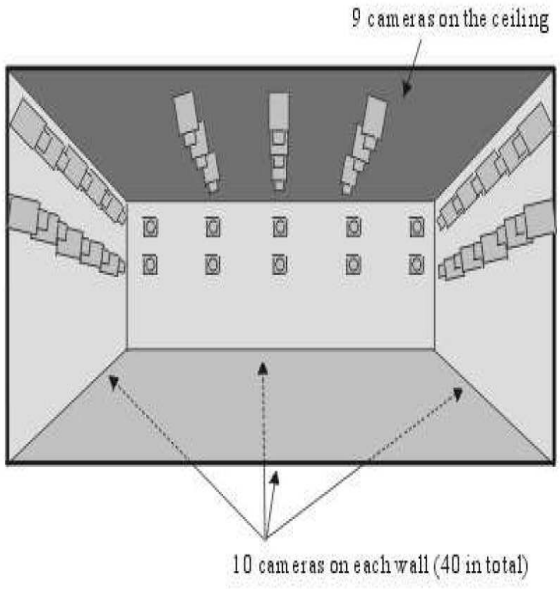

Figure 2.2: 3DRoom at CMU.

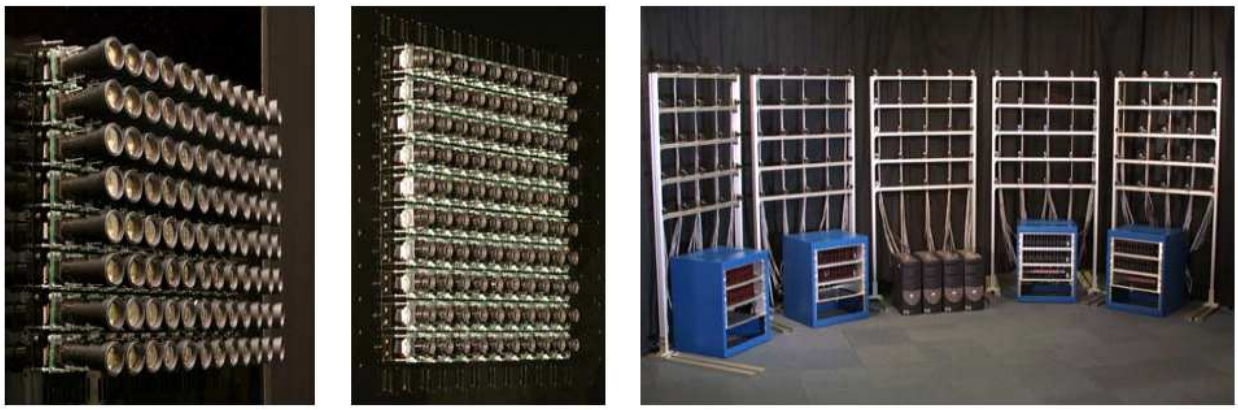

Figure 2.3: Different configurations of camera arrays setup at Stanford University, [92]

et. al developed a system of an array of 16 cameras and projectors as shown in Figure 2.1. This 3DTV [56] system allowed real-time acquisition, transmission, and 3D display of dynamic scenes. The system consists of an array of cameras, clusters of network-connected PCs, and a multi-projector 3D display with a lenticular screen. The display provides stereoscopic color images from multiple view points without glasses. Instead of designing perfect display optics, cameras are used for the automatic adjustment of the 3D display. In 2005, Baker et. al [8] (Figure 2.6) produced synthetic views using 5 streams based on the visual hull method.

2.3 Depth Map Based Representation

The fundamental representation of a single point in 3D space uses a vector of three dimensions (or four dimensions in homogeneous coordinates). The camera distances (depth) of the scene point, whose projections give the pixel locations on the image, are essential to render an arbitrary view of the scene. Therefore, it is better to examine, not a single point, but a regular dense-depth

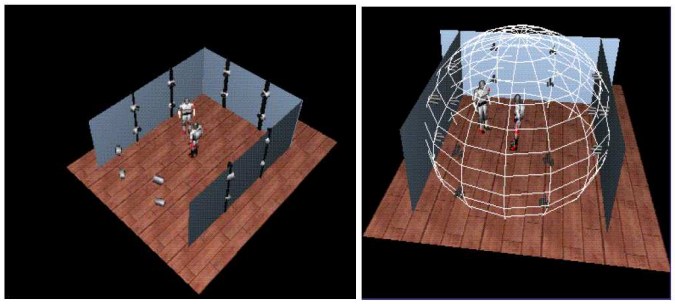

Figure 2.4: Camera setup in rectangular room and a dome-shaped room. [9]

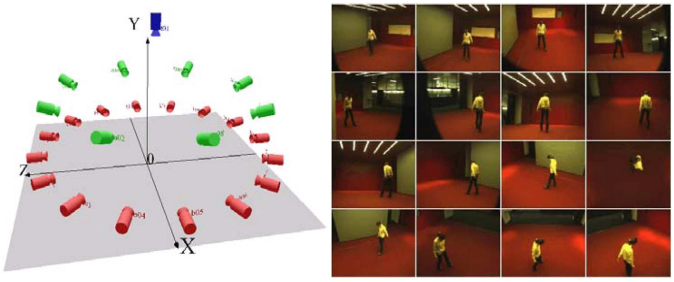

Figure 2.5: Capturing hardware for 3D Video objects.

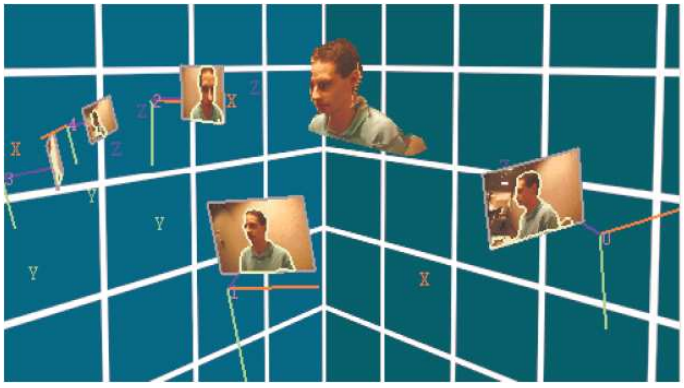

Figure 2.6: View of User in Coliseum space from 5 cameras, [8]

representation of a scene. The distances of the points in a 3D scene from the camera are stored in a matrix defined by the reference image of the scene and is denoted as a *depth map*. The depth map is considered a $2\frac{1}{2}$D representation of a 3D scene from a camera view.

2.3.1 Depth Representation

Shade et al. proposed the concept of layered depth images (LDI) [76], in which a 3D object (or a scene) is represented by a number of views with associated depth maps. Using appropriate scaling and information from camera calibration, it is possible to render virtual intermediate views. The quality of the rendered views and the possible range of navigation depend on the number of original views and camera settings. In case of simple camera configurations (such as a conventional stereo-rig or a multi-baseline video system), LDI can even be utilized for fully automatic real-time depth reconstruction in 3D video or 3DTV applications, which could be denoted as depth image-based rendering [23, 24].

LDI represents an efficient and attractive alternative to 3D mesh representations of scenes. A rendering format for LDI is included in the recent computer graphics extension of MPEG-4, Animation Framework eXtension (AFX) [13]. Using AFX, Smolic et. al [81] made it easy to use LDI in a standardized way. The 3DAV group of MPEG is investigating LDI as a standard format for 3DTV applications [99].

The representation of a 3D scene by dense depth map(s) will face a bandwidth problem in 3D teleimmersion system when delivered over limited bandwidth channels. Hence, this information should be optimally represented and compressed by minimizing both its rate and distortion together. The conventional strategies encode the available depth by lossy image or video compression methods [81].

The multiview dense depth maps can efficiently produce 3D replica of real scenes. They represent the whole scene with point samples, making no distinction between separate objects. Hence, they are easy to construct and space-efficient but incapable of modeling the scene semantics. Graphical realism, progressive modeling, level of detail scalability and animation are fundamental functionalities which are hard to achieve using dense depth representations.

2.4 Depth Image Based Rendering

Depth Image-Based Rendering (DIBR) is the process of synthesizing novel views of a scene from still or moving color images and associated per-pixel depth information [57, 53]. Conceptually, this novel view generation can be understood as a two-step process: First, the original image points are reprojected into the 3D world, utilizing the respective depth data. Thereafter, these 3D space points are projected onto the image plane of a camera, which is located at the required viewing position. The concatenation of re-projection (2D-to-3D) and subsequent projection (3D- to-2D) is usually called 3D image warping in the Computer Graphics literature.

McMillan and Bishop [58] proposed a method to render a scene from new viewpoints by warping the depth image (i.e., an image with color and depth information). One major problem with this method is dis-occlusion artifacts caused when a portion of the scene not visible in the depth image is visible from the new viewpoint. Using multiple depth images from multiple viewpoints can reduce these dis-occlusion artifacts. Layered Depth Images (LDI) merge multiple depth images into a single depth image by keeping multiple depth values per pixel [76]. However, the fixed resolution of an LDI imposes limits on sampling multiple depth images. An LDI tree, an octree with a single LDI in each node, can be used to overcome this limitation [18]. Grossman and Dally [32] create

multiple depth images to model an arbitrary synthetic object. The depth images are divided into 8×8 blocks and redundant blocks are removed. QSplat, used by Rusinkiewicz et. al [72] uses a bounding sphere hierarchy to group 3D scanned points for real-time progressive rendering of large models. Pfister et. al [68] used Surfels that represent objects using a tree of three orthogonal LDIs called a Layered Depth Cube (LDC) tree.

2.5 Real-Time Depth Image Based Rendering using GPUs

The process of rendering depth images is summarized below. Depth images can be rendered using splatting or implied triangulation. Splatting treats each depth/color combination as a 3D point with a certain size in the world or the image. Implied triangulation imposes a triangle-grid structure on the raster-ordered depth or color pairs and draws them using standard graphics hardware. The triangles on the depth discontinuities have a large difference in depth along some of their edges and are not drawn. Depth discontinuities can result in holes in the rendered views. These can be filled by rendering using another depth image which sees that part of the scene. When multiple DIs are rendered, they should be blended when representing the same scene region. Thus, a representation consisting of multiple depth images can provide a complete representation that can use standard graphics algorithms for view generation.

The algorithm to render and blend the set of DIs is given below [63]. The optical axis of the new view is given by $\mathbf{n}$ and that of $\mathbf{DI}_i$ is given by $\mathbf{n}_i$.

for each depth image $\mathbf{DI}_i$ **do**

1. If $(\mathbf{n} \cdot \mathbf{n}_i \leq 0)$ skip i.
2. Generate the new view using $\mathbf{D}_i$ and $\mathbf{I}_i$.
3. Read back image to $\mathbf{I}'_i$ and the depth buffer to $\mathbf{Z}'_i$.

end for

for each pixel $\mathbf{p}$ in the new view **do**

4. Compare the $\mathbf{Z}'_i(p)$ values $\forall i$.
5. Keep the views within a threshold Δz of the nearest z value.
6. Compute the angle θ_i at the 3D point of p between the ray from DI i and the novel view. Compute the weight $w_i(p) = f(\theta_i)$ as a function of the angle.
7. Assign $\sum_i w_i I'_i(p)$ as the colour of the novel view pixel $\mathbf{p}$.

End for

It should be noted that a different combination of DIs are blended for each pixel of the new view, based on the visibility and angle of each DI at that point [63]. The algorithm involves reading the depth and image buffers back and performing the blending on the CPU. These are expensive operations and hence real-time rendering was not achieved. The algorithm was able to render a frame every 2-3 seconds on an AMD64 machine with 1GB RAM and an nVidia 6600GT graphics card with 128MB of video RAM. The synthetic scene used for the performance figures, similar to those given in Figure 2.9, was represented using 20 depth images with ten each located on a circle at two different heights and pointed inwards towards the scene.

The weighting function f ensures that the effect of a particular DI falls smoothly with novel view position. This avoids abrupt changes in color values that can result if multiple DIs have different gains and offsets for their images. Weighting functions like $\cos^k \theta$ or $e^{-k\theta}$ work for values of k of 2 or 3.

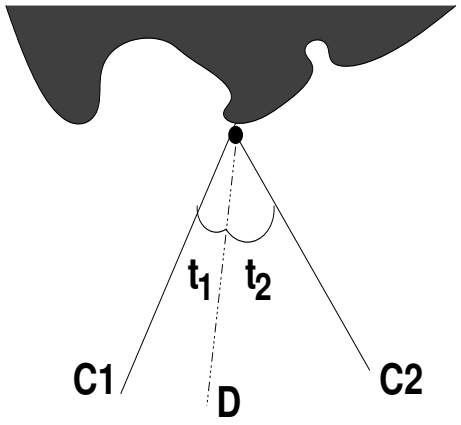

Figure 2.7: Rendered images from D using depth images C1 and C2 are blended based on the angles t_1 and t_2

2.5.1 GPU Rendering of Depth Images

The read back of the frame-buffer is the time consuming operation in the above algorithm. The modern GPUs have a huge memory and computation power. If the read back is avoided and the blending is done in the GPU, the frame rate can possibly reach interactive rates.

Verlani et. al [90] devised a 2-pass algorithm to render multiple DIs with per-pixel blending. The first pass determines which views need to be blended for each pixel and the second pass actually blends them. The property of each pixel blending a different set of DIs is maintained by the new algorithm. The overview of the algorithm is given in Figure 2.8.

Pass 1:

1. Enable z-buffering, disable lights, shading.
2. Clear depths.
3. **for** each Depth Image $\mathbf{DI}_i$ **do**
   (a) If $(\mathbf{n} \cdot \mathbf{n}_i \leq 0)$ skip i.
   (b) Render $\mathbf{D}_i$. Offset each point by Δz away from the novel view camera
   end for

Pass 2:

4. Enable lighting, shading, z-test. Disable z modification.
5. Clear color buffers RGBA.
6. **for** each Depth Image $\mathbf{DI}_i$ **do**
   (a) If $(\mathbf{n} \cdot \mathbf{n}_i \leq 0)$ skip i.
   (b) Render $\mathbf{D}_i$ and $\mathbf{I}_i$ to new view normally.
   (c) At each frame-buffer pixel p, compute the angle θ_i between DI_i and novel view and the weight $w = f(\theta_i)$.
   (d) Set color $c(p)$ at p to $(A(p)c(p) + wI'_i(p))/(A(p) + w)$ where $I'_i(p)$ is the color from rendering the DI i.
   (e) $A(p) = A(p) + w$
   (f) Leave the image in the buffer for next DI.
   end for

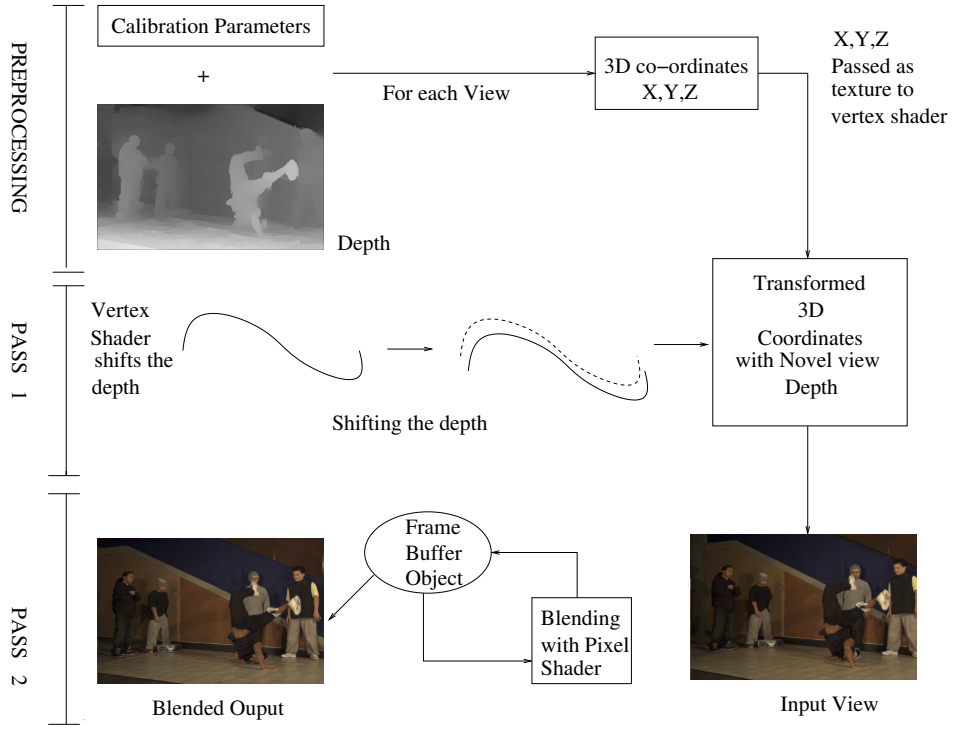

Figure 2.8: Block diagram of the GPU-based rendering algorithm

The first pass leaves z_m, the closest z value, in the Z-buffer for each pixel. The value is offset by Δz so that all pixels with depth less than $z_m + \Delta z$ will succeed the depth test in the second pass and will be blended. The offsetting in eye space is done using a suitable vertex shader program. Lighting, shading and updating of the color buffers are disabled in the first pass to speedup the computations.

The second pass performs the blending using a pixel shader that runs on the GPU. For each pixel, the shader accesses the novel view and DI parameters and the results of previous rendering using a Frame Buffer Object (FBO). Depending on which DIs had values near the minimum z for each pixel, a different combination of DIs can be blended at each pixel. The color values and alpha values are kept always correct. Hence, there is no post-processing step that depends on the number of DIs blended. The algorithm also ensures that there will be no exceeding of the maximum range of color values that is possible if the summing is done in the loop followed by a division at the end.

The GPU algorithm used Vertex Buffer Objects and vertex arrays to store the DIs as triangulated models. The above algorithm was able to achieve a frame rate of 40 fps for the scene involving 10 DIs on a Nvidia 6600GT graphics card. The depth images had a resolution of 512×512. The frame-rate increased to 90 fps when the resolution was changed to 256×256 by dropping alternate rows and columns of the depth map. The video memory on the GPU was saturating and affecting the performance. The frame rate on a 20 DI scene was 10 and 35 for the higher and lower resolutions respectively. Typically, $4 - 5$ DIs were blended for each new viewpoint.

The image based rendering using depth maps to provide novel views in real time can efficiently be used to render 3D scenes at remote locations. Compression of depth maps of 3D scenes has to be efficient enough to transfer the depth maps to the remote site. Such remote transmission is conceptually termed as 3D tele-immersion or 3D tele-presence.

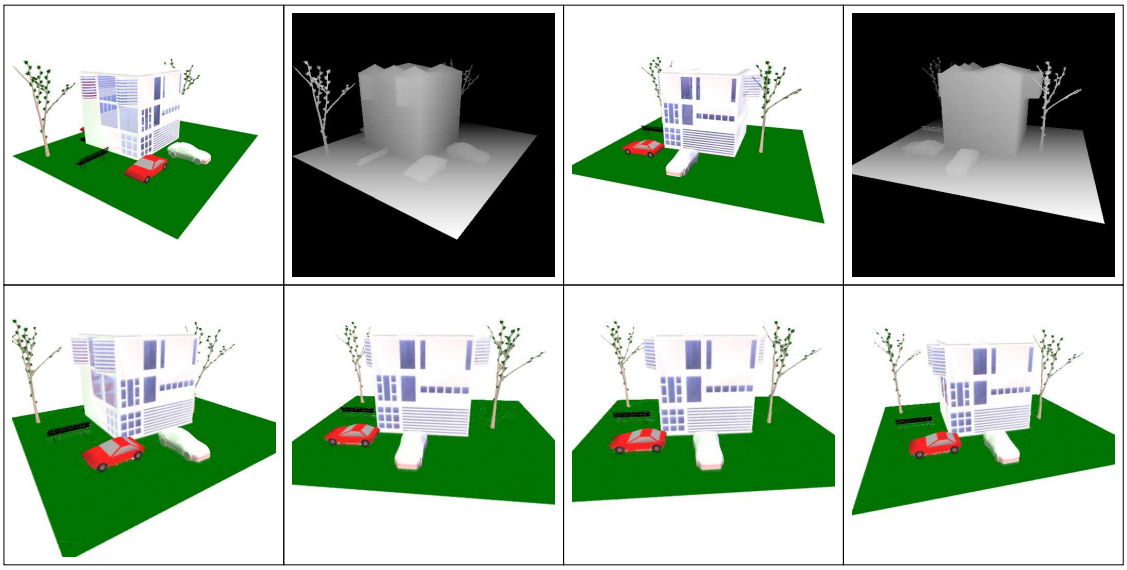

Figure 2.9: Top row: Depth Image pair for a synthetic view. Bottom row: New views generated using them.

2.6 Real-Time 3D Transmission

The topic of 3D tele-immersion incorporates knowledge from multiple disciplines, such as image-based rendering, video coding, optics, stereoscopic displays, multi-projector displays, computer vision, virtual reality, and psychology. Some of the work may not be widely known across disciplines. There are some good overview books on 3DTV [66, 37]. Geometric structure of real-life scenes can be captured using multicamera setups, range scanners, etc. Several systems have been built for this purpose over the past decade [64, 95, 56, 15, 98, 31, 102]. They attempt to capture dense or sparse 3D structure of the scene using cameras as time-varying depth and texture maps or depth movies.

2.6.1 Model-Based Systems

Typical scene models are per-pixel depth maps [24, 102], the visual hull [55], or a prior model of the acquired objects, such as human body shapes [15]. It has been shown that even coarse scene models improve the image quality during view synthesis [28]. It is possible to achieve very high image quality with a two-layer image representation that includes automatically extracted boundary mattes near depth discontinuities [102]. One of the earliest and largest 3D video studios is the virtualized reality system by Kanade et. al [39] with 51 cameras arranged in a geodesic dome, which was later enhanced to a much larger room [38]. The Blue-C system at ETH-Zurich developed by Gross et. al consists of a room-sized environment with real-time capture and spatially-immersive display [31]. Javidi et. al [37] worked on the Argus research project of the Air Force that uses 64 cameras arranged in a large semi-circle. Many other, similar systems have been constructed. All 3D video systems provide the ability to interactively control the viewpoint, a feature that has been termed free-viewpoint video by the MPEG Ad-Hoc Group on 3D Audio and Video (3DAV) [82]. During rendering, the multiview video can be projected onto the model to generate more realistic view-dependent surface appearance [55, 15]. Some systems also display low-resolution stereo-pair of views of the scene in real-time. Real-time acquisition of scene models for general real-world

scenes is very difficult and is a subject of ongoing research. Many systems do not provide real-time end-to-end performance, and if they do they are limited to simple scenes with only a handful of objects.

Theobalt et. al [87] described a system developed at MPI to capture human motion at interactive frame rates without the use of markers or scene-intruding devices. A person is recorded by multiple synchronized cameras as shown in Figure 2.10, and a multilayer hierarchical kinematic skeleton is fitted to each frame in a two-stage process.

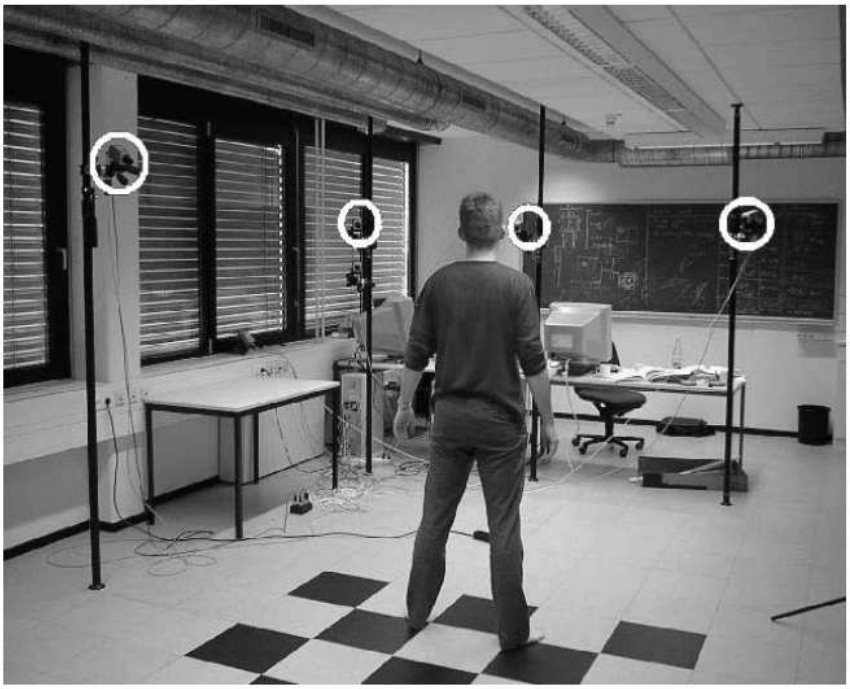

Figure 2.10: Camera Studio at MPI Germany, used by Theobalt et. al [87] with calibration pattern on the floor, 4 cameras marked as circle.

A dense lightfield representation was also used that does not require a scene model, although it was able to benefit from it [28, 14]. On the other hand, dense lightfields require more storage and transmission bandwidth.

2.6.2 Lightfield Systems

Levoy et. al [48] termed a lightfield as representing radiance as a function of position and direction in regions of space free of occlusions. The ultimate goal, which was called the hyper display [60], is to capture a time-varying lightfield passing through a surface and emitting the same (directional) lightfield through another surface with minimal delay. Early work in image-based graphics and 3D displays has dealt with static lightfields [48, 28]. Acquisition of dense, dynamic lightfields has only recently become feasible. Some systems use a bundle of optical fibers in front of a high-definition camera to capture multiple views simultaneously [37] . The problem with single-camera systems is that the limited resolution of the camera greatly reduces the number and resolution of the acquired views.

Now-a-days, most systems use a dense array of synchronized cameras to acquire high-resolution

lightfields. Typically, the cameras are connected to a cluster of PCs [73, 62, 97]. Wilburn et. al [4] devised the Stanford multi-camera array, which consists of up to 128 cameras and special purpose hardware to compress and store all the video data in real-time. Most lightfield cameras allow interactive navigation and manipulation (such as freeze frame effects) of the dynamic scene. Some systems also acquire [62] or compute [73] per-pixel depth maps to improve the results of lightfield rendering.

2.6.3 Multiview Video Compression and Transmission

Multiview video compression has mostly focused on static lightfields [52, 70]. There has been relatively little research on how to compress and transmit multiview video of dynamic scenes in real-time. A notable exception is the work by Yang et al. [97]. They achieve real-time display from an 8×8 lightfield camera by transmitting only the rays that are necessary for view interpolation. However, it is impossible to anticipate all the viewpoints in a TV broadcast setting. They transmit all acquired video streams and use a similar strategy on the receiver side to route the videos to the appropriate projectors for display.

Most systems compress the multiview video offline and focus on providing interactive decoding and display. An overview of some early online compression approaches can be found in [37]. Motion compensation in the time domain provides temporal encoding, and disparity prediction between cameras gives spatial encoding as defined by Tanimoto et. al [84]. The Blue-C system converts the multiview video into 3D video fragments that are then compressed and transmitted [45]. However, most current systems use a centralized processor for compression, which limits their scalability in the number of compressed views.

Another approach to multiview video compression, promoted by Fehn et. al in the European ATTEST project [24], is to reduce the data to a single view with per-pixel depth map. This data can be compressed in real-time and broadcast as an MPEG-2 enhancement layer. On the receiver side, stereo or multiview images are generated using image-based rendering. The core for ATTEST is a flexible and scalable syntax for image-based 3D data representation, which opens for different display types and viewing conditions, as shown in Figure 2.11.

However, as seen in Chen et. al [19], it may be difficult to generate high-quality output because of occlusions or high disparity in the scene. Moreover, a single view cannot capture view-dependent appearance effects, such as reflections and specular highlights. High-quality 3D TV broadcasting requires that all the views are transmitted to multiple users simultaneously. Smolic et. al, the MPEG 3DAV group [82] have been investigating compression approaches based on simultaneous temporal and spatial encoding.

2.7 Compression of Depth Images

Since 3D data is massive in size, it needs efficient compression for representation and transmission. The standard image compression methods like JPEG give a maximum perceived visual quality. These algorithms are psycho-visually motivated and hence may not be the best for depth images, especially for the depth-maps which carry the geometric information.

Several methods have been reported for this. Levoy et al. [48] described the lightfield compression technique using vector quantization. Later, they [51] compressed lightfield using disparity compensation techniques. Girod et al. [17] and Tong et al. [88] have described disparity compensation techniques for compressing multiple images. Ihm et al. [35] and Girod et al. [17] used wavelet tranforms for compression. Wilson et al. [93] proposed an incremental representation exploiting

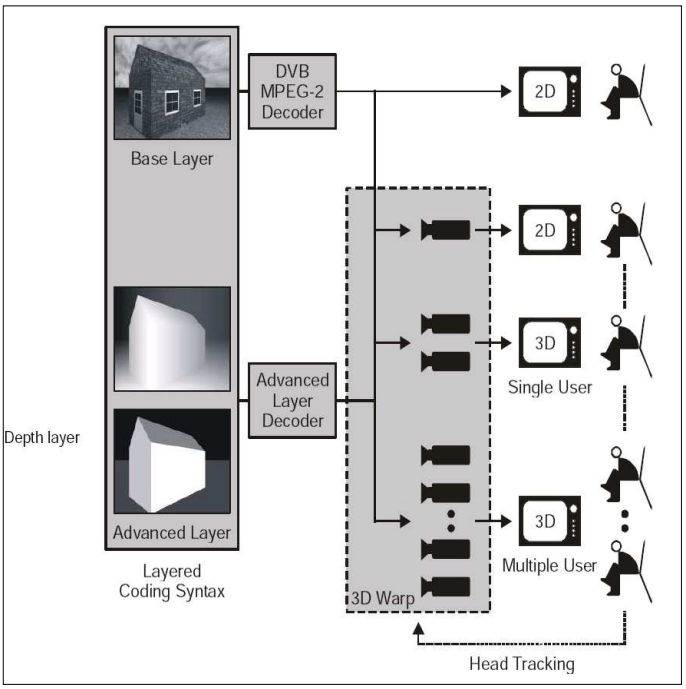

Figure 2.11: The layered coding syntax provides backward compatibility to conventional $2D$ digital TV and allows to adapt the view synthesis to a wide range of different $2D$ and $3D$ displays.

spatial coherence. Ahuja et al. [36] proposed a compression algorithm based on the use of Wyner-Ziv codes, which satisfies the key constraints for IBR streaming, namely those of random access for interactivity and pre-compression. These techniques are used to compress the images alone without using any geometry.

Magnor et al. [50] showed the enhancement in prediction accuracy using geometry such as depth maps and 3D models. Figure 2.12 shows the prediction of images using the geometry. Gotz et al. [29] proposed spatially encoded video, which uses spatial coherence to encode sample images using model-based depth information. All these techniques look for compression of lightfield or multiview images.

Geometry proxy is an approximated geometric model. Performance of rendering of novel views can be increased by using geometry proxies [14, 28, 101, 55, 79]. Geometry proxies are also used to increase appearance prediction by depth correction. All these techniques used geometry proxies for increasing the quality of rendering views. Here, we use geometry proxy for compressing multiple depth maps.

Krishnamurthy et al. [42] used Region of Interest (ROI) coding and reshaping of dynamic range where the accuracy of depth is crucial for compressing depth maps. They showed that JPEG compression on depth maps causes loss of depth information. The idea of having a compact representation of 3D objects with depth images instead of polygon meshes to represent a scene was introduced by Levkovich et. al [47]. They generated depth and treated it as a gray map. They gave texture compression methods like simple textures, point textures and octree images.

Magnor et. al [50] used block-based disparity compensation for encoding multi-view image data, mainly emphasizing on reducing the texture efficiently. They dealt with texture compressions and had less to do with depth compression. Towles et. al [89] developed a system to transport and

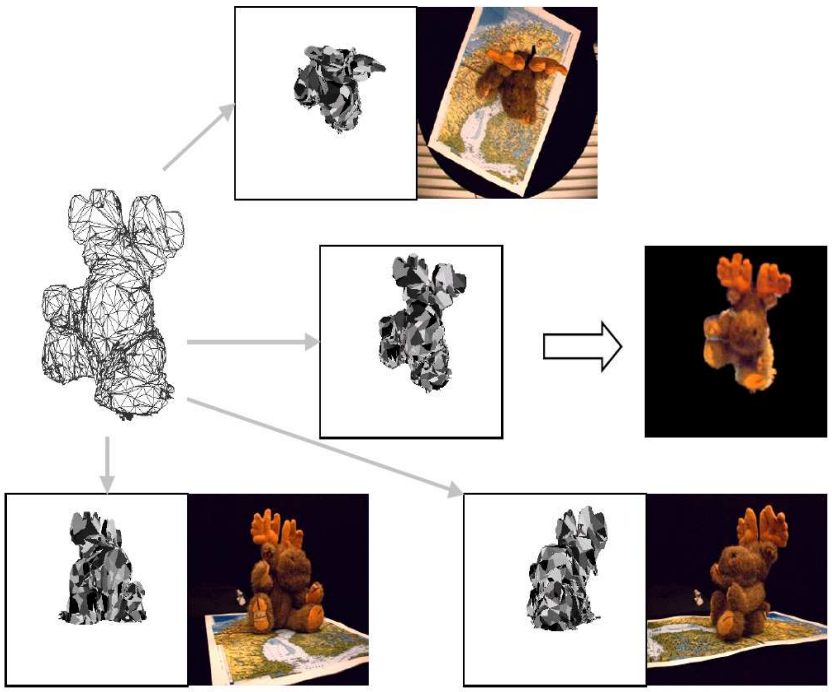

Figure 2.12: Geometry Proxy introduced by *Girod et al.*

render 3D Tele-Immersion data. The temporal and spatial coherence between the depth streams was exploited to compress the depth maps efficiently.

For dynamic scenes, texture has been compressed by finding video objects (VOs) and video object planes (VOPs) by Wu et. al [94] but depth maps were considered as simple gray maps as shown in Figure 2.13. Depth movie compression can be efficient for 3D scenes, by exploiting the temporal and spatial coherence of depth streams from various cameras. Kum et. al [43] attempted to compress multiple depth streams of a scene, where they encode color and depth streams using separate motion vectors. They concluded that for encoding a depth stream with high quality, using separate motion vectors to encode color and depth performs better than using a single motion vector.

Penta et. al [67] used parametric geometric proxies for defining 3D representation of depth maps. They defined geometry proxy as an approximate description of the scene that is used to model the common, position-independent, scene structure. The geometry proxy P can be a parametric model or an approximate triangulated model. The proxy is assumed to represent the geometric structure of the scene adequately. The depth value of each grid point is replaced by the difference of the input depth map from the distance along the same direction to the proxy. The difference at each grid point between the predicted and the actual depth values is stored as residues. The residues are small in range everywhere if the proxy is a good approximation of the scene geometry.

The geometry proxy could be a parametric object like a bounding box, a best-fit ellipsoid, or an approximate polygon-based model of the scene. Such proxies can be created from the input depth maps themselves. Figure 2.15 shows how the residue images R_i are computed by projecting the proxy to each depth image. These residues were bit-wise encoded to get the compressed forms. To

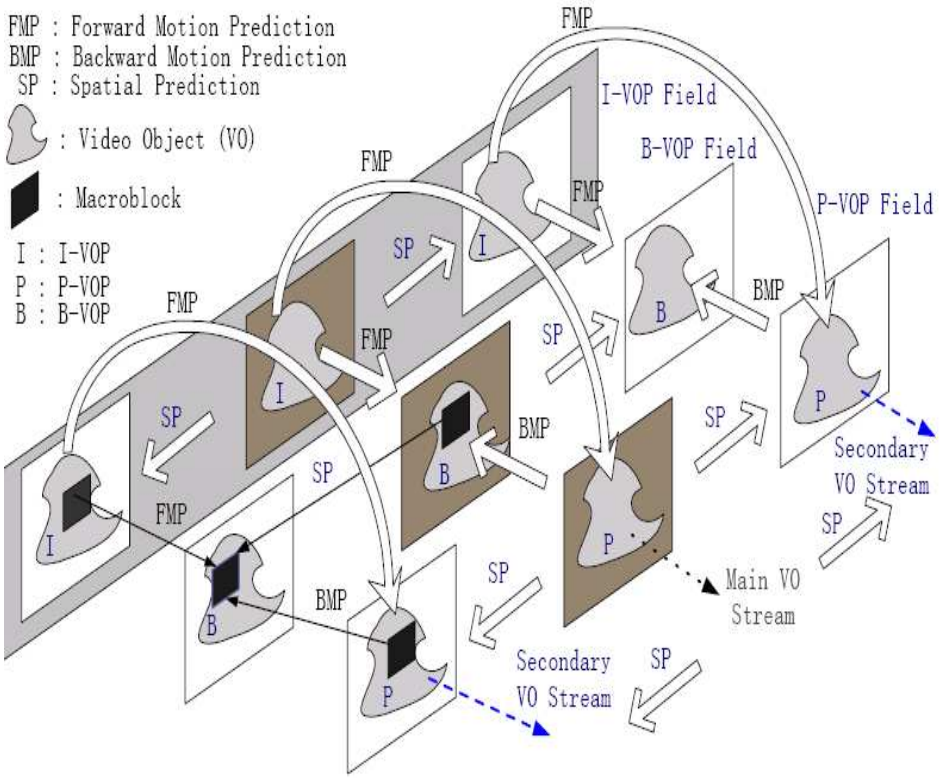

Figure 2.13: Texture coding of an IBR object in the plenoptic video as shown by Wu et al.[94].

get the accurate representation back from the residues, penta et. al added an encoded bit to the 3D model and each time the generated 3D model got more closer in details with the input model.

2.8 Human Body Representation in Scene

Dynamic events involving humans is of special interest to telepresence. Therefore, the 3D representation of the human body merit special attention in different scene representation technologies for 3DTV.

Modelling the skeleton and body appearance Several articulated 3D representations and procedural formulations have been proposed to model the structure and movement of the human body. A human body model can be represented as a chain of rigid bodies, called *links*, interconnected to one another by *joints*. Links are generally represented by sticks [3], polyhedrons [96], generalized cylinders [34] or superquadrics [26]. A joint connects two links by means of rotational motions around their axes. The number of independent rotation parameters defines the degrees of freedom (DOF) associated with a given joint. Development of a highly realistic human body model is a computationally expensive task, involving a problem of high dimensionality. In computer vision, where models need to be only moderately precise, articulated structures with low DOF are generally adequate [26, 21]. But, the stick forms of Aubel et. al [3] are considered to be highly accurate representations consisting of more than 50 DOF and are usually desired. The models proposed for the body appearance can be classified into four categories: stick figure models, surface

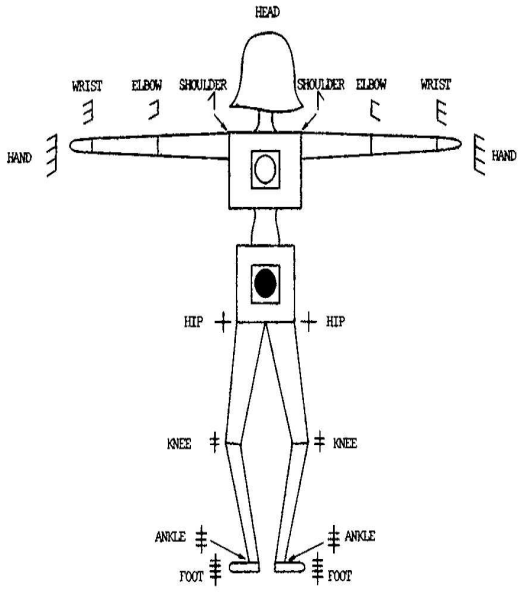

Figure 2.14: Body Parts as shown by *Balder et al.* [7]

models, volume models, and multilayer models. *Stick figure models* as represented by Badler et. al [7], are built using a hierarchical set of rigid segments, connected by joints; they allow for easy control of movement, but realism is limited. They also gave a crude representation for the division of human bodies in order to parametrize the body, as shown in Figure 2.14.

Surface models are based on two layers: a skeleton, which is the backbone of the character animation, and a skin. The skin can use different types of primitives: points and lines, polygons [85], curved surface patches [49, 41], and subdivision surfaces [22]. In *volumetric models*, simple volumetric primitives, such as ellipsoids, spheres and cylinders as shown by Yoshimoto et. al [100] or implicit surfaces [85, 12] are used to construct the shape of the body. They perform better than surface models but it is difficult to control a large number of volumetric primitives during animation. *Multilayer models* consist of three layers: *skeleton, muscle* and *skin*. Complex motions are produced easily by building up the animation in different layers. Chadwick et al. were the first to use a muscle layer [16]. Nedel and Thalmann simulated muscles by a mass-spring system composed of angular springs [65].

Motion of the Skeleton: There are a number of ways to procedurally model an articulated human body using the kinematics and dynamics approaches. A mathematical model that describes the parameters of the links and the constraints associated with each joint is called a *kinematics model* and it can only describe the possible static states of a system [16, 6, 27]. In a dynamic model, the state vector includes positions, linear and angular velocities, accelerations, and the underlying forces and torques that act on this model [27, 5]. Dynamic model-based approaches are used to realistically animate walking models. However, dynamic model-based techniques are computationally more expensive than kinematics-based techniques. Determining the motion parameters explicitly at each frame, even for a simple motion, is non-trivial. Hanrahan et. al [33] gave the solution of specifying a series of key-frame poses and interpolate the joint parameters between those key-frames. Linear interpolation is the simplest method of generating the intermediate poses, but it produces a robotic motion due to discontinuous first derivatives in the interpolated joint angles.

Obtaining smooth velocity and acceleration requires higher order interpolation methods, such as piecewise splines by Steketee et. al [83].

Since dynamics simulation cannot solve all animation problems, motion capture techniques have been introduced to animate virtual characters from real human motion data. Motion capture methods are mainly used in the film and computer-game industries. The motion of a real actor is captured by tracking the 3D positions and orientations of points located on him, using mechanical, electro magnetic or optical technologies [80, 59]. This method produces realistic and highly detailed motion in a short time. Gavrilla et. al [25] have also investigated marker-free optical methods, as many application scenarios require no visual intrusion into the scene. Human body model can be described in various ways but for human body models to be interchangeable, a standard for animation is required. The Web 3D H-anim [1] standards for human representation and the MPEG-4 representations for facial and body animation have been developed to meet this need [69].

Body Modelling: There are various methods to acquire 3D human body models. Some commercial systems require special hardware, but they are expensive and can not be used in certain cases. Using video frames rather than using special hardware is preferable. A number of techniques using video frames have been proposed [30, 11, 3, 96, 3, 26, 21, 7, 85]. Gavrila et. al [26] simplified the acquisition of the shape parameters with known poses. Figure 2.16 shows models used by them.

First of all, different views of the subject are obtained from different calibrated cameras or one moving camera. From each view, the 2D silhouette of the subject is extracted. Then using volume intersection, different views of the subject are intersected and a volumetric description of the subject is defined. Finally, a model of the human body is fitted to the volumetric description of the subject. Sticks, ellipses, cylinders, super-quadrics can be used for the predefined model. In some methods [34, 11, 26] the subject needs to perform some initial movements in order to obtain the model more accurately.

Body Motion Tracking: The initial position and posture of the person are assumed to be known. Prior to human body motion estimation, the segmentation of human silhouette from the background should be made. Then the feature extraction and tracking follows. The prediction of movement is also used to solve the occlusion problem. Some tracking techniques try to determine the precise movements of each body part as mentioned by Thalmann et. al [49], while other methods focus on tracking the human body completely [41, 100]. Tracking techniques may also be classified as 2D and 3D. Using a 2D approach, the motion in the image plane is analyzed either by exploration of low-level image features or by using a 2D human body model. 3D tracking tries to obtain the parameters that describe body motion in three dimensions. 3D tracking allows 3D pose recovery, position estimation of the body parts in 3D space and orientation estimation of the body relative to the camera. The 3D pose parameters are commonly estimated by iteratively matching a set of image features extracted from the current frame with the projection of the model on the image plane. The overview of existing human motion analysis techniques can be found in [12, 16, 65, 6].

Body Motion Recognition: Human motion recognition may be achieved by analyzing the extracted 3D pose parameters. Girad et. al [27], instead of obtaining the exact position of a human body, defined human motion recognition to identify the action performed by a moving person. Most of the known techniques focus on identifying actions belonging to the same category (e.g. specific sport movements, sitting down, standing up, walking, running, etc.) [27, 5]. Some of the techniques recognize and identify several persons and their interactions [33, 83, 80]. Some of them are developed to work in special environments and try to use prior knowledge about the layout of the room, as defined by Menache et. al [59].

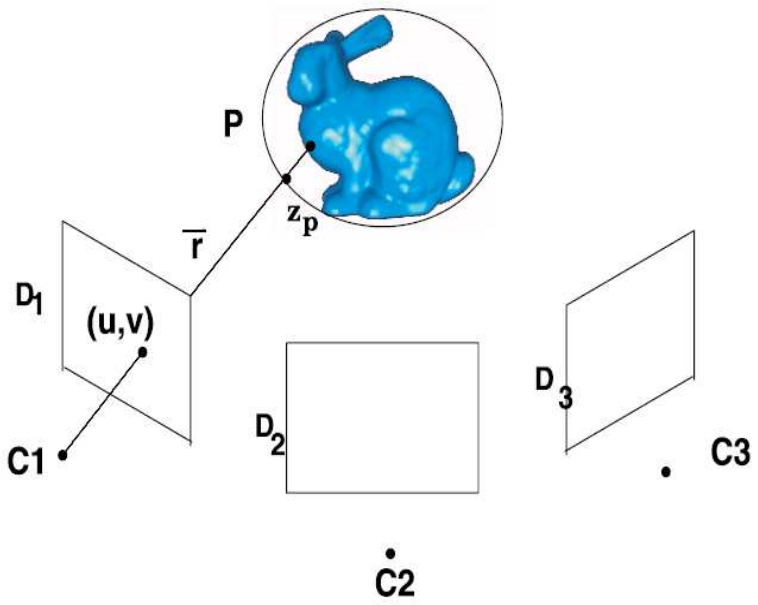

Figure 2.15: The geometry proxy P (an ellipsoid in this case) represents common structure. It is projected to every depth image. The difference in depth between the proxy and the scene is encoded as the residue at each grid point of each depth map.[67].

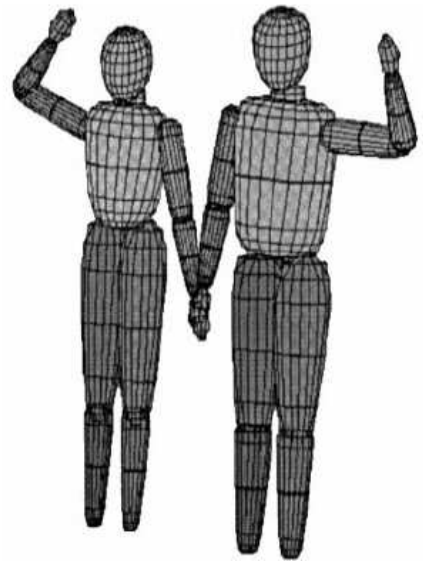

Figure 2.16: 3-D human models "ELLEN" and "DARIU" using tapered superquadrics, [26]

2.9 Summary

In this chapter, we reviewed earlier work done in representation of 3D scene, depth images, compressing depth images and its applications to 3D Tele-immersion. We saw that major works that has been done in past for 3D dynamic scene representation does not consider the difference between texture compression and depth compression. The D channel of dynamic scenes has been left uncompressed or inefficiently compressed. This inefficiency incorporates for non-usability of an important 3D scene representation, *depth movies*, for 3D tele-immersion. In the following chapters, we present our work on proxy based compression and 3D scene representation using Depth Movies. We will present our method of parametric proxy based compression of Depth Movies where we have concentrated on depth movie compression exploiting both the temporal and spatial coherence of depth movies. As human motion is regarded as the most complex of motions with a very high degree of freedom, we have considered the example of scenes with complex human motions. Our method involves parameterizing the human motions in a dynamic scene with the concepts highlighted by Section 2.8. We have represented these parameterized scenes with depth movies and compressed them using proxy-based compression schemes in the following chapters. Next chapter presents proxy-based compression of depth images in detail and also explains the preliminary experiments to compress depth movies. Later we present the parameterized proxy-based compression of depth movies.

Chapter 3

Depth Map Compression

Depth images are viable representations that can be computed from the real world using cameras or other scanning devices. The depth map provides 2-$\frac{1}{2}$D structure of the scene. A set of depth images can provide hole-free rendering of the scene. Multiple views of the scenes need to be blended to provide smooth hole-free rendering, however. Such a representation of the scene is bulky and needs good algorithms for real-time rendering and efficient representation. In this chapter, we present a discussion on the depth image representation and provide a proxy-based compression scheme for depth images.

3.1 Representation, Rendering and Compression of Depth Images

1. Representation: As depth images do not carry photometric information, they need to be handled differently from the textured images. The depth images contain real numbers whose range depends on the resolution of the rendering algorithm. As images with 16 bits per pixel can represent depths upto 65 meters with a 1 millimeter resolution, bit reduction can be applied as preprocessing to the depth maps before rendering.

2. Rendering: A depth map can be considered as a cluster of 3D points. These 3D points from each depth image are rendered using splatting or triangulation. Since, a single depth map lacks the full information about the scene structure, holes or gaps corresponding to the part of the occluded scene being seen from the novel view. Multiple depth maps are rendered in the vicinity of the view to fill these holes. The parts of the occluded seen are visible to the multiple cameras nearby and blending such views gets the new scene without holes.

3. Compression: The image representation of the depth map may not lend itself nicely to standard image compression techniques like JPEG, which are psychovisually motivated. Since, depth maps contain common information between the near-by views, spatial coherency can be exploited to compress well after removing the redundant information. Thus, epipolar constraint, disparity, multi-linear tensors, etc. can be exploited for compression of Depth Images.

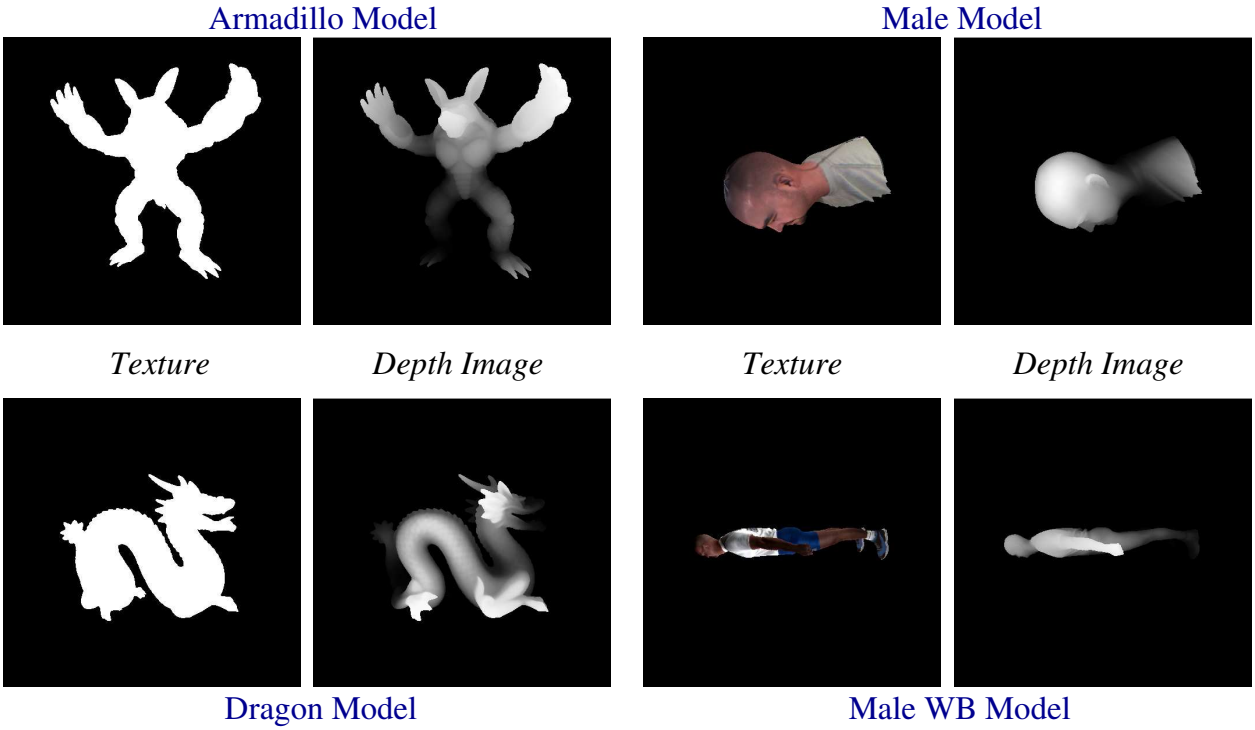

Figure 3.1: An overview of a few models - texture and corresponding depth images.

3.1.1 Method: Description of the Models

The 3D models used for the experimentation are standard models like *Armadillo, Happy Buddha, Buddha, Dragon*. We also used a few CyberWare models like *Male, Female, Ganesh and IndianGodess*. The 3D models used in the experimentation were obtained from Georgia Institute of Technology's Large Geometric Models Archive, Tupperware, Cyberware and Konstanz 3D Model Repository. Figure 3.1 shows a few of the models used and Table 3.1 gives the description of the various models used. We also experimented with Stanford Bunny, horse, some small models like hand. A brief overview of the method described below is shown diagrammatically in Figure 5.3.

3.1.2 Generating the Proxy Models

For each 3D model we generate the corresponding proxies. We consider, a proxy model as the approximation of the original model with reduced number of triangles and the size of the file. We reduced the Level of Detail(LOD) of the models using the GLOD, which is a lightweight tool to generate geometric level of detail. GLOD performs simplification on objects as a collection of patches. For each model, we generated five different LODs of Proxy models, ranging from few hundreds to half the original triangles. The main aim is to get a proxy model with size with out much of the detail of the model. The only parameter used while reducing the level of detail is the number of the triangles. For example, one of our model initially had 8,71,414 triangles. We generated the proxies with 400K, 200K, 50K, 10K and 1K triangles.

Table 3.1: **Model Descriptions**

Model Name	Repository Name	Number of Triangles
Armadillo	Georgia Tech	345944
Shakyamuni	Konstanz	499996
Buddha	Georgia Tech	1087716
Dragon	Georgia Tech	871414
Ganesh	Cyberware	413236
IndianGoddess	Cyberware	274822
Male	Cyberware	605902
Male WB	Cyberware	296272
Female	Cyberware	605086
Female WB	Cyberware	243442

3.1.3 Setup

After proxy models are generated for each and every model, these models along with their proxies are subjected to scaling. The cameras are placed in a circle round the Y-axis in a plane parallel to XZ-plane. We placed 20 cameras in 2 planes with different angles of elevations, usually θ, 2*θ, circling the model with ten cameras in each plane. We store the camera positions as the 3 matrices, calibration matrix (K), rotation matrix (R) and translation matrix (T).

3.1.4 Depth Maps and Masking

The crucial part is generation of the depth map for each. The depth maps are represented using 16-bit values in an uncompressed format. This helps in capturing of 65535 different depth values which is sufficient for any real practical situation. For each model, one depth map and a mask corresponding to each view is stored. The mask is a bitmap which stores 1 for the background and 0 for the foreground, as a binary image. This mask is used later while calculation of the residues and encoding or decoding the residues. We also calculated depth maps for all the proxy models.

3.1.5 Residues

Residue is the difference between the depth map and its corresponding depth map of the proxy model from the same view. We can store the difference of all pixels, or only the difference of the pixels that are in the foreground of the original model,i.e we store the difference of only those pixels that are 1 in the mask. The mask is available while reconstructing the depth map back so we are not loosing the position of the pixel in the original image. We store sign bit of these in a separate bitmap, 0 and 1 representing the positive and negative sign of the residue.

In order to have a better compression of the residues, we need the variation of the difference in depths to vary smoothly because it helps in encoding the residues with less number of bits and these can be compressed well with standard methods like ZIP, LZW etc. We calculated the maximum number of bits per pixel in the residue for all the data. A significant number of pixels need around 15 bits to represent the residue when depths are 16-bits long. This is not efficient, as the residue needs a similar number of bits. A typical geometry proxy is as shown in Figure 3.3.

In order to reduce the big numbers at the fringes we assumed that there is a infinite plane present at the depth equal to the average depth of proxy. This helps in shifting these large numbers to smaller numbers and the maximum number of bits for the residues using this method is around 11

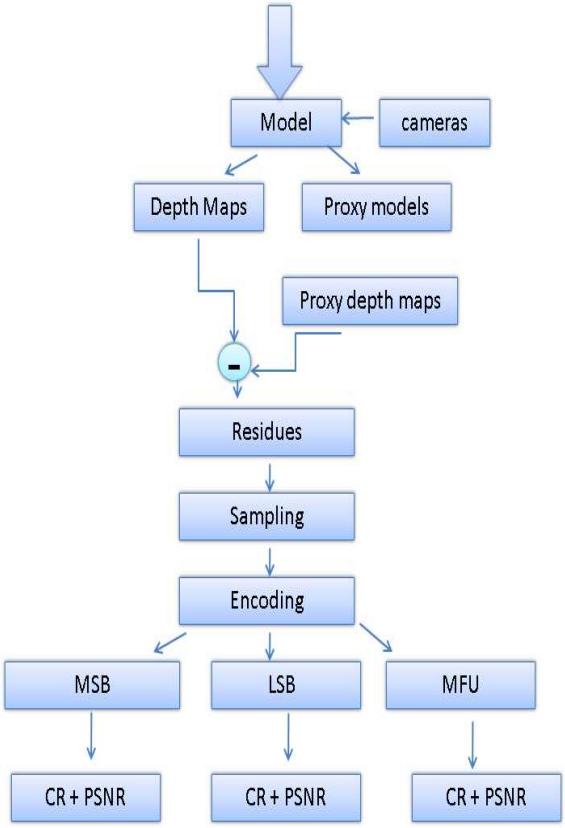

Figure 3.2: A Flowchart overview of the proxy based compression strategy for Depth Images.

bits, only few pixels need more than 8 bits. The following Table 3.2 explains how we handle the foreground and background situations.

We can easily calculate the mean depth of the proxy while reconstructing. The main aim of this average plane at average depth is to reduce the maximum number of bits per pixel for the residue.

3.1.6 Encoding of Residues

We encode the 16-bit residues by using only the information in the foreground with the help of mask and reduced the maximum number of bits required to around 11 bits instead of 16. So, we try to encode these residues with various number of bits and compared the compression rates and mean square error. We encoded the residues by three methods - most significant bit (MSB), least significant bit (LSB), most frequent bit (MFU). We tried the bitrates from 0-8, at 8 bits the error values were less.

- Most Significant Bits - MSB While encoding the depth maps with various bits we consider the most significant bits first. Suppose we have the depth map with maximum bits of 12 then we never store 0-4 least significant bits in any of the bitrate. Suppose the bitrate is 4 then we consider only 12^{th}, 11^{th}, 10^{th} and 9^{th} bits in store. We then calculate the compression rates and the mean square error. MSB mechanism has good compression ratios with less mean-square-error compared to the LSB/MFU which will be discussed below.

- Most Frequently Used - MFU The one problem we observed in the MSB mechanism is that a lot of pixels, nearly 30-40% or even more depending on the view/proxy model, has their

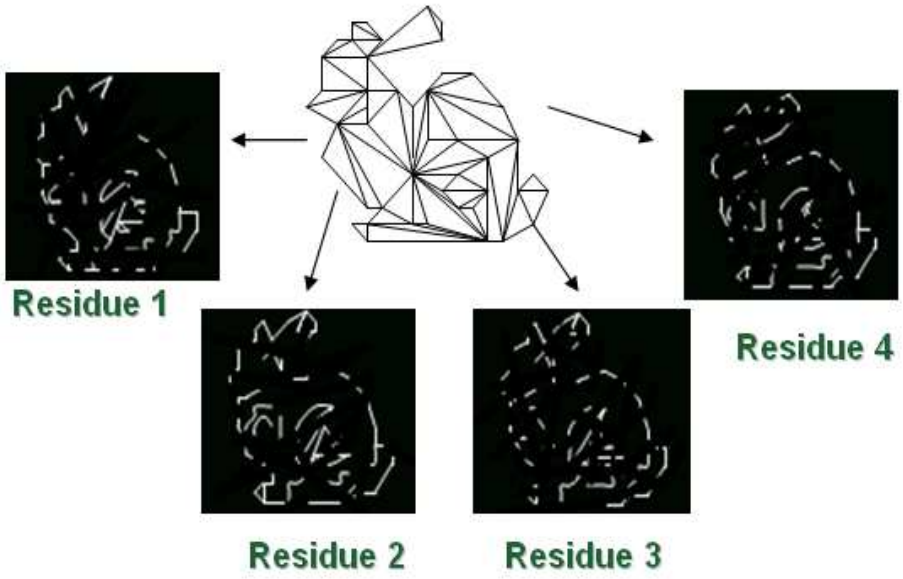

Figure 3.3: A general triangulated geometric proxy and residue computation.

Table 3.2: ForeGround Background

Original	**Proxy Model**	**Value Stored in Residue**
ForeGround	ForeGround	absolute difference of both the depths
ForeGround	BackGround	absolute difference of the original depth and the average depth of the proxy model
BackGround	ForeGround / BackGround	Nothing is stored. We store the difference only when it is the foreground of the original image

residue values in the first 3-4 least significant bits or 4-8 bit positions from LSB side. Hence, we try to send the bit positions with more number of pixel information. After analyzing the bit planes of the pixels we store the more frequently occurring plane of bits first and so on. This has compression rates comparable to MSB method but the mean-square-error is high. We need a good error measure to compare MSB and MFU because mean-square-error can vary a lot if we have one pixel with large pixel value as a error compared to that of the small value.

- Least Significant Bits - LSB we also tried to encode the residue using the LSB but the results are not encouraging. Both compression ratios and mean-square-error are not good. so we discard this mechanism. Among the above three mechanisms MSB and MFU out performs LSB method. MSB is good compared to that of MFU. It has high compression rates and low MSE compared to other mechanisms.

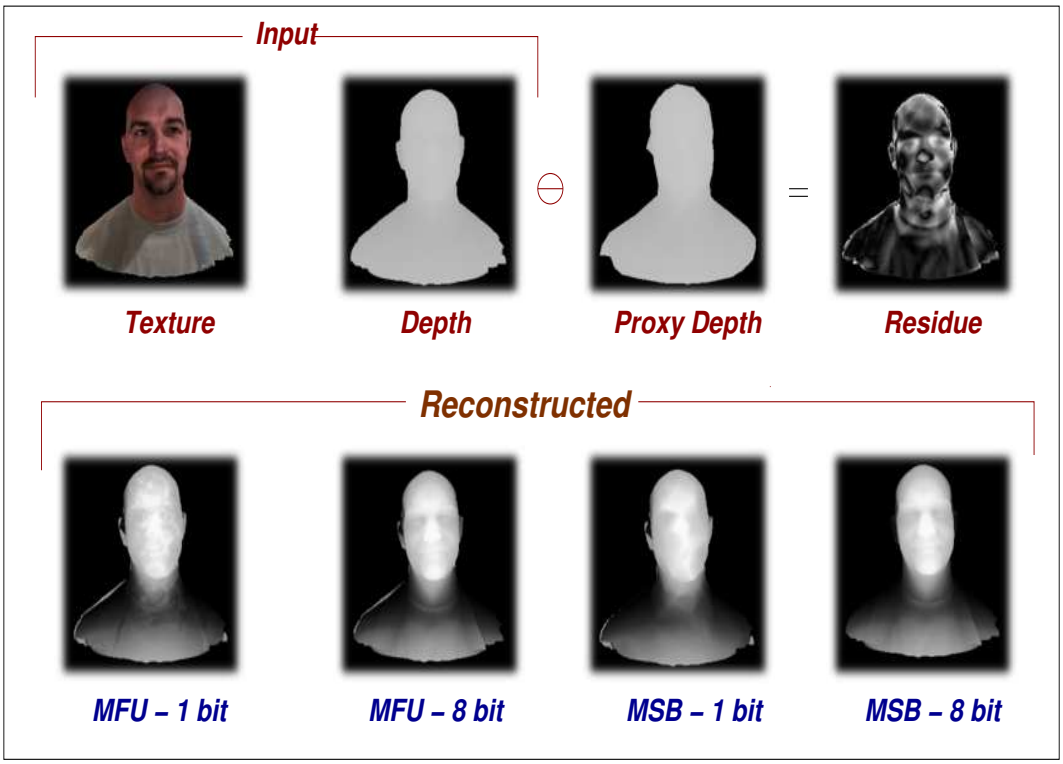

Figure 3.4: Figure shows original texture and depth map of male model, the depth map of a proxy model, residue depth map and the reconstructed depth maps at 1 and 8 bits with MFU and MSB methods used for coding

.

3.1.7 Decoding and Reconstruction of Depth Maps

For reconstruction of the depth maps, first we need to decode the residues. For decoding we need the mask,average depth,sign Bit. Depending on whether it is background or foreground, we add/subtract the residue values to the average depth or normal depth based on the mask. Figure 3.4 shows such reconstructed depth maps.

3.2 Results

The Table 3.3 shows the results for the 3 datasets with varying proxies. Among the datasets that we used, Armadillo is the most dense dataset with 3,390,515 points and about 7,500,000 triangles. The model was created using 114 scans and then VRIP was used to arrange all the scans together.

The analysis of the experiments was done as to how the trend between MSE, compression ratio (CR), number of bits, proxy levels, sampling rate (SR) and other parameters varies. We observed that CR decreases with the increase in the number of bits taken for the residues. This is justified because, as the number of bits for storage increase the compression ratio also reduces. On the other hand, CR increases with the sampling rate (SR) as less number of triangles are used to store the same model. If we lower the number of triangles in the proxy of the model, more sudden is the change in CR (steeper is the curve) when the number of bits to represent the residue decreases.

For different proxies of a model, MSE values increase if the number of bits to represent the residues decreases. As the number of triangles to represent the proxy reduces, MSE increases

Table 3.3: Compression ratios for Armadillo, Buddha and Dragon datasets with varying proxies and 8 bit compression

Compression Ratios and MSE for 3 Models						
Dataset	**triangles**	**CR/SR=1**	**CR/SR=2**	**CR/SR=3**	**MSE/SR=1**	**MSE/SR=2**
	100K	9	28	52	2	6.0e+06
	50K	8	25	48	3	5.5e+06
Armadillo	1K	4	12	23	13	6.0e+06
	100K	8	27	49	0.9	5.0e+06
	50K	7	24	44	1.7	5.0e+06
Buddha	1K	4	13	25	7	5.1e+06
	100K	9	29	51	0.5	5.0e+06
	50K	9	30	53	0.7	4.5e+06
Dragon	1K	4	15	28	5	4.0e+06

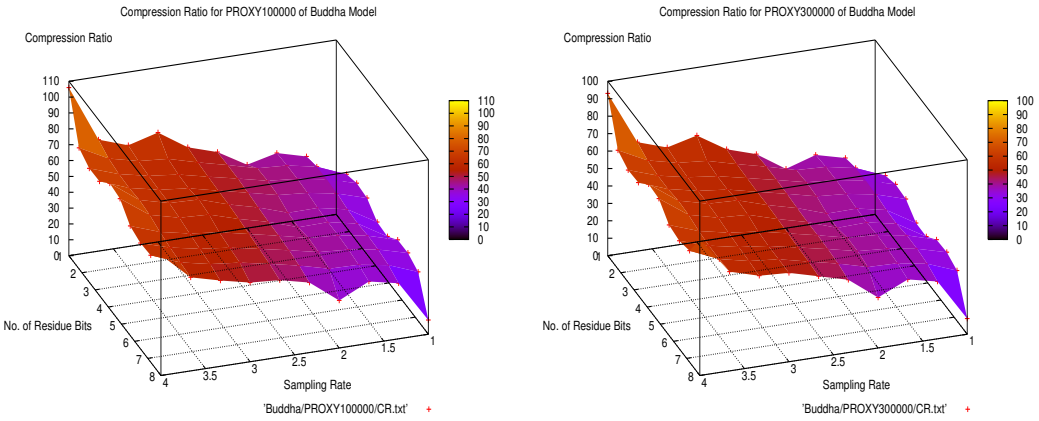

Figure 3.5: Graphs for Buddha model showing trends of compression ratios with SR, bits, camera views for two different proxies of 100000 and 300000 triangles.

since the reconstructed model drifts more and more from the original model without compression and a representation with all the triangles. MFU gives higher MSE in comparison to MSB. MSE increases drastically if sampling rate (SR) increases, also with SR=2,3,4, since the resolution of the reconstructed model reduces. For higher sampling rates (values not equal to 1), MSE remains almost same for all the proxies, as the values are in 10^7. CR values decrease with the increment in the JPEG Quality factor, since with the JPEG quality factor the size of the textured images also increases. By compressing textures(using JPEG directly), we find that there is a gradual change in CR with the proxy change, increment in sampling rate (SR) and decrement in number of bits to represent the residues. In comparison to without texture compression, CR values decreases in the compression with textures.

The graphs in the Figures 3.5, 3.6 show the variation of varies trends in different models.

3.3 Summary

In this chapter, we have presented a proxy based depth image compression method using different 3D real models. This chapter has given us deep insight in a general proxy based compression of depth images from multiple views. The triangulated proxy used here is a specific example of a general common proxy.

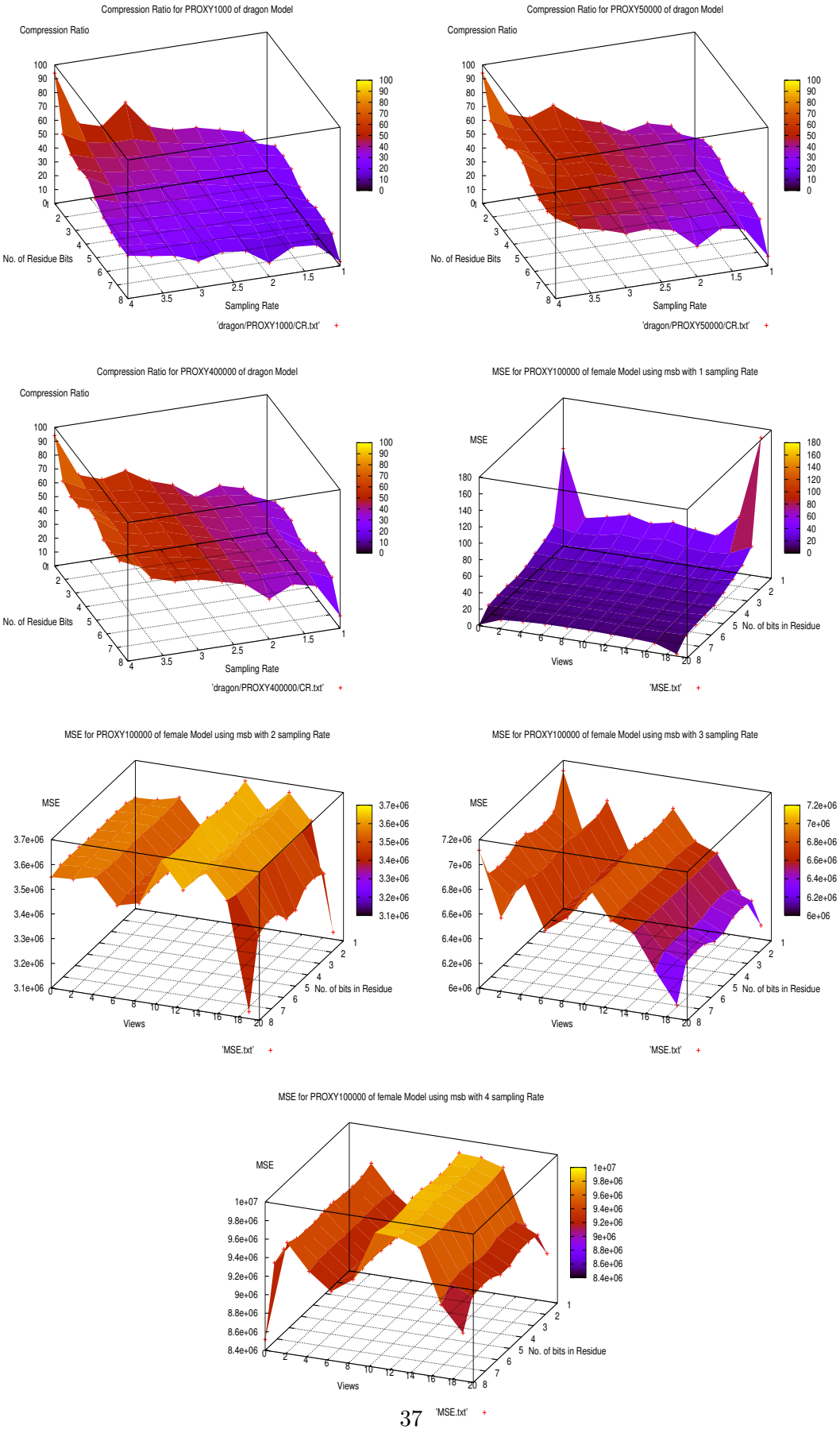

Figure 3.6: Graphs for dragon and female models showing trends of compression ratios and MSE with SR, bits, camera views and proxies

Chapter 4

Depth Movie Compression: Preliminary Work

In this chapter, we explore some elementary ideas on compressing multiple depth movies. We study the use of standard image and video compression methods on depth movies primarily.

4.1 Depth Movie

A depth movie, as described in the previous chapters, consists of 3 channels as described below in brief.

- I Channel The **I** channel contains a conventional video and can be represented as one. Traditional motion-compensated representations such as MPEG can work well with this channel. However, the images of I channel are meant to be used for generating new views using the depth maps.

  The standard video-compression algorithms use perceptually motivated compression schemes based on DCT, DWT or its variants. They result in a softening of sharp variations in colour and intensity. Colour would seem to bleed across a boundary between two sharply different colours. The boundaries in colour are also likely to be boundaries in depth. A boundary in the depth map is very likely to also correspond to a boundary in the image. Bleeding of colours across a depth boundary is distracting and will be noticed quickly as the viewpoint varies. Thus, the compression for the I channel should be performed with this in mind.

- D Channel

  The **D** channel can be thought of as a video of depth values. It may appear that video compression schemes (like MPEG) can be applied to them. There are very critical differences that make this tricky and undesirable.

  1. The depth values are typically floating point numbers, unlike images which are unsigned integers or bytes. They need to be normalized. This can cause artifacts such as the loss of precision.
  2. Colour constancy of a region is an implicit assumption in motion compensation. With depth maps, the depth values represent z-coordinates and not colours. The values undergo a transformation as per the 3D motion of the objects. The region is moved and the values are affected in a coordinated manner.

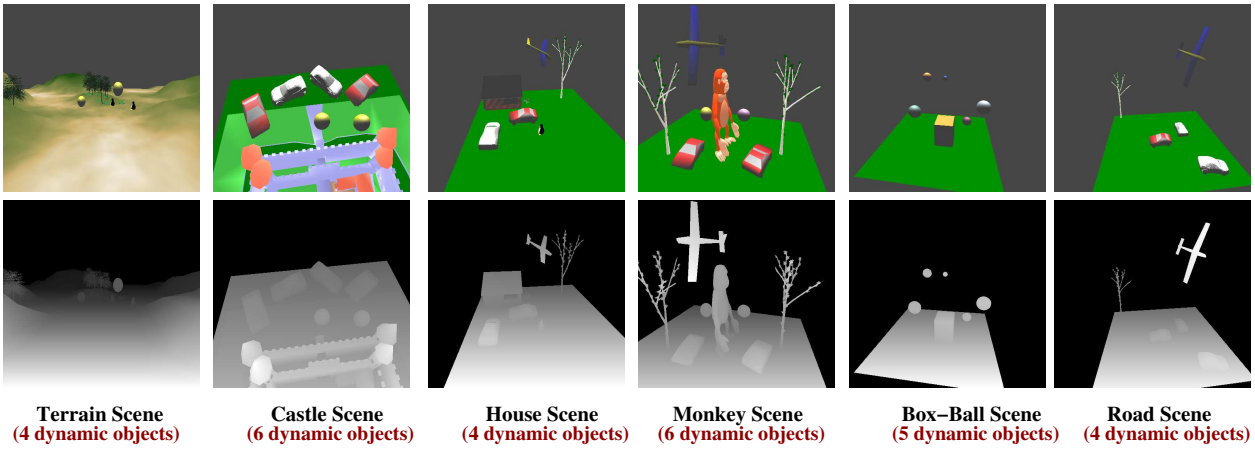

Figure 4.1: Images and Depth Images of all the scenes used for Experiments

- C Channel

  The **C** channel gives the calibration parameters across time. They can be specified in terms of a 3×4 matrix of real numbers or as explicit representation in terms of the position, orientation, focal length, etc. In either case, only a few numbers are needed to specify one set of calibration parameters. They are likely to change infrequently or not at all since the camera is not likely to make rapid motions. Thus, no special representation or compression is necessary for the **C** channel.

4.2 Compression of the D Channel

The D channel encodes time-varying depth values and is similar to a video channel. The depth values can in practice be converted into integers by selecting an appropriate scale for depth representation. Using 16-bit unsigned integer values, 64K different depths can be represented. This provides sufficient precision as well as range in most practical situations. For example, a space of 65 meters can be described by this scheme with a precision of one millimeter. Thus, standard video compression schemes can be used to compress the D Channel, after converting it to integer values.

Lossless compression schemes are designed to reconstruct the data exactly. They produce lower compression ratios but the quality of the D Channel after compression and decompression is perfect. Lossy compression schemes potentially throw away data but in such a way that the unimportant data is discarded first. This results in the uncompressed image to be different. The impact of the discarded data needs to be analyzed. Since we are interested in using depth maps and texture for image-based rendering, we use the original and compressed depth maps for new view generation using the rendering algorithm reported in [90]. The corresponding PSNR values are then reported.

4.2.1 Full Frame Compression

We first look at the compression strategies that treat each depth frame as a whole, in contrast to the differential schemes which work on the frame-to-frame differences in depth values. The depth values are encoded using 16-bit integer values. The full range of depths may not be used in many situations in some depth maps. The entropy-based compression schemes may be able to exploit

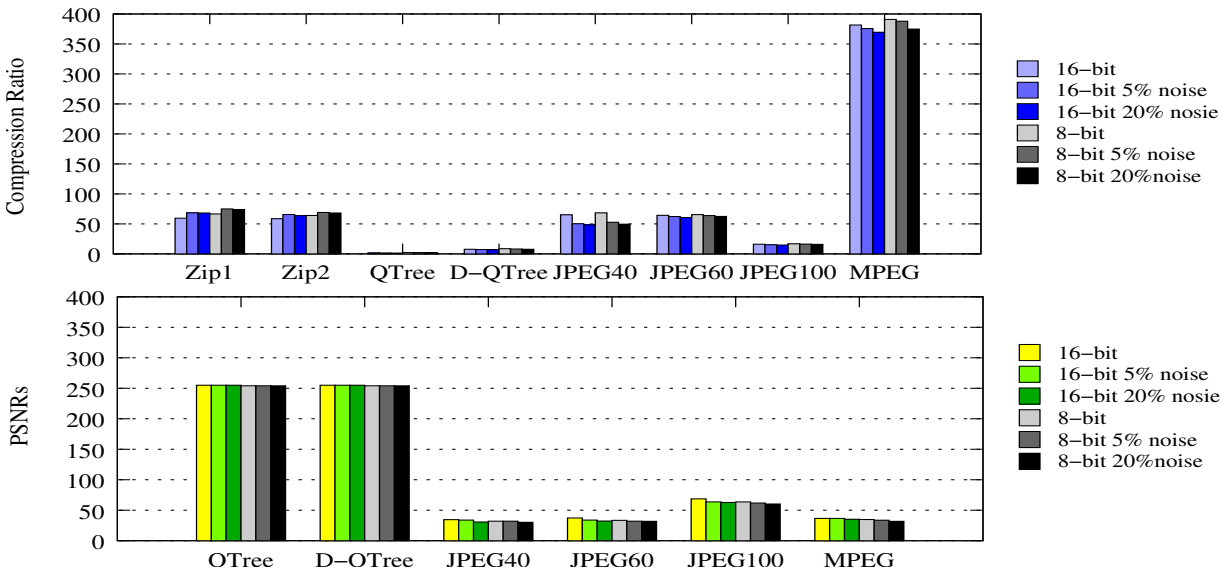

Figure 4.2: CR and PSNR of Monkey scene for 16 and 8 bit depth values with no noise,5% noise and 20% noise

this automatically. We, however, explore the performance of the representations when the range is transformed from 16 bits to 8 bits.

- LZW Compression: Entropy-based lossless compression schemes such as Huffman encoding produce optimal compression, but require the probability distribution of the symbols used. A standard symbol table with associated probabilities can be built for a domain such as depth maps after examining sufficient standard documents. Adequate performance can be obtained by other algorithms such as the LZW which does not need such a table a priori. The encoding algorithm builds a table of substrings as the input is scanned. The table does not need to be stored along with the compressed data and can be built at the decoder, though the table is tuned to the data being compressed. We explore the amount of data redundancy present in the depth image sequence. We use the LZW compression as implemented by the `zip` package for this. The depth values are stored as 16-bit numbers in the raw format into files, one for each time instant. These files are compressed together using the zip package on Linux and the resulting compressed file size is compared with the total storage requirements of the input files to determine the compression ratio. The above process compresses each frame independently and the inter-frame similarities may not be exploited to the fullest. We modify the experiment by storing the entire depth sequence as a single binary file with the frame $(i+1)$ immediately following the frame i.

  The simple compression scheme exploits the redundancy adequately. The compression performance increases with the length of the sequence as the commonality increases.

- Quadtree Representation and Compression: A quad-tree representation of an image can compress it if the image has sufficient spatial redundancy. We compress each frame of the depth sequence by first representing it using a quadtree as in [74] followed by a compression of the tree data using the `gz` algorithm.

- Quadtree Differences: The quadtrees of successive frames is highly correlated. We can then encode the first quadtree in full and the subsequent ones using the difference with the previous

ones. For this, the tree is scanned from the root. The tree nodes that are same are encoded as a single symbol. If a node differs in the later frame, that fact is recorded and the entire subtree is stored. The assembled difference data structure is later compressed using `gz` to get rid of the data redundancy.

- JPEG Encoding: We can encode each depth frame using JPEG, treating it like an image. This can introduce artifacts as JPEG is perceptually motivated and those assumptions may not hold for depth values.
- MPEG Encoding: The depth sequence can be treated like a video and compressed using a standard compression scheme like MPEG.
- Depth Difference Coding: The depth values at corresponding locations of successive frames are likely to be highly correlated and hence change slowly with time. A differential encoding scheme can exploit this. We explore several options for this. The range of depth differences is even lower than that of the individual depth values. We explore the representations when the differences are encoded using 16 and 8 bits per value.
- LZW Compression of Differences: The depth sequence can be encoded using a standard differential coding technique. In this, the pixel-wise difference between successive frames is computed as a difference map, which is compressed using a lossless scheme like `zip` to eliminate data redundancy.
- Quadtree Encoding of Differences: The difference map between frames can be encoded using a quadtree. If the subsequent frames are highly correlated, the difference map will contain small and uniform values.
- JPEG Encoding of Frames: We can encode each depth frame using JPEG, treating it like an image. This can introduce artifacts as JPEG is perceptually motivated and those assumptions may not hold for depth values.
- MPEG Encoding of Movie: The depth sequence can be treated like a video and compressed using a standard compression scheme like MPEG.

4.3 Experimental Results

We conducted our experiments on 6 different dynamic scenes as shown in Figure 4.1. All had either 1,2 or 3 cameras.The number of objects moving in the scenes varied from 4 to 6. Figure 4.3.1 and Figure 4.3.2 show compression ratios for all 6 scenes using 16-bit and 8-bit depth values. Figure 4.3.3 and Figure 4.3.4 show PSNRs for the corresponding cases. When we add noise to the depth maps, the CRs and PSNRs show a varying trend as shown in Figure 4.2 and Figure 4.2 and these trends are explained in section 4.4.

4.4 Discussions and Conclusions

For an application that uses depth maps, it needs to analyze the trends of depth image compression and the factors that effect the compression ratios (CRs) and PSNRs. Looking at the graphs in the Section 4.3, we see that for data without noise, the best compression ratio is achieved by MPEG, which is in agreement with the fact that it works on B and P frames and stores the motion vectors only as a result the CR is exceptionally high, though the PSNR suffers.

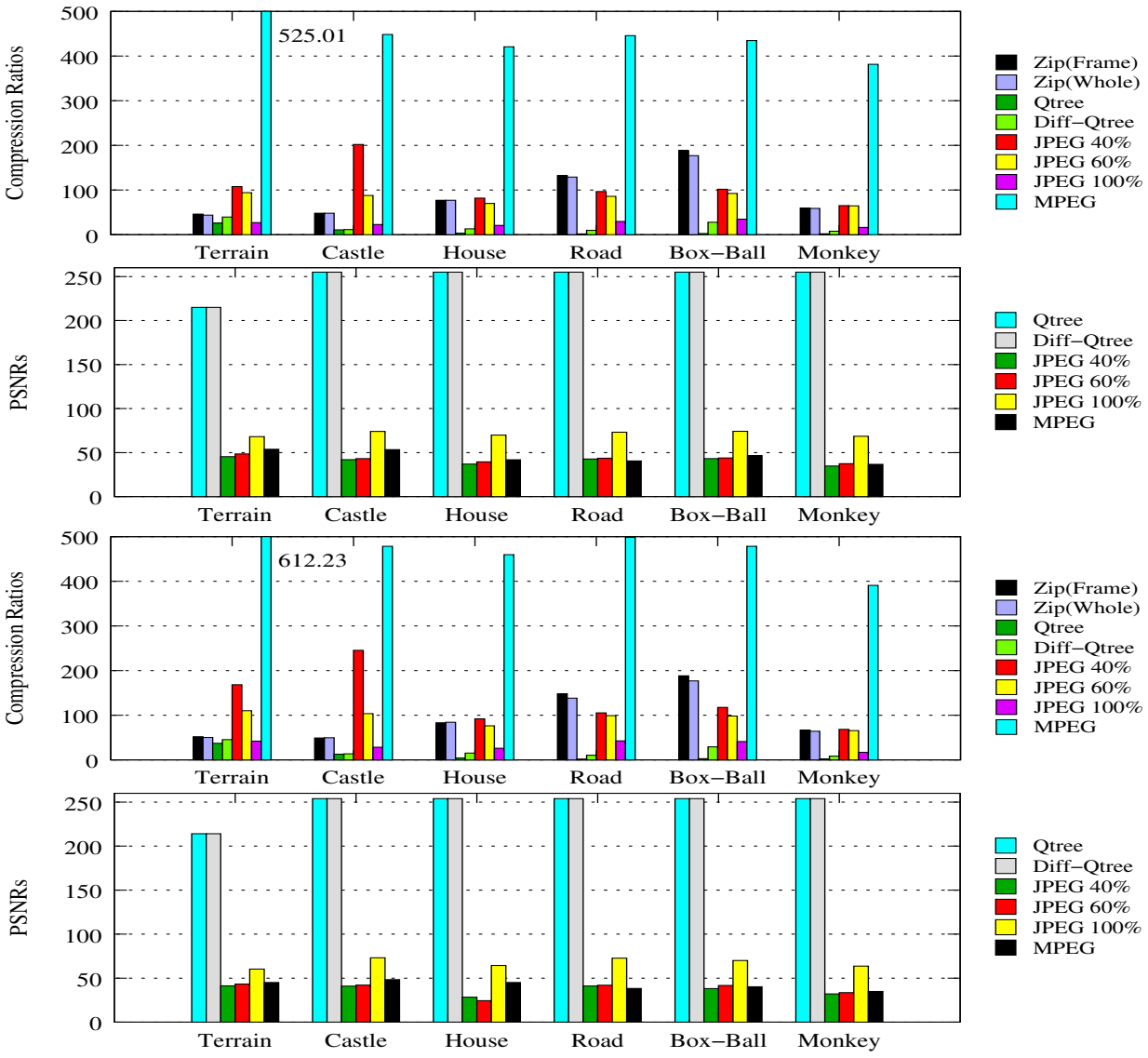

Figure 4.3: Results for all all 6 scenes where ; 1st bar graph shows CR with 16-bit frames; 2nd bar graph shows PSNR for 16-bit frames; 3rd bar graph shows CR with 8-bit frames; 4th bar graph shows PSNR for 8-bit frames

The next best CR is achieved by JPEG with 40% quality, but at the same time the PSNR falls sharply. This trend can be attributed to the fact that JPEG compression is in itself lossy so JPEG at 40% quality will be more lossy than 60% or 100% quality compressions. Zip compressions being lossless have PSNRs as infinity as there is no loss; however, CRs are significant values. On the other hand, Quadtree methods being lossless have lower CRs but at the same time high PSNR values. Differential Quadtree method has better compression ratio. We are storing lesser amount of redundant data so the CRs for each of the above mentioned methods with 8-bit depth values are better than those for 16-bit depth values. PSNR values show a reverse trend from that of CR. Also, when we add noise to the data sets, the compression ratios go down in all the cases as shown in Figure 4.2. Figure 4.2 shows that PSNR values for all the cases decrease as the noise level increase in the frames of the depth movies.

4.5 Summary

We experimented on compressing depth movies using the already existing methods for movie compression. We analyzed the results and found that these methods are generic methods for any movie and we need to exploit the properties of depth movies, keeping the integrity of depth values in the depth movies. In the next chapter, we introduce the concept of *Parametric Proxy-based compression* of *Depth Movies*.

Chapter 5

Parametric Proxy Based Compression of Depth Movies

A generic real-life scene can consist of moving or static humans and objects. They can be widely divided as rigid/non-rigid, dynamic/static with time in shape/position. The common example of such a scene is a moving human subject. Therefore, we concentrated our study of representing and compressing scenes with human subjects performing various actions, like dancing, playing, exercising, running, etc, as shown in Figure 5.2, represented using multiview depth movies. We use 16-bit integer depth values for the depth maps. This gives a compact and exact representation than using floating point, without compromising on the range of values in practice. A generic basic scheme of compression is as given in Figure 5.1.

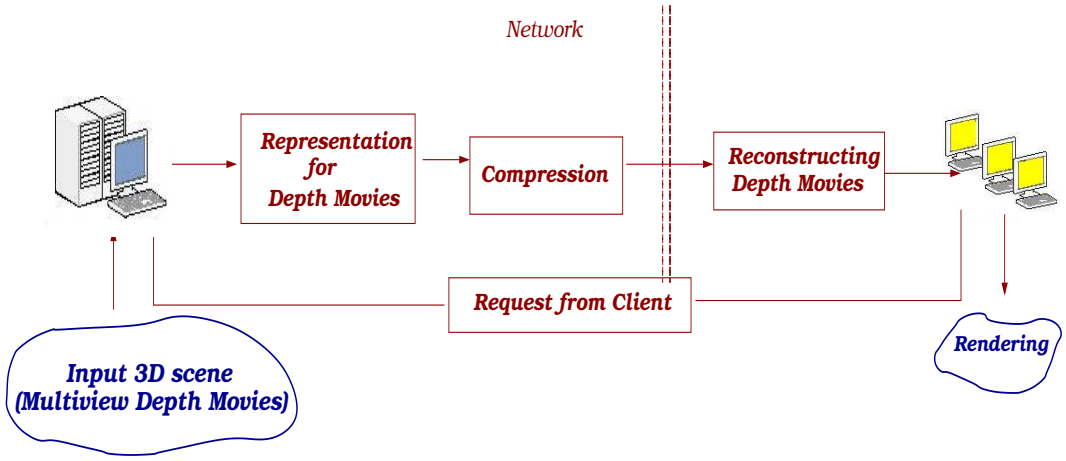

Figure 5.1: Basic scheme for compression.

In this chapter, we focus on the algorithms to compress such scenes using a parametric proxy representation for the scene.

5.1 Algorithm Overview

Parametric proxy as explained before, is a way of representing an object using a set of parameters and a standard model. We extend proxy-based compression to dynamic scenes or depth movies. Figure 5.3 presents the overview of compressing a generic 3D scene using a parametric proxy. The

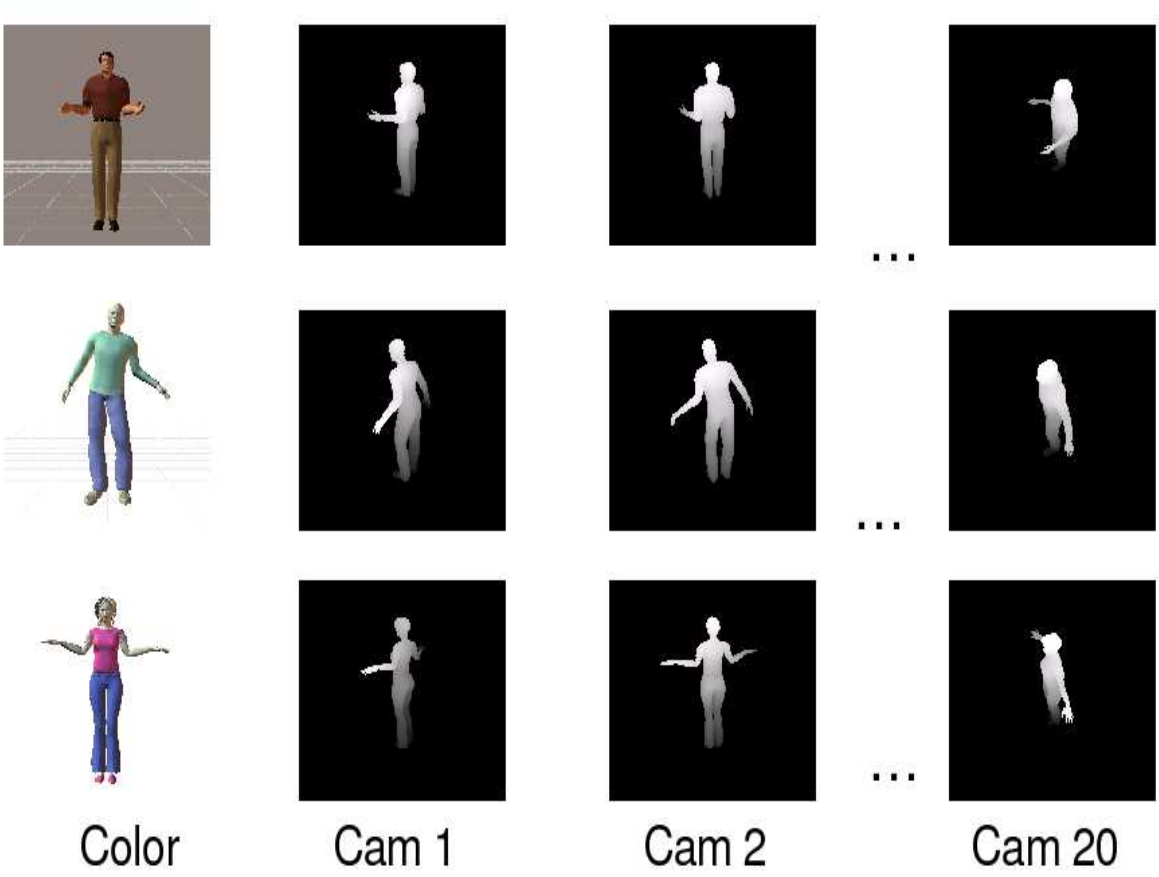

Figure 5.2: Figure Shows three humans in different poses from 20 cameras.

input to the system is a 3D scene comprising of multiview depth and texture maps of a human performing some action. A standard articulated human model is used as the parametric proxy. The first task is to fit a proxy to the point cloud for each frame. Figure 5.4 shows the depth maps from one of the cameras, the point cloud representation obtained by projecting depth estimates from all the cameras to a common reference frame and the fitted articulated human model. Details of our fitting tool are given in section 5.2.3. The depth maps of the proxy model for each frame serve as a prediction of the input depth maps. The prediction error between the original and the proxy depths is calculated by subtracting the proxy depth from the input depth. These residues are encoded and sent to the client along with the parameters of the proxy. At the client's side, the parameters are applied on the articulated human to generate the proxy model and its depth. Residues are added to these depth maps to get the real depth for rendering.

5.2 Proxy-Based Compression of Depth Movies

A proxy is a common *prediction* of a generic model. In this scheme, we assume that the scene consists of human subjects in different poses and performing different actions. The generic articulated model defines the proxy for the subject with changing bone parameters for every new frame. Angles of the bones are the parameters that define the proxy for each frame. First, this proxy is fitted to the input and then the parameters are computed and recorded. The residues are then processed using

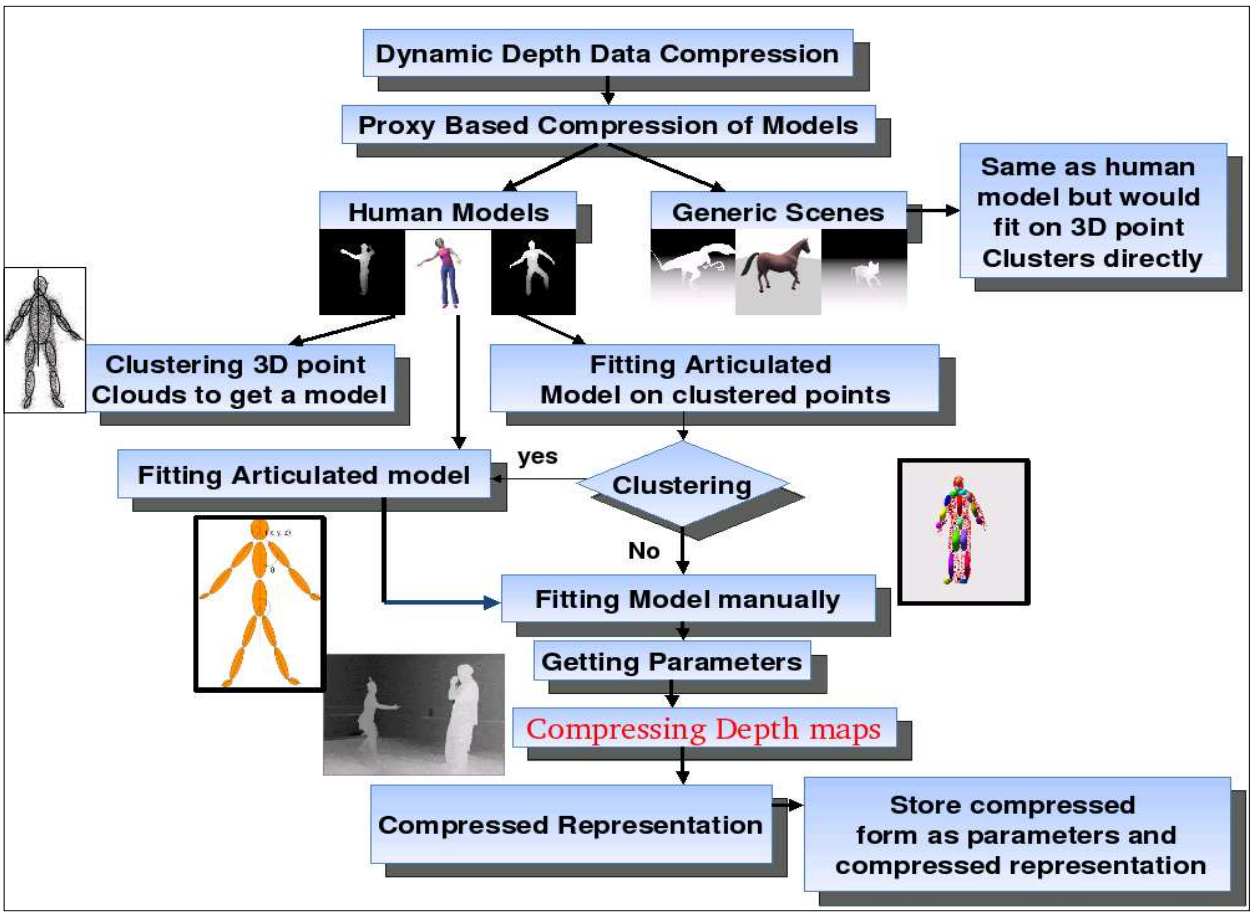

Figure 5.3: Overview of 3D Depth Movie compression and transmission

different schemes and algorithms to compress and send to the client.

5.2.1 Camera Setup

A setup of cameras is used as shown in Figure 5.5, with **m** cameras set in a room to capture a human in different poses. The calibration parameters of the cameras are also included in the representation so that the 2D sampling grid and the imaging rays of the depth image can be known in 3D space. The combination of a depth map and an image describes the local structure from one viewpoint and can be used to reconstruct the views of the scene from other viewpoints.

We use synthetic cameras to generate the depth movies. We also use datasets that are publicly available.

5.2.2 Parametric Proxy for Human Models

The parameters of a common articulated parameterized model are joint angles that change from one frame to another through the movie, but are same for all the cameras. The articulated model we use has 18 vertices as joints. The joint angles are the free variables of the model and serve as the parameters using which any position of the actor can be represented. The parameters change over time, but are same for all views at any time instant. Each bone position needs 3 parameters for the angles as heading (h), pitch (p) and roll (r). Thus, for each frame, a total of $18 \times 3 = 54$

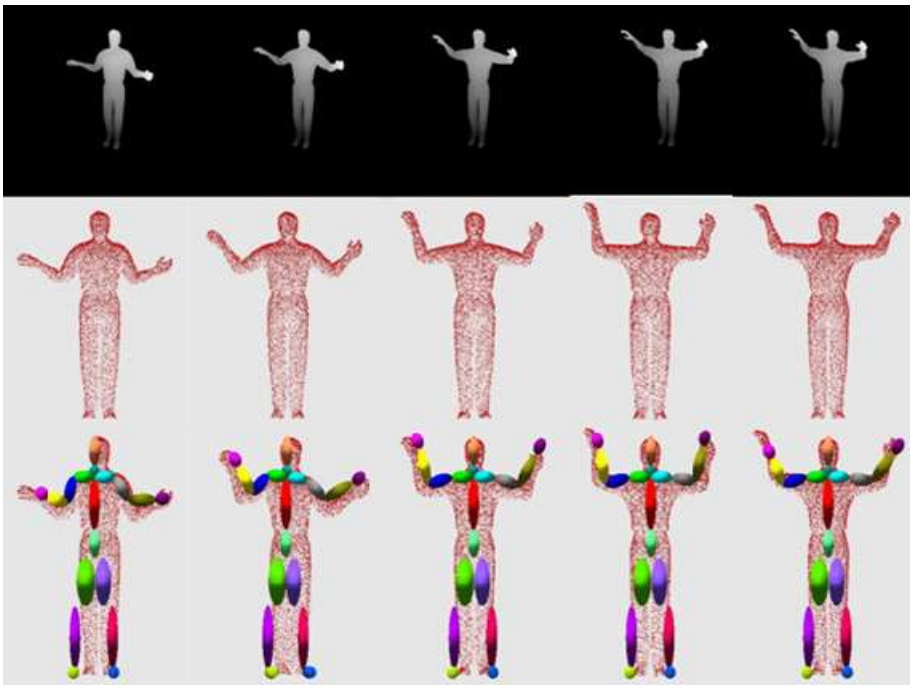

Figure 5.4: Articulated model fitting: Top row shows the depth maps for 5 frames, middle row shows the corresponding point clouds for each frame and the bottom row shows the corresponding fitted articulated model

parameters are needed. We use a single size parameter to scale the base model uniformly. The basic model is skeletal. A triangulated skinned model is used to generate the realistic articulated human model. We assume the skeletal and skinned model to be available to both the encoder and the decoder. This way, for each unique pose of human, only a few bytes are needed to represent the parameters.

The proxy model we used is a graphical model developed in AC3d and attached to it is a texture storing the information for the vertices and the colors related to it. Figure 5.8 and 5.9 show the hierarchy of the model, its 3D coordinates and the texture coordinates. This model is used for computing the parameters in our fitting tool.

5.2.3 Computing the Proxy Model

The input depth movie consists of mn depth maps and texture images for n frames from m cameras. We unproject the depth maps from each camera to get a point cloud for each frame and fit an articulated model to this representation. This can be achieved either automatically or manually. At present, we use a semi-automated tool to fit a skinned model for each frame to the point cloud. After getting the point cloud for each frame, the proxy model is aligned with it for the first frame. The next point cloud representation is progressively fit by changing a few parameters from the previous frame. The problem of fitting an articulated model from images and from depths has been studied in the past [2, 26, 86]. The fitting can be performed by optimizing the error between the skinned model and the point cloud as shown in Figure 5.6. Fitting subsequent frames is easier as the parameters change slowly. Since the model fitting is not our main focus, we fit the model interactively using a semi-automatic tool built for the purpose. We can fit the first frame of a sequence in less than 60 seconds and the subsequent frames in less than 15 second per frame, on

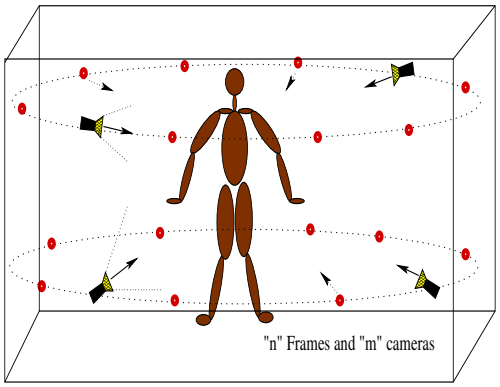

Figure 5.5: Setting of 20 cameras around the scene.

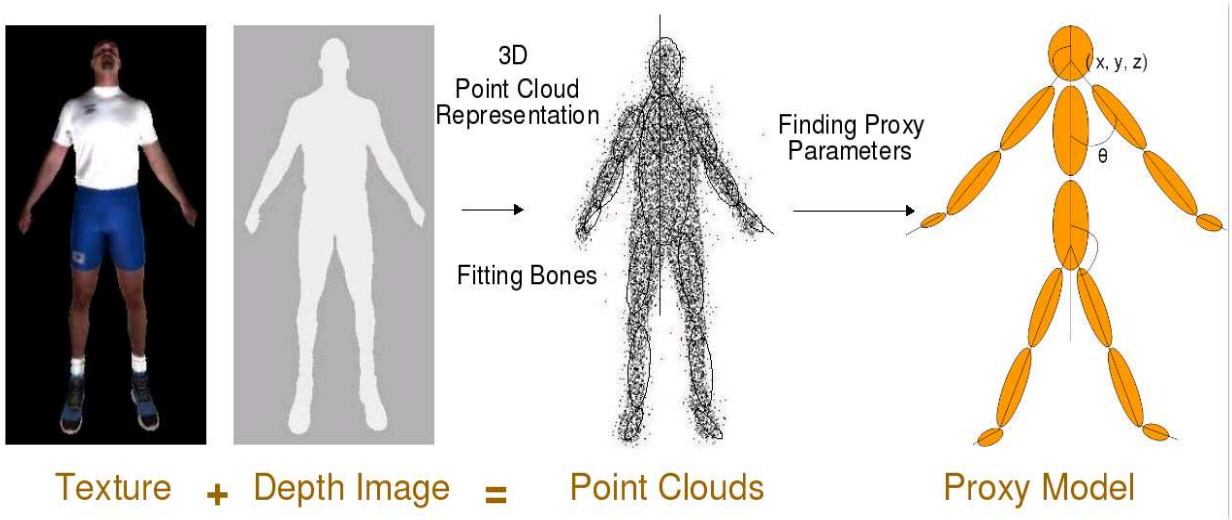

Figure 5.6: The primitive design of our Fitting Procedure.

an average. In the end, the parameters of the articulated model for each frame represent the scene parametrically as a proxy.

Our fitting tool is developed over the opensource pLib library. It provides a user interface for the user to fix the angle parameters of the proxy as per the input human subject. User can visualize the 3D point cloud and fit the 3D articulated model to it. For each unique pose of human, only a few bytes are needed for the parameters. These parameters are stored in a bone file and the depth maps of the proxy are captured from m cameras set in the tool.

5.2.4 Residue Computation

The fitted proxy model consisting of the generic skinned model and the bone-angle parameters, can be projected to each view to get its depth map and a mask bit. The mask has a 1 for every pixel part of the original depth map and helps identify the true projection of the subject so that the computation can be restricted to it. For each residue map, a foreground mask bit identifies pixels belonging to the actor in each view. The mask is obtained by thresholding the input depth values.

The difference between the actual input depth map and the proxy depth map is stored as the residue or the prediction error for each frame. The residue values are represented using a sign-

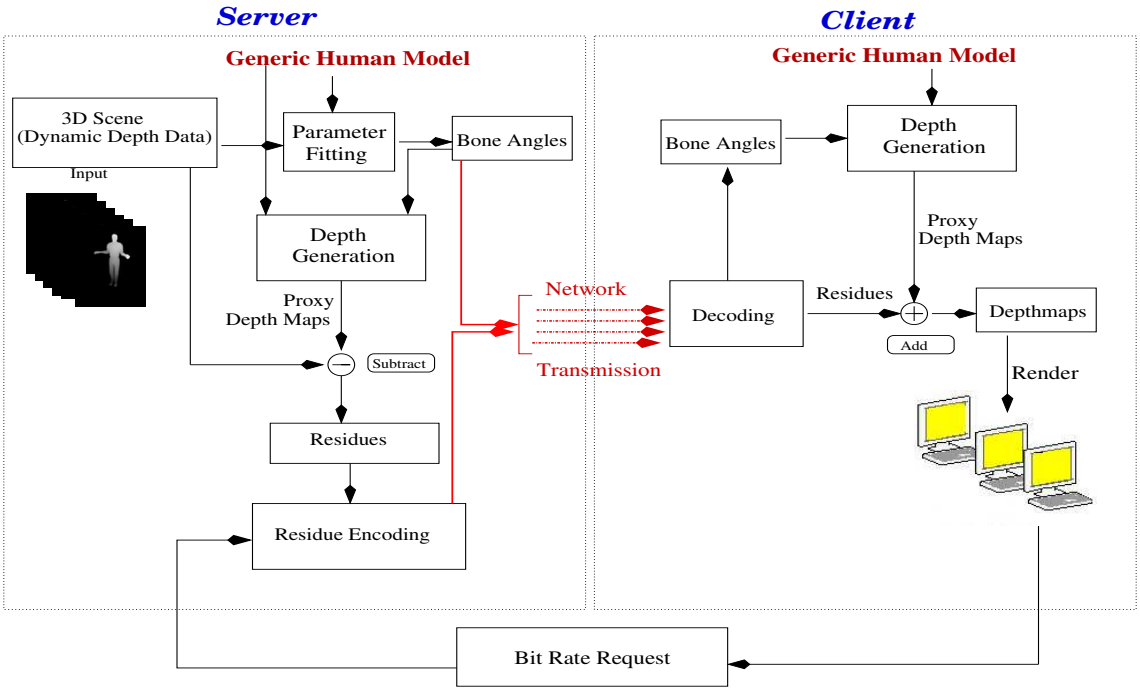

Figure 5.7: Overview of 3D Depth Movie compression and transmission using simple residue encoding.

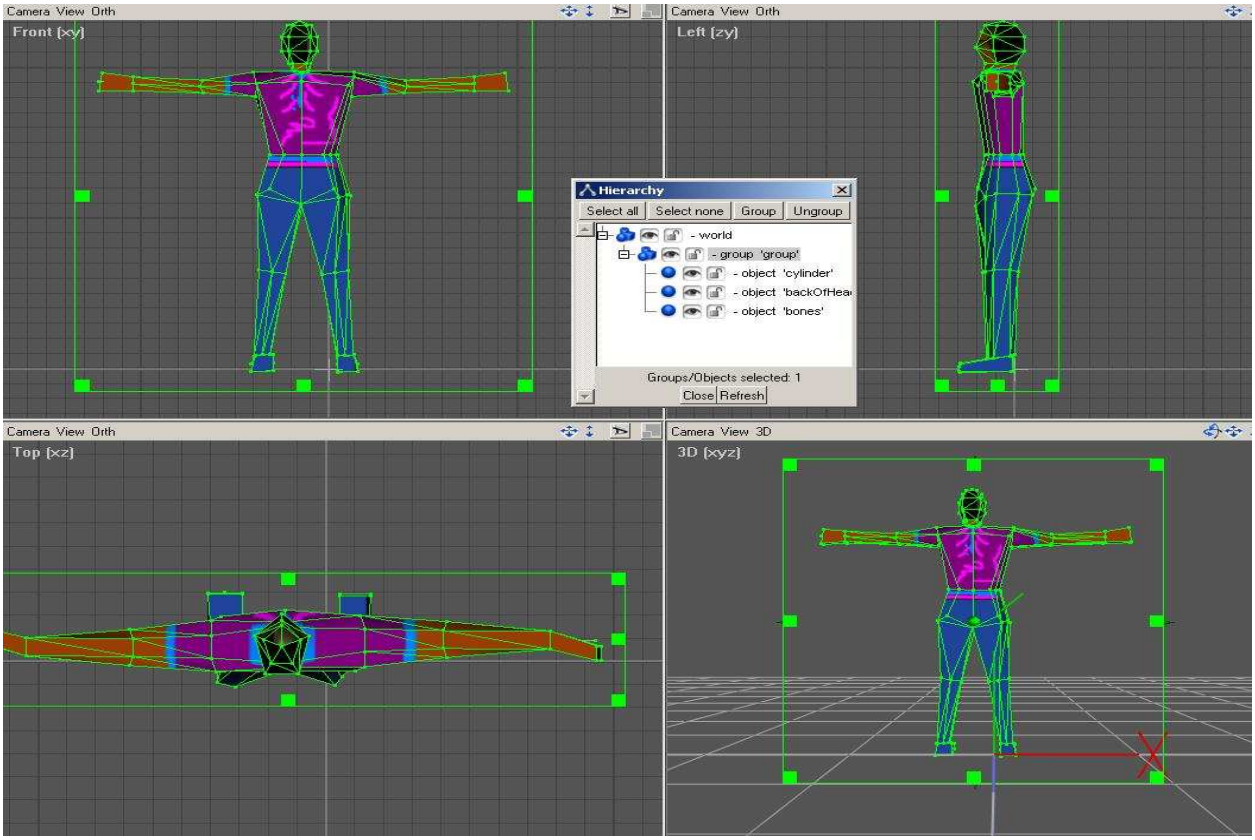

Figure 5.8: Generating a 3D graphical proxy model in AC3d.

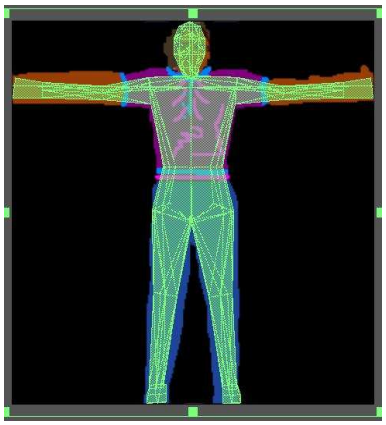

Figure 5.9: Fixing the texture coordinates to the proxy model.

magnitude format to facilitate bit-plane encoding of residues. The sign bit and the magnitude bits of the residue are stored separately. Thus, we have mn residues, mask and sign images for each frame and view. Figure 5.13 shows the input depth map, the fitted depth map and the difference between the two as the residue map and the sign image.

Algorithm 1 Direct Encoding At the Server Side for n frames from m cameras.

Input : n depth maps from each of m cameras, k-bit for encoding.
Output : mn residues, n parameter files (Bone-File).

1: Fit the parameters of the proxy for the input point cloud. Recover bones angles from each frame.
2: Project the depth of fitted proxy to each camera, to get the proxy depth maps.
3: Compute the residue R by subtracting proxy depth from input depth.
4: Compute mask images.
5: Encode R for the given bit rate as $R^{'}$, using mask.
6: Do a Run-Length Encoding (RLE) on Mask images.

5.3 Direct Encoding and Decoding of the Residues

The encoding at the server side is done as per Algorithm 1. The residues are small in range if the proxy model fits the original model well. Residues are more correlated across time since the difference of proxy and original varies very smoothly from one frame to another. Hence, residue movies compress better than the depth movies. We compress the residues either using MPEG or bit-plane encoding.

- *MPEG* Compression:
  The basic idea behind MPEG video compression is to remove spatial redundancy within a video frame and temporal redundancy between video frames. Motion-compensation is used to exploit temporal redundancy. The images in a video stream usually do not change much within small time intervals. The idea of motion-compensation is to encode a video frame based on other video frames temporally close to it. DCT-based (Discrete Cosine Transform) compression is used to reduce spatial redundancy. We encode all the frames from each of the

m cameras as a *MPEG* movie, to get m residue movies. This however could lose quality of the depths. MPEG can incur high reconstruction costs also.

- *Bit-Plane* Encoding:
  We break up an image into bit planes and apply run length coding to each plane. We combine the most significant bits for each pixel into one bit plane, the next most significant bits into another bit plane, and so on. This scheme allows for the use of a reduced number of bits for the residues, something that *MPEG* cannot. Thus, the residue can be encoded incrementally. This is important in a server-client setup. As more bandwidth is made available, more bits can be sent to increase the quality.

  The bit-plane coding scheme compresses the residues to as many bits as client asks for, by using the most significant bits. This is shown in Figure 5.7. The most significant k-bits are taken to represent a residue.

5.3.1 Compressed Representation

The representation to be sent to the rendering client includes the following. (a) The bone-angle parameters for each frame, (b) the mask bits for each view and each frame, (c) the sign bits for each frame and view, (d) m MPEG streams for the residue values when using MPEG or (d) the bit-plane encoded residues for each frame and each view when using bit-plane encoding. All data is zipped together at the end as a simple entropy-encoding method. This information is sent to the rendering client, which decodes the depths back. Figure 5.7 shows the server-client system thats used for the proxy based parametric compression of depth movies.

5.3.2 Decoding

Decoding at the client is done as given in Algorithm 2. First the encoded residues are extracted along with the bone parameters. These parameters are used to generate the proxy depth maps of the articulated model. The residues are then added to these depth maps to generate the depth maps of the subject at the client side. This can be used for immersive rendering of the scene along with its appearance. Figure 5.13 shows the decoded depth maps after adding 3-bits and 8-bits of depth maps generated using bone parameters at the client side

Algorithm 2 Decoding at the Client Side for $\mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots, \mathbf{n}$ frames

Input :mn residues, mn Masks and n bone parameter files.
Output :n depth maps from each of m cameras.

1: Get the bone parameters file for the frame.
2: Capture the proxy depth maps, D using the bone angles for articulated proxy model.
3: Do RLE Decoding to get the masks.
4: Get bit encoded residue R.
5: $D + R \rightarrow D^{'}$, use $D^{'}$ to render the 3D scene.

5.4 Difference Residue Encoding and Decoding

By compressing residues using bit-planes, we were able to exploit the spatial coherence of Depth movies. Compression of one depth map frame is totally incoherent with the next frame or any other frame in the movie. In this section, we propose another method to exploit the coherence between

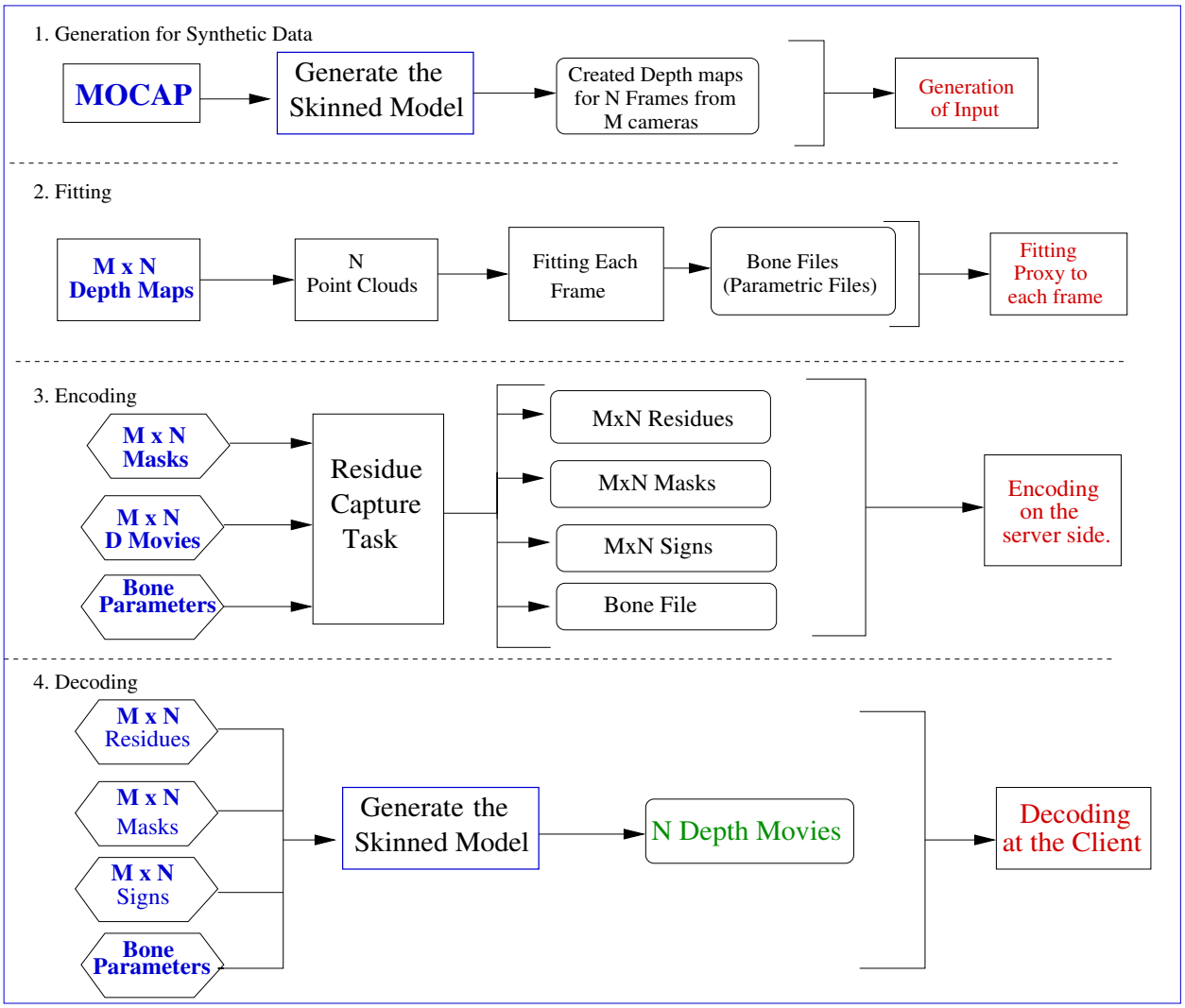

Figure 5.10: Steps for Encoding and Decoding at Server and Client

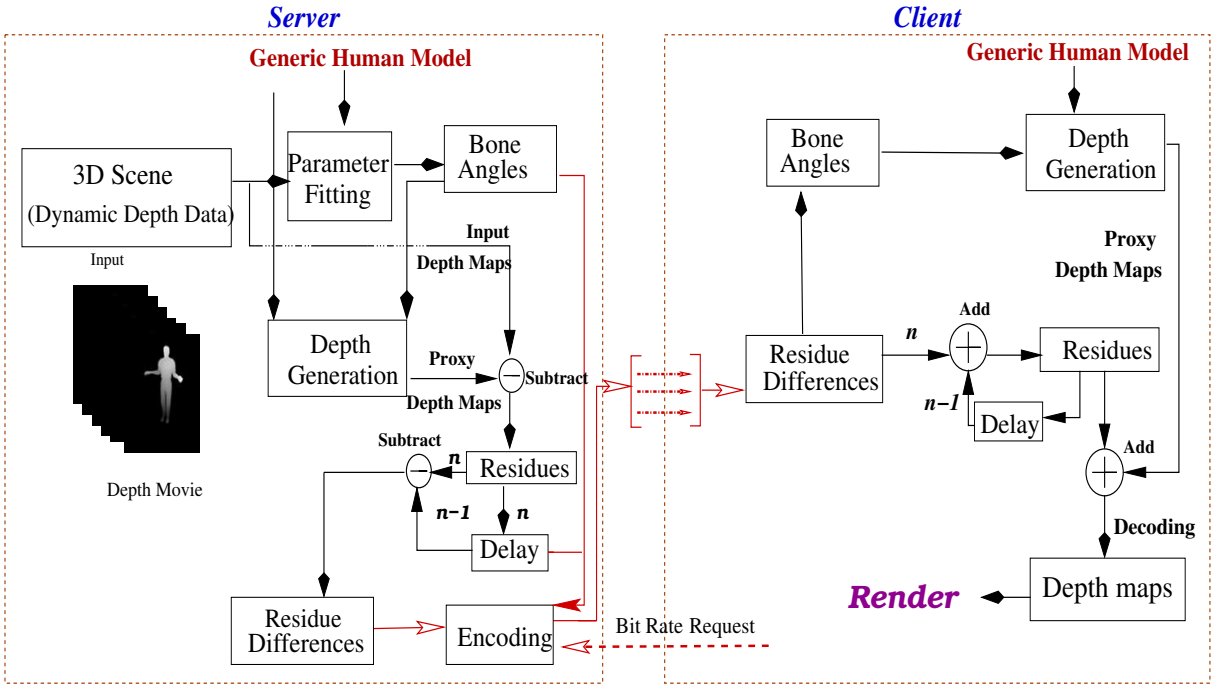

Figure 5.11: Overview of 3D Depth Movie compression and transmission using Difference of Residues encoding.

the two consecutive frames in a depth movie. The key idea is to compute the difference of residues between successive frames to exploit the temporal coherence.

Algorithm 3 Encoding depth maps for n frames from m views/cameras using Difference Residues

Input: n depth maps from each of m cameras, k for encoding.

Output: mn Residues Differences, n parameter values (one per frame)

1: Fit the proxy model to the point cloud for each frame and recover the bone-angle parameters.
2: Compute the mask image for each frame of each view.
3: Project the fitted proxy to each view and compute the proxy depth maps.
4: Compute the residue R by subtracting proxy depth from input depth.
5: Identify the key frames, and the block sizes between consecutive key frames.
6: Compute the residue difference D for the frames in a block by subtracting $R_{(i+1)}$ from R_i.
7: Encode residues R with K-bits and residue differences D with k-bits.
8: JBIG encode on mask images.

As discussed before, we exploit temporal correlation by computing *residue differences* as $D_i = R_i - R_{i-1}$, where R_i is the residue for frame i, as shown diagrammatically in Figure 5.11. This exploits the temporal relation between frames of the residues and uses it to compress the data. The residue differences are compressed as bit-planes which significantly reduces the space for data storage.

We encode the residue map as *blocks* of frames that contain one residue map or **R** frame of residue values and several **D** frames of residue differences. A block is a random-access unit and its length is determined by the requirements for random access. We have experimented with various lengths of the blocks.

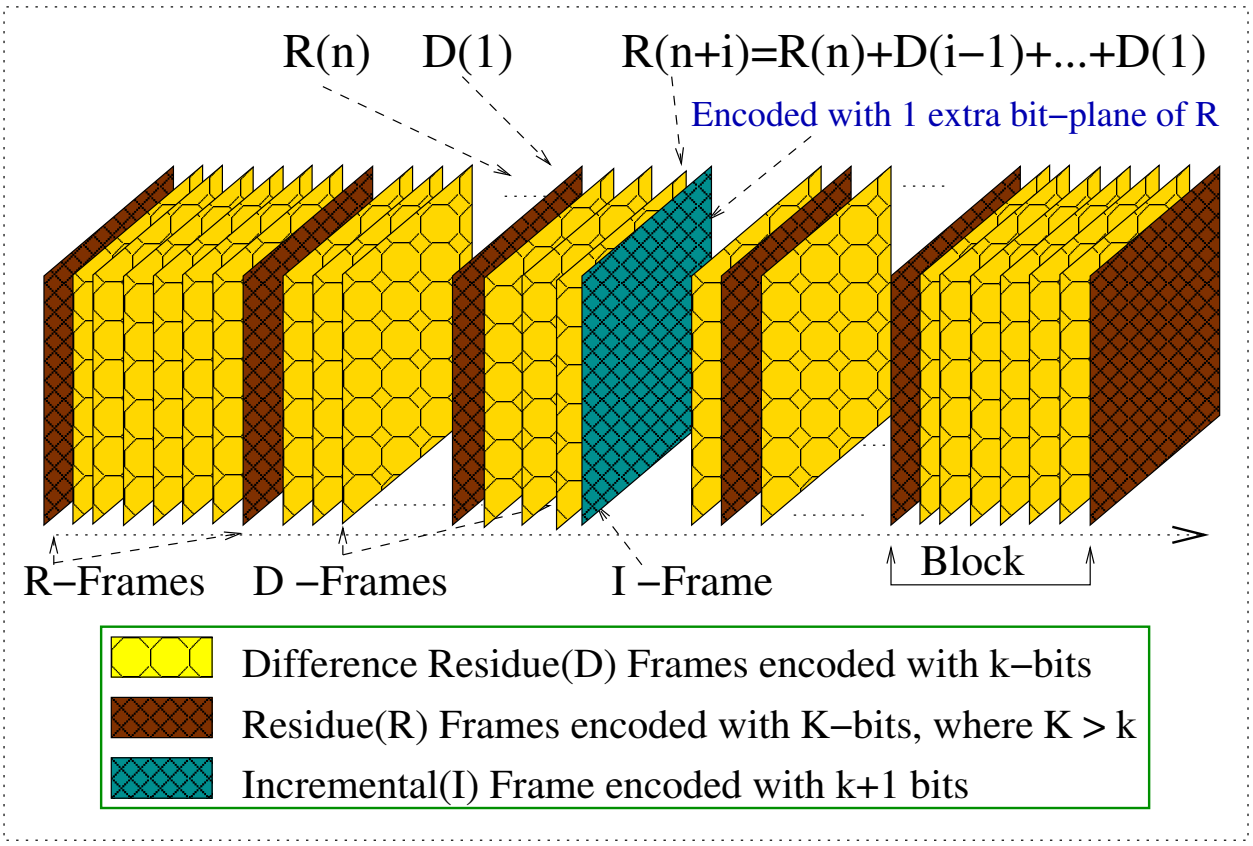

Figure 5.12: The structure of a block with R frames, D frames and optional I frames.

The **R** frames are linking *key frames* and are encoded with K most-significant bits of the residues. The **D** frames with k most-significant bits. K and k bits are specified as per the quality, network bandwidth and compression requirements.

Mask and sign images are bi-level maps with 0 and 1 values only. We compress these with JBIG encoding. ***JBIG*** is a lossless image compression standard from the Joint Bi-level Image Experts Group, standardized as ISO/IEC standard 11544 and as ITU-T recommendation T.82. JBIG was designed for compression of binary images, particularly for faxes, but can also be used on other images. JBIG uses a form of arithmetic coding patented by IBM known as the Q-coder. It bases the probabilities of each bit on the previous bits and the previous lines of the picture. In order to allow compressing and decompressing images in scanning order, it does not reference future bits. JBIG also supports progressive transmission with small (around 5%) overheads.

Different quality points can be obtained by varying K and k, which can be varied in a real-time client-server setup. We also use an incremental representation to exploit any additional available bandwidth. We send the next i.e., $(K+1)^{th}$ bit-plane of the residue as an incremental frame or **I** frame when the resolution has to be increased. The value received is added to the current **R** frame, thus improving the quality of all frames till the end of the block. As shown in Figure 5.12, an extra bit at **I** frame provides the increment in the bit representation for the following frames, thus there

on increasing the quality of the movie.

$$R_i = R_0 + D_1 + D_2 + \cdots + D_{i-1} \quad (5.1)$$
$$R_i^{'} = R_0^{'} + D_1 + D_2 + \cdots + D_{i-1} \quad (5.2)$$
$$(5.3)$$

Equation 5.1 shows R_0 encoded with K bits and subsequent D frames encoded with k bits. When 1 extra bit is added to the R frame as shown in Equation 5.2, $R_0 \rightarrow R_0^{'}$, and the subsequent R_i frames also get better representation. Thus, the quality of $R_i^{'}$ gets more than R_i.

The compression using difference residues can be summarized as following steps:

1. Exploit temporal correlation by computing *residue differences* as $D_i = R_i - R_{i-1}$, where R_i is the residue for frame i.
2. Encode the residue map as *blocks* that contain one **R** frame of residue values and several **D** frames of residue differences. A block is a random-access unit and its length is determined by the requirements for random access.
3. Encode **R** frames with K most-significant bits of the residues and the **D** frames with k most-significant bits of the residue difference.
4. Encode mask bits using the JBIG algorithm.

5.4.1 Compressed Representation

The data to be sent to the rendering client includes the following. (a) The bone-angle parameters for each frame, (b) the mask bits for each view and each frame, (c) the sign bits for each frame and view, (d) m MPEG streams for the residue/difference residue, **R**/**D**, values when using MPEG or (d) the bit-plane encoded residues/difference residues, **R**/**D**, for each frame and each view when using bit-plane encoding.

The articulated model and the skinning triangles are available at both ends and need not be transmitted. The model parameters (bone-angles) take only a few bytes per frame. The mask and sign images contain only 0 and 1 values. They are compressed using JBIG compression scheme as discussed before. We also tried run-length encoding and other such schemes to compress bit-maps, but JBIG gave the best performance in all cases as its based on probability analysis and is lossless too. When we used MPEG for coding R and D frames, we encoded the data using the `ffmpeg` library with the given bit-rate as asked by clients. Otherwise, when we used bit-plane encoding, the **R** and **D** frames are compressed into a long sequence of bits and then entropy coded using `zip`.

The frames of each view are also independent, with some using residues and others using residue differences. All data is zipped together at the end. This packed information is sent to the rendering client, which decodes the depths back. The final representation contains one parameter file per frame, one mask, one sign bit, and an **R** or **D** plane per frame per view. The data for a whole block (between two **R** frames for bit-plane coding or between two **I** frames for MPEG) needs to be together logically and can be treated like a package to be sent to the client.

5.4.2 Decoding

The decoding process is shown Figure 5.11 and is described in Algorithm 4. Each block of data is treated independently by the client. The bone-angle parameters are applied to the articulated model. The resulting model projected to the camera of each depth stream, giving the proxy depth

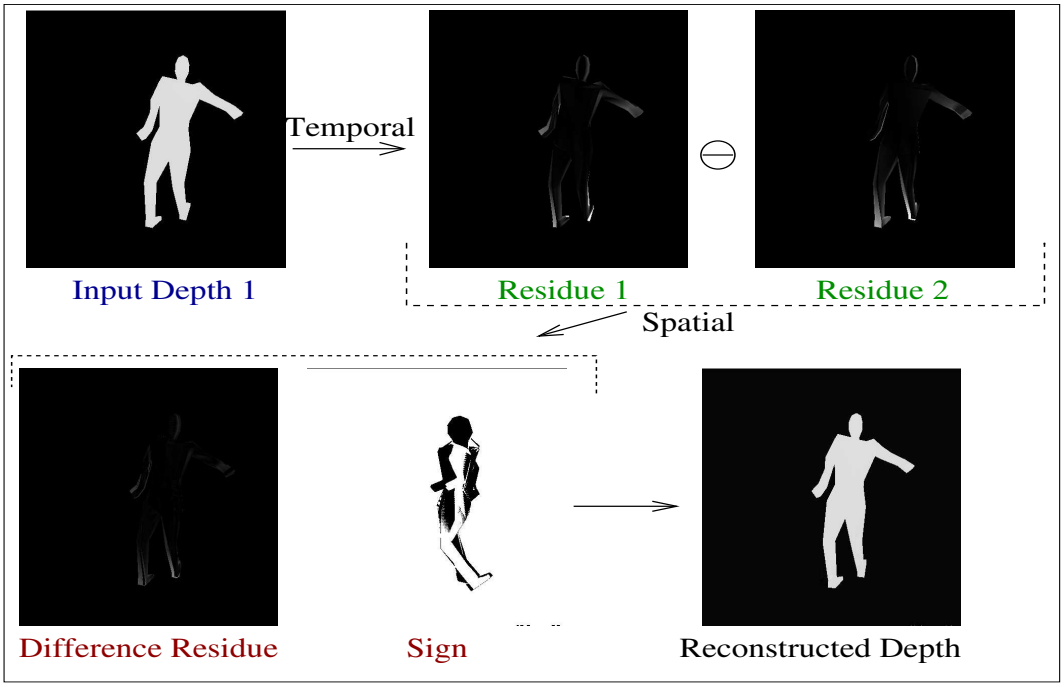

Figure 5.13: Results showing input depth, residues, residue differences, sign and reconstructed depth.

Algorithm 4 Decoding at the client side for $\mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots, \mathbf{n}$ frames

Input: mn residues differences, mn masks and n bone-angle parameters

Output: mn depth maps for each view/frame for rendering

1: Get the bone-angle parameters for the frame from the input stream.
2: Capture the proxy depth maps D using the parametric proxy model for each frame and view.
3: Decode the sign bit and mask bit.
4: Compute the residue difference R_D for each frame and view.
5: Use equation 5.2 to get R.
6: $D' \leftarrow D + R$. Use D' to render the 3D scene.

maps. The residue maps are recovered from the **R** frames of the packet. The residue maps are added to the proxy depth map to get the decoded depth map for that view and frame. For **D** frames, the residue differences recovered from the packet are added to the current residue map R_i to get the next residue map R_{i+1}, which is added to the proxy depth map for that frame to get the decoded depth values as given in Equation 5.2. The foreground mask is needed to keep track of the changing silhouette of the actor. The depth map of frame $i+1$ may include pixels not in frame i. If frame $i+1$ is a **D** frame, the current average residue value is used as the reference for the residue difference. If the incremental frame arrives, the bit-plane for it is assembled and added to the current running residue R'_i. Improved quality results till the end of the current block. Figure 5.13 shows the effect of the algorithm on a given depth map sequence from one of the cameras.

5.5 Summary

In this chapter, we discussed the proxy-based compression schemes for multiview depth movies of a scene involving dynamic human action. We showed that for sending such huge 3D information with accurate and most useful data, parametric proxy method can be good along with the residue encoding algorithm and difference residue encoding. A summary of the system with depth difference residue encoding and decoding scheme is summarized in Figure **??**. We used a simple articulated model as the proxy and joint angles as the parameters that approximate the model for a particular frame. The schemes explained in this chapter provide good compression ratios at acceptable quality levels, as shown in the next chapter. The proxy-based schemes provide several controls on the amount of data to be sent. This makes it ideal for sending the captured data for applications like 3D teleconferencing. In the next chapter, we give details about our datasets, the experiments we did on them and an analysis report.

Chapter 6

Experimental Results

6.1 Datasets

The Proxy based compression method for depth movies was experimented on synthetic and real datasets. We used two kinds of datasets, synthetic (simulated using real MOCAP datasets on synthetic human models) and real (real motions of a real human being).

6.1.1 Synthetic Datasets

We created synthetic datasets with many simulated real life MOCAP (Motion capture) datasets in POSER. Random datasets with POSER act as preliminary datasets to analyze depth movies with no significant real actions. We were able to simulate real life actions using MOCAP in POSER. We used CMU's MOCAP repository in achieving long sequences of significant actions like running, dancing, etc.

A standard POSER human model was animated using the MOCAP parameters and 16-bit depth maps were captured from 20 viewpoints. 16-bit depthmaps help in capturing depth to 65535 meters. The joint-angle parameters for compression are very similar to the MOCAP files. We, however, added noise to the joint angles to simulate bad fitting of the model to real data. We also added slow-varying noise to the depth values to simulate errors of the depth-recovery process. The depth noise has a small random component at each pixel which rides on top of a larger component that varies slowly over the whole depth map.

6.1.2 Real Dataset

Real Datasets for depth movies are hard to find. We have shown our experiments on a Doo-Young karate sequence from ETH-Z as shown in Figure 7.1. The dataset consists of frame sequences rendered from 15 different camera views located in a hemisphere around the scene, background images, segmentation masks, depth images, and camera parameters. The recorded scene contains human performing punches of a Kung-Fu fighter. The camera setup and the further information about the dataset has been given in appendix-2.

6.2 Experimental Setup

Fitting procedure is simple with minimal human involvement. We can fit the first frame of a sequence in less than 60 seconds and the subsequent frames in less than 15 seconds, on an average.

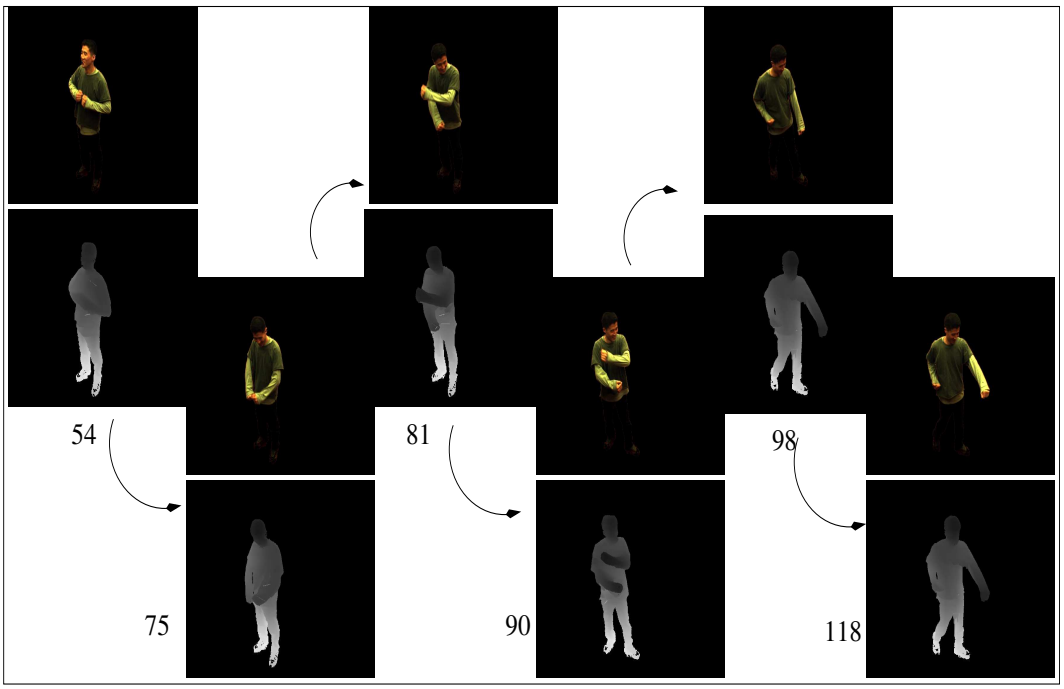

Figure 6.1: Doo-Young Dataset Depth movie from ETH-Z.

6.2.1 Synthetic Data Unit

As we show in Figure 6.3, the synthetic data that we get from POSER is processed in the ADU to generate the input depthmaps and textures for each N frames from M cameras. When we get the data from the Poser we get the additional information of the 3D Model representing the initial frame and α the initial bone angles for each pose of the human subject. This the major difference between real and the synthetic data and to make our experiments of the final compression rich we plan to exploit this difference, by creating a varied scenarios of experiments. In the second step from ADU the information of M + $\alpha^{'}$ is given to the Residue Unit. In RU we generate the difference residue between the input model and the present frame bone configuration $\alpha^{'}$ and we also give the new approximate representation of the scene represented as M + α. This approximate representation of the scene is known as the Proxy for the human subject. Residue is represented as the difference between the original model and the proxy model. We store this residue as a representation for the spatial aspect of the compression. The temporal effect of the dynamic data can be compressed either using MPEG compression or using Difference encoding. Difference encoding scheme stores the difference along the temporal line. The decompression is done after adding the differences to the initial frames. Thus residue, R can be represented as the difference between X (original M + $\alpha^{'}$) and A (proxy as M + α). This residue is compressed using varied number of bits (1 to 8) as the representation for the residues. This is then encoded using difference encoding or MPEG for a motion vector representation of the same. In the decompression unit, the encoded bit representation is decoded back to give the depthmaps for each M cameras in N frames. This is the used for PSNR computation. After decoding, the residue bits are added to the proxy M + $\alpha^{'}$, and hence quality is governed by the number of bits added to get to the original depthmaps. If the original depthmap of $n^{th} frame$ and from m^{th} camera is represented as X, then after DU we get (X - A) + $R^{'}$, as the decompressed depth map. Thus the difference between X and (X - A) + $R^{'}$ is majorly reflected in the PSNR computation.

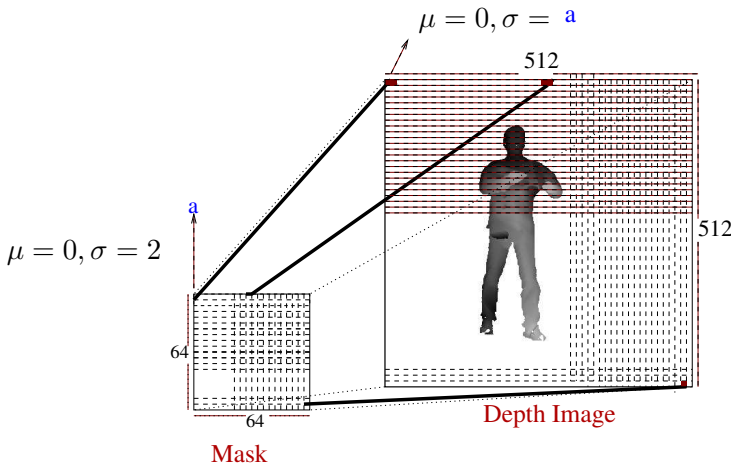

Figure 6.2: Double-gaussian noise insertion method for the depth maps.

α in the Figure 6.3 is denoted as the bone parameters for each n^{th} frame. $\alpha^{'}$ represents the noisy version of α obtained from the input data. Here noise represents the slight variations in the bone angle parameters for the bones of the human subject/ model/ proxy.

Different experiments and their variations/ combinations to be conducted in such a scenario of synthetic data can be listed as -

1. Keeping Same M for input and the proxy, but changing the α values, i.e., representing the M by 18 bones or 28 bones(as specified by POSER).

2. Keeping the same number bone structure but varying the proxy model and the input model.

3. Changing both model M and number of bone elements in α.

4. Input Depth map with different amount noise variations. The noise that we refer here is a kind of gaussian noise that varies locally and very smoothly. Basically, the mask of 8x8 blocks is created on the depth map and each of these blocks have a gaussian of their own with a σ and μ specified.

5. Compression factors can be varied by either doing an MPEG compression or by difference encoding.

6.2.2 Noise Generation

Since, real data of depth movies isn't available freely, we experiment with MOCAP datasets simulated as real data by adding two noise. We induces two kinds of noise to the MOCAP data. First, the bone noise is added to the bone angles. Stereo noise is a very general noise occurring in real life depth maps, so for the second noise, we induce a localized but correlated noise to the mn depth maps. Generated sigma is chosen to govern the gaussian over the whole depth map, which makes the noise correlated. In the next step, the localized noise with the mean as the value from previous common gaussian is applied as a localized gaussian over some $k \times k$ mask in the image. Thus, $512/k \times 512/k$ blocks get a localized but centrally correlated noise that signifies stereo noise.

We induce preprocessing noise in order to have the depth maps and bone files from input and the Parametric Proxy Generation Tool (PPGT), different in values, as is expected to be in case of real data. We see that Depth Noise hasn't been a simple noise. We take proper care to introduce the

Algorithm 5 Preprocessing step for MOCAP datasets.

Require: ∃ MOCAP file

repeat
  for $i = 1$ to n **do**
    Get the bone parameters file, B-File.
    Use B-FILE to generate the proxy depth maps from m cameras in the MOCAP file.
    Induce Bone Noise
    for $b = 1$ to 18x3, where ,18=number of bones, 3=h,p,r angles **do**
      Convert $b \Rightarrow b^{'}$, where $b^{'} = b \pm (rand)$ x $factor$, where $factor = 0.2, 0.3, 0.4$.
    end for
    Induce Depth Noise
    Divide image into 8x8 blocks of pixels.
    Compute a Gaussian mask of $\mu = 0$, and $\sigma = 2$ and (512/8)x(512/8) size.
    Use each pixel value of this mask to be the σ for each 8x8 block created before, to create the Gaussian noise with $\mu = 0$.
  end for
until All the Frames are done.

depth noise because, if depth gets random noise, then the 3D subjects in the scene get deformed. So, by our method, depth values get a local deformation and thus the shapes remain intact.

1. Calculate the Bone parameters from the MOCAP file and make the bone file, B-FILE compatible with the PPGT.

2. Use B-FILE to render the depth maps in PPGT for each frame in the MOCAP file.

3. Induce Bone Noise
   for each bone value b in 18 x 3 space
   where, 18=number of bones, 3=h,p,r angles
   Convert b to $b^{'}$, where $b^{'} = b \pm (rand)$ x $factor$, where $factor = 0.2, 0.3, 0.4$.

4. Induce Depth Noise (our image size is 512x512)

   - Divide image into 8x8 blocks of pixels.
   - Compute a Gaussian mask of $\mu = 0$, and $\sigma = 2$ and $(512/8)x(512/8)$ size.
   - Use each pixel value of this mask to be the σ for each 8x8 block created before, to create the Gaussian noise with $\mu = 0$.

6.2.3 Real Data Unit

We have real data from MSR, the "breakdance" seqeunce and the "Ballet dance" sequence and from ETH-Z, the "taekwando" sequence. MSR data has N=100 frames from M=8 cameras. ETH-Z data has N=100 frames and M=3 cameras. As we can see in the Figure 6.4, the major difference between the processing of the synthetic and the real data is that as we don't have the original input model with us, we have no prior knowledge of M and α. Thus, RDU in contrast with ADU doesn't generate the depthmaps and textures for each frame, instead it is provided as the dataset. Now, in the second step, as we don't have a prior information for M + α, we approximately fit the real data using our human proxy fitting tool for each frame in FU. This tool comes in handy for guessing the

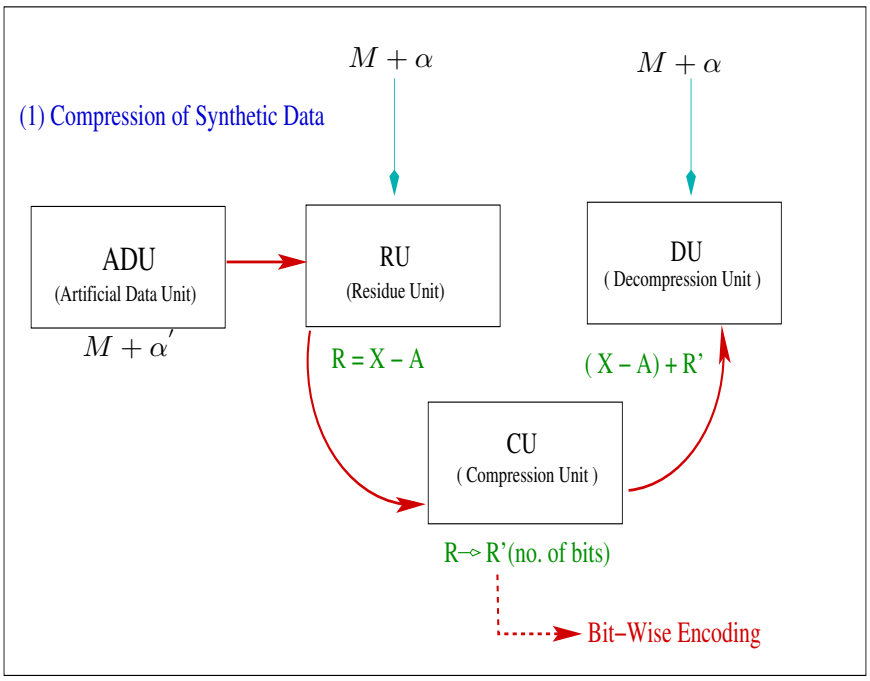

Figure 6.3: A schematic diagram to show the steps to be followed for experimenting the proxy based dynamic depth compression strategy with synthetic MOCAP data

approximate bone angles for each frame and also it gives the depthmaps for the proxy with that α configuration from M cameras. Thus, using this fitting tool M(proxy) + α is obtained with a noise in alpha itself. The rest of the units, RU, CU and DU are same as before.

Different experiments and their variations/ combinations to be conducted in such a scenario of real data can be listed as -

1. Input Depth map with different amount noise variations. The noise that we refer here is a kind of gaussian noise that varies locally and very smoothly. Basically, the mask of 8x8 blocks is created on the depth map and each of these blocks have a gaussian of their own with a σ and μ specified.

2. Compression factors can be varied by either doing an MPEG compression or by difference encoding.

6.3 Results of Residue Encoding

The MOCAP data and the bone angle parameters are the same, except for the bone noise. The proxy model is articulated using the noisy bone angles to generate the depth maps. Residues are generated by subtracting the fitted proxy depth maps from the input depth maps, which uses the MOCAP data without noise. These residues are compressed using both schemes and with varying number of bits.

For decoding, bone parameters are given to the proxy model available to the client to get the proxy depth maps. The decoded residues are added to approximate the original depth maps as shown in Figure 6.3. The quality of the reconstruction is governed by the quality of the residues.

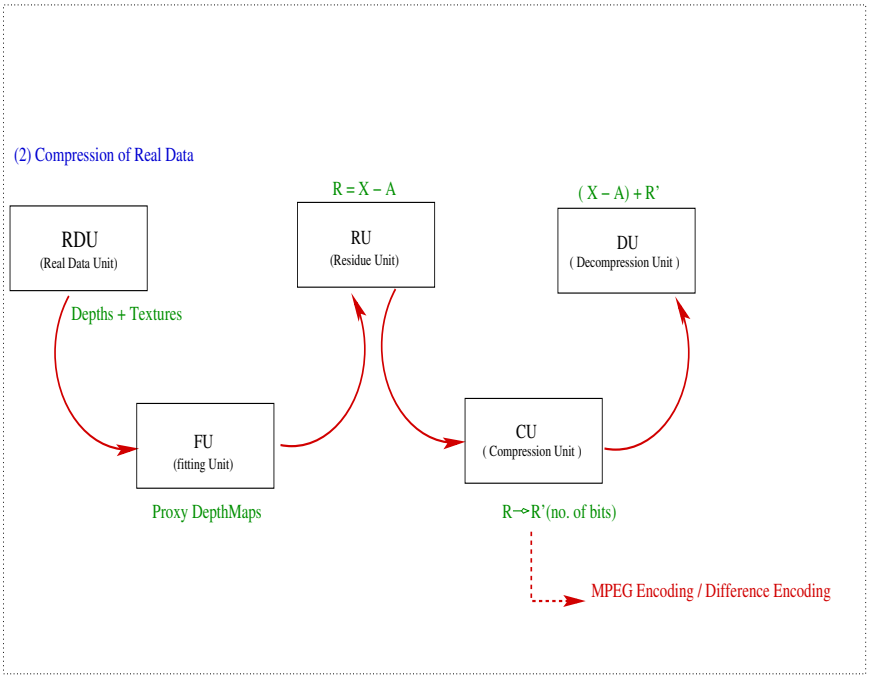

Figure 6.4: A schematic diagram to show the steps to be followed for experimenting the proxy based dynamic depth compression strategy with Real Data

If residues are encoded losslessly, the original depth maps can be reconstructed. Lossy encoding using MPEG on the residues or fewer number of bits will distort the model. For comparison, we also encode the original depth maps directly using MPEG and compare the reconstruction results for all three options.

Results on a few synthetic datasets are shown in Table 6.1. We experimented on 5 datasets with 100 to 300 frames. The compression ratio is with respect to the original, uncompressed depth maps. The PSNR is calculated by comparing the reconstructed depth maps with the input depth maps. MPEG compression exploits the spatial and temporal redundancy in the data so sometimes it improves over the proxy based compression scheme. The bit-plane scheme provides high compression and moderate quality at low number of bits and good compression and excellent quality at higher number of bits. It provides totally random access of the depths of individual frames. Most interestingly, the option of using 0 bits of residue provides a very low bit-rate approximation of the input scene. The error of such approximation is somewhat high as the reconstructed shape at the client will be that of the articulated model. Figure 6.5 plots the PSNR and the compression ratio against the number of bits used to encode the residues for one dataset. It can be seen that the PSNR varies slowly with the number of bits, but the compression ratio of bit-plane encoding is very good. The MPEG compression of depth and residues (MPEG-R and MPEG-D in Table 6.1) provides decent compression and quality, but the bit-plane encoding scheme provides more size to quality tradeoffs to suit any situation.

#bits used	Throw (100 frames)	Spin Kick (100 fr.)	Break dance (120 fr.)	Dance (230 fr.)	FootBall (300 fr.)
0	10584/10.11	19859/10.59	12563/9.35	11309/10.33	14391/12.34
1	4927/24.56	9354/25.56	5397/23.31	6371/22.60	6927/26.07
2	4839/25.03	9247/26.85	5036/24.56	6005/23.91	6543/28.11
3	4737/25.41	9218/28.31	4821/25.91	5941/24.35	6307/29.67
4	4650/25.52	9137/29.92	4598/26.74	5803/25.69	6251/30.93
5	4542/26.82	9116/31.53	4512/28.34	5749/26.78	6201/31.16
6	4458/28.93	9069/32.63	4439/29.16	5710/28.54	6149/33.32
7	4386/30.91	9032/34.17	4387/29.55	5673/29.01	6111/34.51
8	4257/32.57	8959/35.09	4297/29.97	5592/29.36	6021/36.93
MPEG-R	4239/26.49	9129/29.43	4353/26.84	5549/25.81	5945/29.91
MPEG-D	4154/24.58	8938/28.23	4155/25.35	5302/25.96	5713/27.82

Table 6.1: Compression ratios and PSNR for different datasets with varying bit-wise compression, MPEG encoding of the residues (MPEG-R) and MPEG encoding of the input depth maps (MPEG-D). The first number is the compression ratio and the second the PSNR.

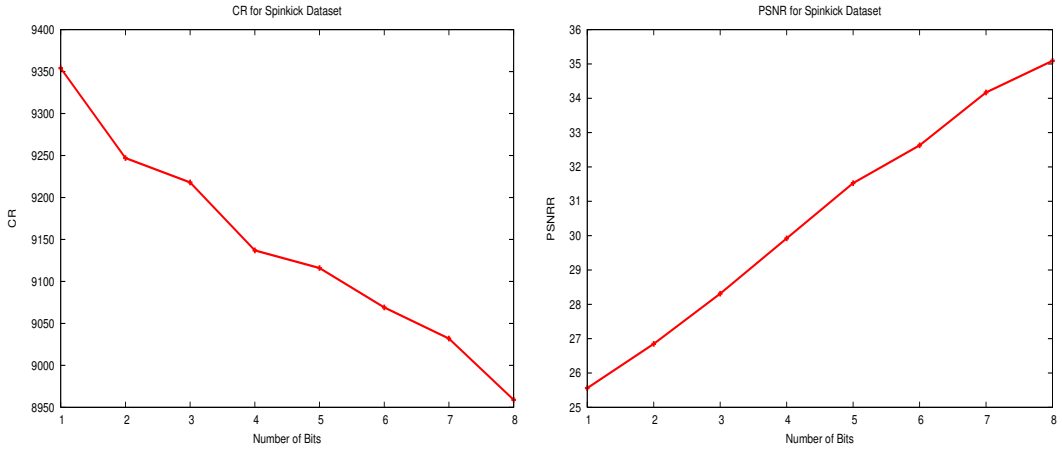

Figure 6.5: Results for 1-8 bit-wise encoding on Spinkick dataset with 100 frames

6.4 Results of Difference Residue Encoding

The following results are using Difference Encoding with various combinations of bit encoding, noise values, block size, etc., on 3 MOCAP generated datasets and one real dataset.

Results on a few synthetic datasets are shown in graphs of Figure 6.8. We experimented on three MOCAP sequences, Indian dance, Ballet and Exercise, each with around 300-400 frames each. The compression ratio is with respect to the original, uncompressed depth maps. The PSNR is calculated by comparing the reconstructed depth maps with the input depth maps. The residue compression exploits the spatial redundancy but we do residue difference encoding scheme that exploits both the temporal and spatial aspects of Depth movies. The bit-plane scheme provides high compression and moderate quality at low number of bits and good compression and excellent quality at higher number of bits. As we increase the number of bits in encoding, the compression ratio, as expected, decreases with increase in the quality factor. Also, it provides totally random access of the depths of individual frames. Most interestingly, the option of using 0 bits of residue

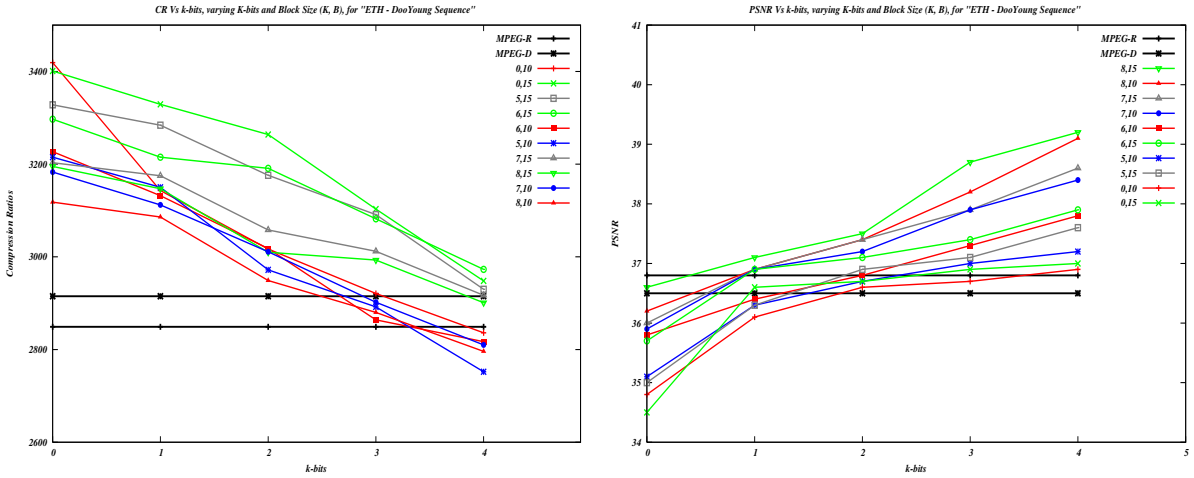

Figure 6.6: Compression ratio and PSNR for Doo-Young real dynamic depth movie dataset.

provides a very low bit-rate approximation of the input scene.

Other than joint angle noise and depth noise, we varied the block sizes for coding the depth movies to get nice compression figures with good quality. We observed that as the block size increases, the average compression ratio increases and the PSNR decreases. Thus, higher blocks are preferable for coding as encoding with K bits is lesser than k bits for D-Frames, where $K > k$. Keeping the block size constant, with increasing the joint noise the compression ratio reduces as higher bits are needed to fully represent the residues. Figure 6.25 plots the PSNR and the compression ratio against the number of bits used to encode the residues, K and number of bits used to encode residue differences, k, for one dataset. It can be seen that the PSNR varies slowly with the number of bits, but the compression ratio of bit-plane encoding is good.

We compared our method with the present state of Art, MPEG. The MPEG compression of depth and residues (MPEG-R and MPEG-D in graphs of Figure 6.8) provides decent compression and quality, but the bit-plane encoding scheme provides more size to quality trade-offs to suit any situation.

With real dataset we carried out the same experiments. Doo-Young dataset consists of 8-bit images. The results for compression ratios and quality are as shown in Figure 6.6. The point cloud, is fitted in the same manner as in the MOCAP dataset. Here, we do not have any noise levels since no simulation of noise is required as it being a real dataset. We observed that the trade-offs are much similar to the MOCAP simulated real dataset.

We observed from graphs in Figures 6.8, 6.6, if the remote client asks for a particular range of compression ratios and quality, he has a set of choices among various combinations of K-bits, k-bits and block sizes. This makes the system effective for a remote-server-client teleimmersion environment with user compatible service options.

6.5 Conclusions

We presented results of a proxy-based compression scheme for multiple depth movies of a scene involving dynamic human action. The scheme provides good compression ratios at acceptable quality levels. The proxy-based scheme provides several controls like number of bits, block sizes, etc. for the client to control the amount of data to be sent. We have shown results on three synthetic datasets, “Indiandance”, “Exercise” and “Ballet”. These are real world Motion Capture

(MOCAP) sequences applied on graphical synthetic models. The proxies of these MOCAP datasets are simulated to behave as real sequences by inducing bone noise and depth noise for all the frames.

Table 6.2 shows the results on three synthetic datasets with bone noise of 3 degrees and table 6.3 shows the same results with bone angle noise of 5 degrees. Both the tables show compression ration and PSNR (CR/PSNR) values for varying k-bits and K-bits. The plots show the stats for compression ratio and PSNR with varying k, K bits for encoding.

We have noticed that increasing the bone noise factor, as the difference between actual depth and the proxy depth increase, more is the residue generated. Hence maximum bit for full representation increases and thus more bits are required to handle the 3D scene with quality at the client side. As shown in graphs of Figure 6.8, for the same block size, decreasing the K-bits increases the compression ratio but reduces the quality of the scene. Compression ratio increases as the number of bits sent to client decreases, and quality (PSNR) of the scene decreases as the number of bits to represent the scene reduces. Since, compression ratio is inversely proportional to quality of the scene achieved at the client side, there exists a trade-off for optimal bits to be sent through the network.

Also, from the same Figure 6.8, we notice that increasing the block size increase the compression factor as the number of data sent as difference residue increases. Since, difference residue are represented as k-bits where $k < K$ bits, the data to be sent reduces. Thus, at the receiver end, client can decide on the number of bits and block sizes on the deciding factors like bandwidth available and the quality required.

6.6 Plots for the results

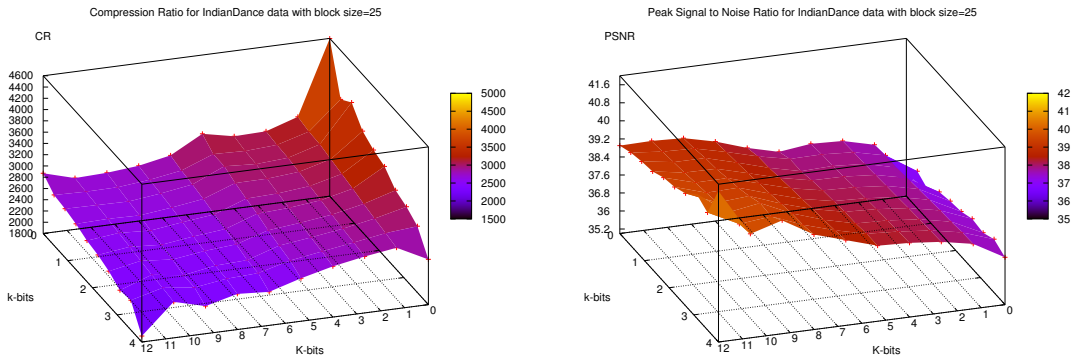

Figure 6.7: Results for IndianDance Dataset with block size=25, bone noise=3, and K=0,5-14, k=0-4

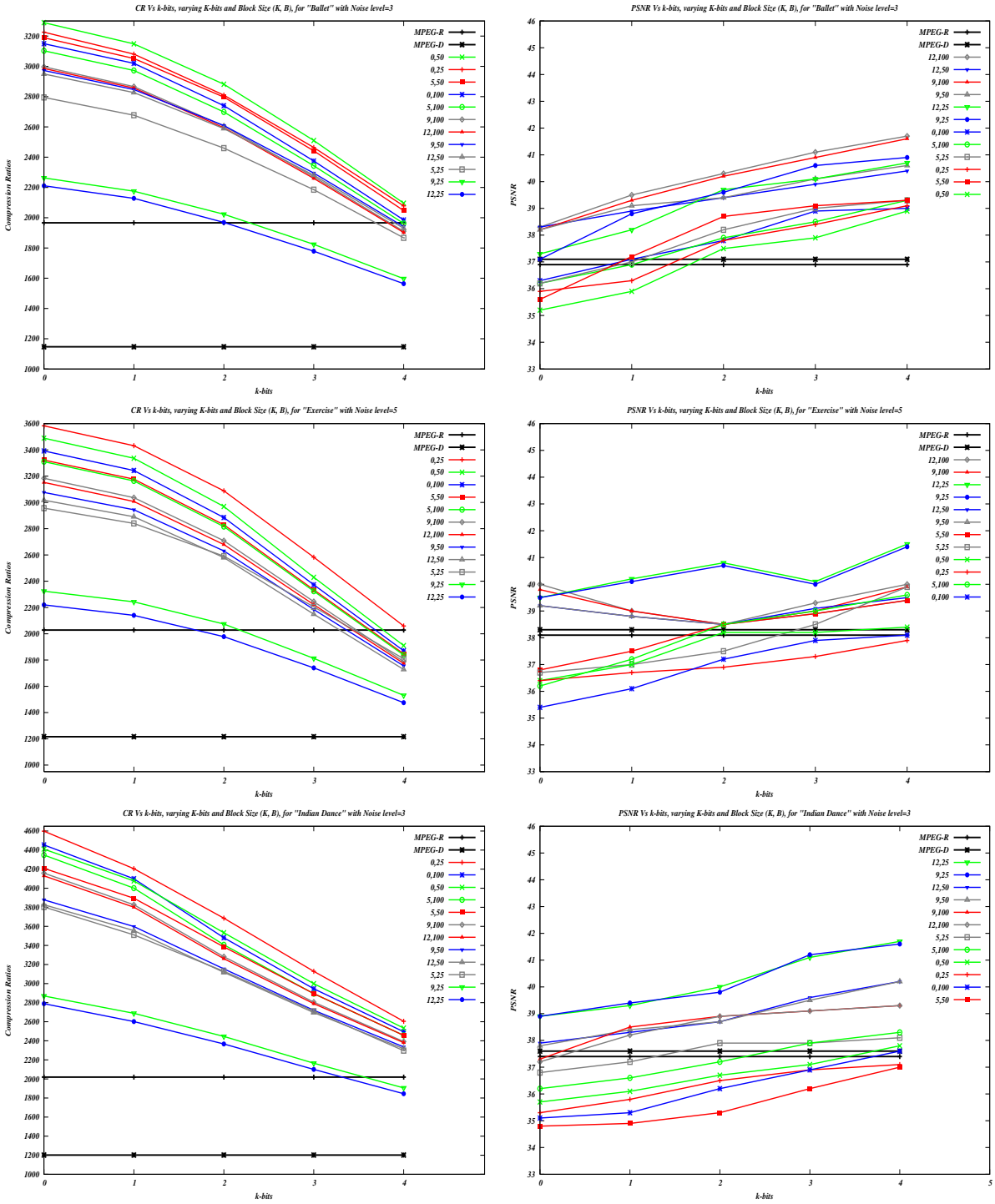

Figure 6.8: Compression Ratio and PSNR results for Ballet, Exercise and IndianDance Dataset with block size=25/50/100 and varying joint angle noise levels, K=0,5,9,12, plotted against k

#bits		IndianDance			Exercise			Ballet		
K	k	$B=25$	$B=50$	$B=100$	$B=25$	$B=50$	$B=100$	$B=25$	$B=50$	$B=100$
0	0	4598/35.3	4410/35.7	4454/35.1	3295/36.2	3214/36.2	3141/35.3	3226/35.9	3289/35.2	3150/36.3
	1	4205/35.8	4077/36.1	4099/35.3	3173/36.5	3081/36.6	3018/36.7	3081/36.3	3149/35.9	3020/37.1
	2	3686/36.5	3533/36.7	3481/36.2	2919/37.1	2806/38.0	2750/37.2	2810/37.8	2882/37.5	2740/37.8
	3	3128/36.9	2999/37.1	2946/36.9	2583/37.8	2433/38.4	2390/37.4	2464/38.4	2510/37.9	2375/38.9
	4	2604/37.1	2533/37.8	2495/37.6	2227/38.0	2043/39.2	1996/38.8	2078/39.1	2096/38.9	1985/39.0
5	0	3803/36.8	4208/34.8	4348/36.2	2895/36.9	3098/36.2	3084/36.4	2795/36.2	3189/35.6	3103/36.2
	1	3511/37.2	3895/34.9	4001/36.6	2789/36.8	2969/37.0	2962/37.1	2677/37.0	3052/37.2	2973/36.9
	2	3129/37.9	3385/35.3	3407/37.2	2583/37.7	2708/38.4	2700/38.1	2460/38.2	2797/38.7	2699/37.9
	3	2710/37.9	2895/36.2	2894/37.9	2312/38.0	2356/38.7	2350/38.9	2185/39.0	2442/39.1	2343/38.5
	4	2297/38.1	2457/37.0	2457/38.3	2011/38.2	1986/38.9	1967/39.2	1867/39.3	2047/39.3	1962/39.3
7	0	2991/37.5	3923/36.3	4189/36.4	2401/37.1	2928/38.3	2998/36.5	2344/36.9	3029/36.3	3025/36.7
	1	2796/37.9	3644/37.0	3859/37.2	2323/38.6	2809/38.5	2880/37.0	2258/38.0	2902/36.8	2900/37.0
	2	2533/38.2	3183/38.2	3300/38.7	2165/39.5	2562/38.9	2624/38.2	2089/39.0	2658/37.2	2629/37.6
	3	2235/38.5	2747/38.5	2817/38.9	1956/40.2	2243/39.4	2288/39.5	1873/39.4	2334/38.6	2288/38.8
	4	1948/39.0	2351/39.1	2401/39.0	1741/41.1	1903/39.9	1923/39.9	1639/39.7	1970/39.7	1923/39.9
9	0	2789/38.9	3827/37.8	4130/37.3	2231/36.9	2849/37.8	2956/38.2	2211/37.1	2950/38.2	2986/38.2
	1	2602/39.4	3556/38.4	3801/38.5	2156/37.9	2735/38.6	2835/39.1	2128/38.8	2827/39.1	2857/39.3
	2	2367/39.8	3118/38.7	3261/38.9	2013/39.9	2494/39.5	2585/39.8	1969/39.6	2589/39.4	2594/40.2
	3	2100/41.2	2696/39.5	2789/39.1	1833/40.6	2187/39.6	2257/40.6	1779/40.6	2281/40.1	2260/40.9
	4	1845/41.6	2317/40.2	2381/39.3	1640/41.9	1862/39.9	1901/40.8	1564/40.9	1930/40.6	1902/41.6
12	0	2870/38.9	3878/37.9	4158/37.2	2292/36.9	2887/37.9	2976/38.3	2263/37.3	2973/38.3	2997/38.3
	1	2688/39.3	3598/38.3	3827/38.2	2216/37.9	2769/38.5	2852/38.4	2175/38.2	2847/38.9	2866/39.5
	2	2446/40.0	3154/38.7	3282/38.9	2075/40.0	2526/39.5	2604/39.8	2023/39.7	2609/39.4	2606/40.3
	3	2165/41.1	2720/39.6	2805/39.1	1885/40.9	2210/39.8	2273/40.2	1824/40.1	2295/39.9	2270/41.1
	4	1906/41.7	2336/40.2	2392/39.3	1681/41.9	1882/40.0	1912/40.9	1597/40.7	1941/40.4	1910/41.7
M-D		2019/37.4	2177/37.2	2184/37.2	2038/36.1	2238/37.2	2287/36.6	1996/36.3	2166/35.7	2350/37.8
M-R		1110/37.6	1165/35.6	1202/36.1	1009/38.9	1093/38.8	1174/37.4	1147/37.1	1218/36.9	1275/36.1

Table 6.2: Compression ratios and PSNR for 3 different datasets with varying bit-wise compression of key frame residues R , residue differences D,varying block sizes=25/50/100, bone noise=3, MPEG encoding of the residues (M-R) and MPEG encoding of the input depth maps (M-D). The first number is the compression ratio and the second the PSNR.

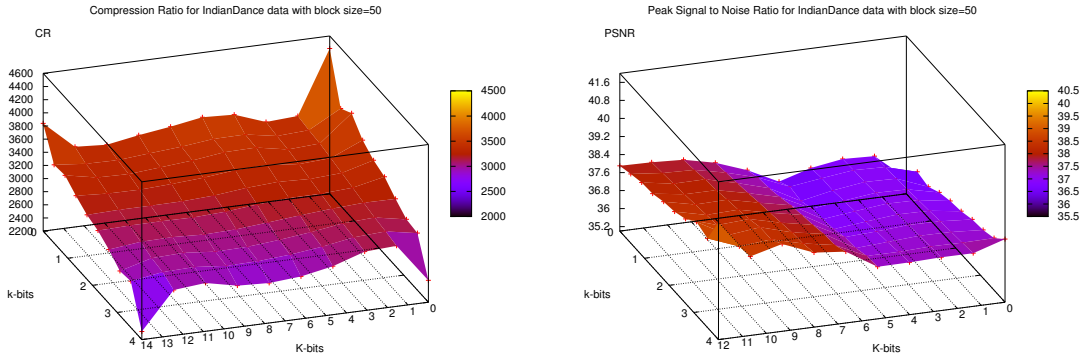

Figure 6.9: Results for IndianDance Dataset with block size=50, bone noise=3, and K=0,5-14, k=0-4

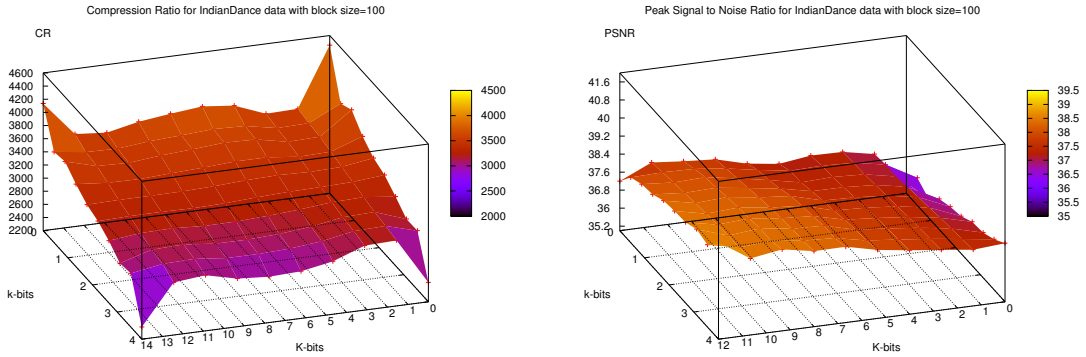

Figure 6.10: Results for IndianDance Dataset with block size=100, bone noise=3, and K=0,5-14, k=0-4

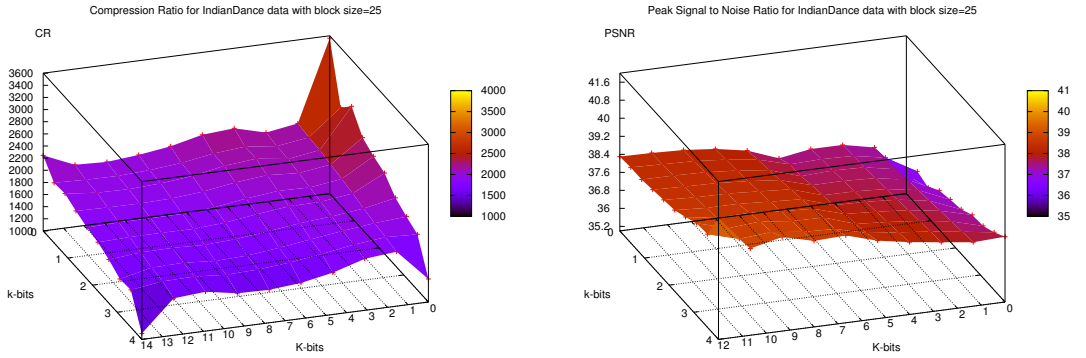

Figure 6.11: Results for IndianDance Dataset with block size=25, bone noise=5, and K=0,5-14, k=0-4

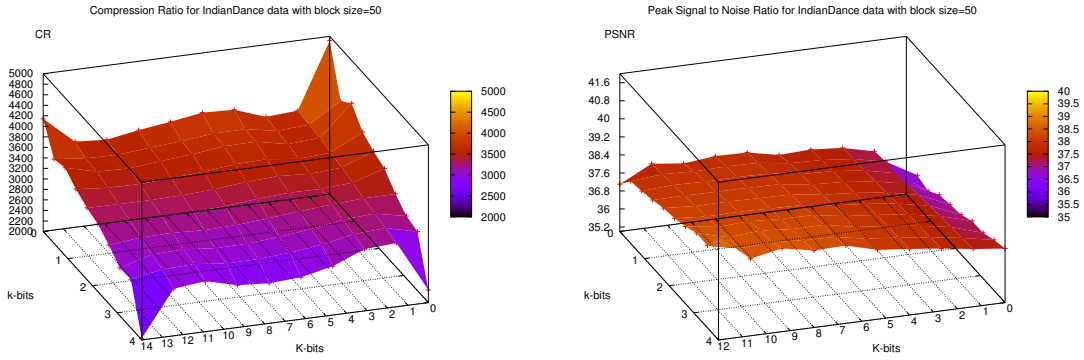

Figure 6.12: Results for IndianDance Dataset with block size=50, bone noise=5, and K=0,5-14, k=0-4

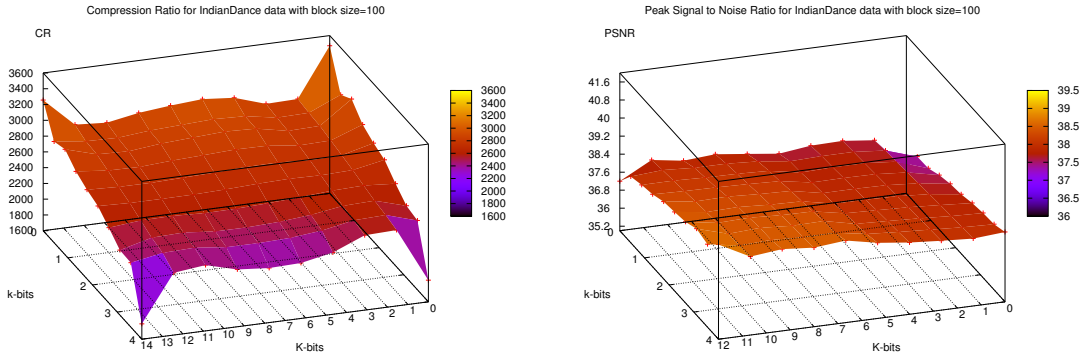

Figure 6.13: Results for IndianDance Dataset with block size=100, bone noise=5, and K=0,5-14, k=0-4

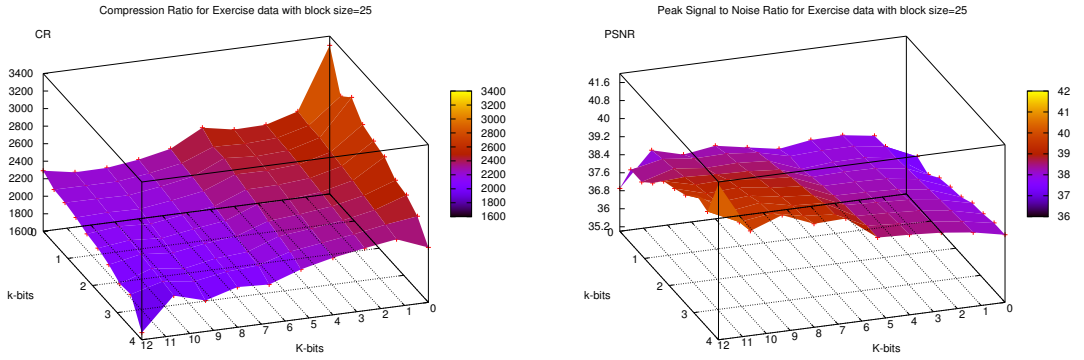

Figure 6.14: Results for Exercise Dataset with block size=25, bone noise=3, and K=0,5-14, k=0-4

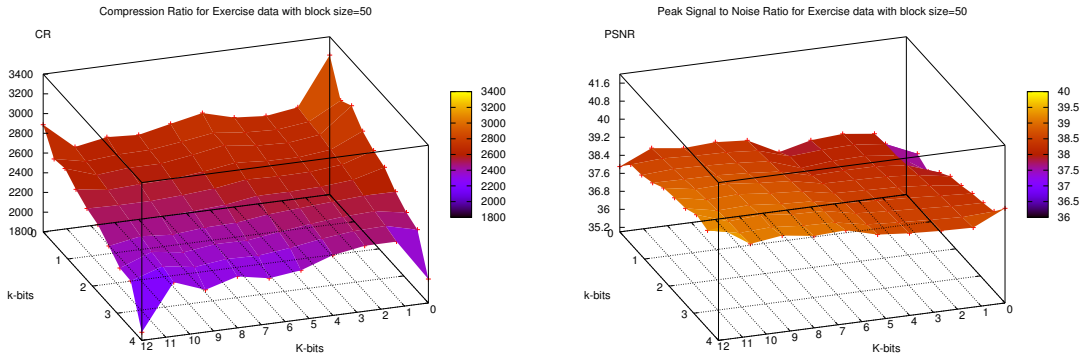

Figure 6.15: Results for Exercise Dataset with block size=50, bone noise=3, and K=0,5-14, k=0-4

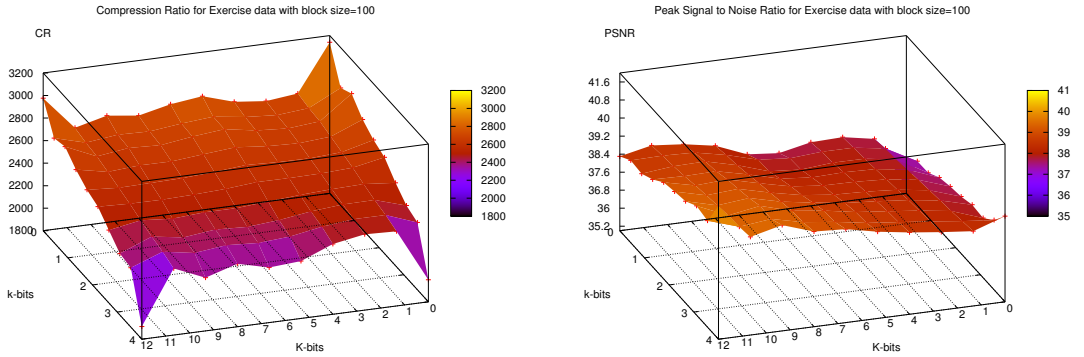

Figure 6.16: Results for Exercise Dataset with block size=100, bone noise=3, and K=0,5-14, k=0-4

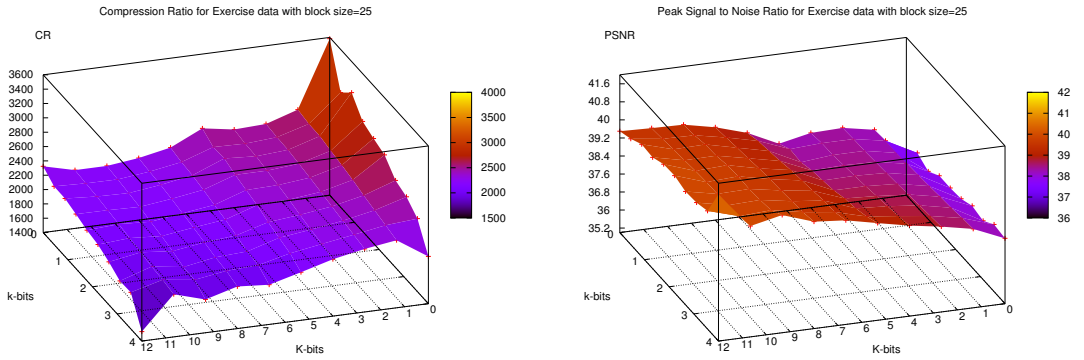

Figure 6.17: Results for Exercise Dataset with block size=25, bone noise=5, and K=0,5-14, k=0-4

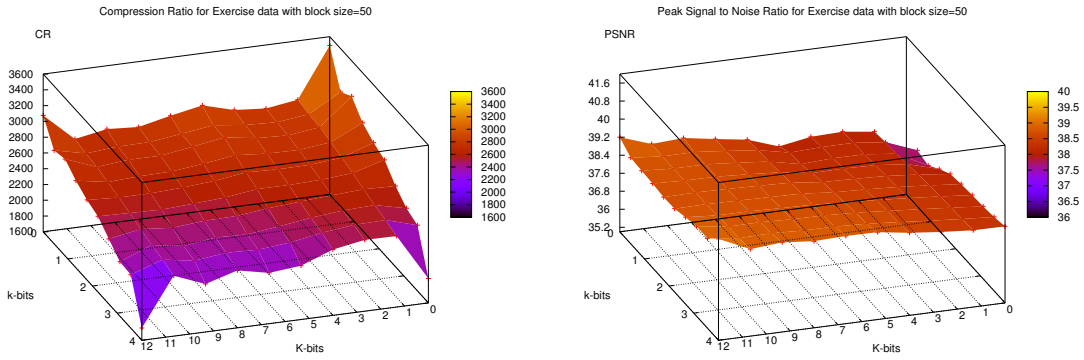

Figure 6.18: Results for Exercise Dataset with block size=50, bone noise=5, and K=0,5-14, k=0-4

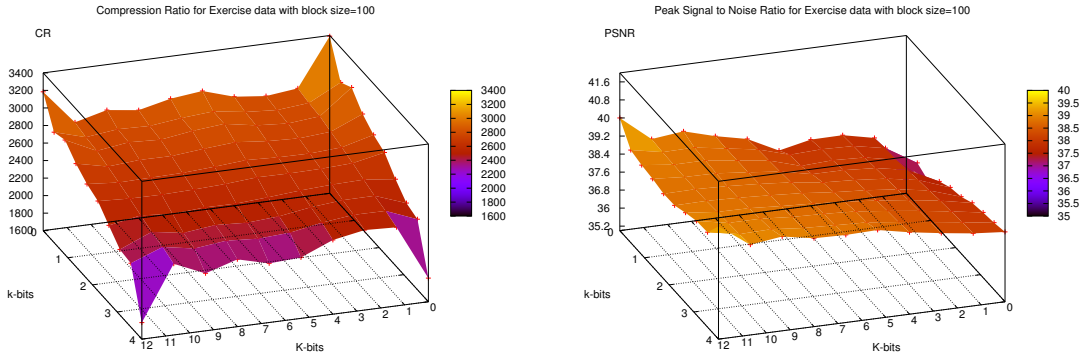

Figure 6.19: Results for Exercise Dataset with block size=100, bone noise=5, and K=0,5-14, k=0-4

#bits		IndianDance			Exercise			Ballet		
K	k	$B=25$	$B=50$	$B=100$	$B=25$	$B=50$	$B=100$	$B=25$	$B=50$	$B=100$
0	0	3561/35.1	4919/35.0	3472/36.1	3583/36.4	3489/36.4	3392/35.4	3463/36.1	3560/35.2	3438/36.0
	1	3280/35.9	4607/35.8	3321/37.0	3432/36.7	3336/37.0	3243/36.1	3283/36.8	3386/36.2	3268/36.8
	2	2690/36.5	3959/36.3	2954/37.5	3088/36.9	2969/38.2	2885/37.2	2959/37.7	3035/37.1	2876/37.1
	3	1981/37.2	3058/36.9	2416/37.9	2583/37.3	2430/38.2	2373/37.9	2468/38.1	2486/38.0	2332/37.9
	4	1381/37.9	2241/37.4	1875/38.1	2059/37.9	1910/38.4	1870/38.1	1962/38.3	1952/38.6	1817/38.9
5	0	2870/36.2	4589/36.9	3385/36.8	2956/36.7	3322/36.8	3310/36.2	2870/36.3	3407/36.5	3365/36.0
	1	2671/36.8	4302/37.5	3237/37.3	2840/37.0	3177/37.5	3164/37.2	2735/37.1	3241/37.2	3197/36.9
	2	2241/37.2	3719/37.9	2882/37.5	2591/37.5	2830/38.5	2816/38.5	2496/37.9	2911/38.3	2813/37.4
	3	1720/37.9	2908/38.1	2366/38.0	2205/38.5	2335/38.9	2324/39.0	2119/38.0	2398/38.9	2291/38.0
	4	1243/38.9	2154/38.1	1842/38.7	1810/39.9	1847/39.4	1838/39.6	1733/38.8	1893/39.4	1790/38.8
7	0	2338/37.5	4231/37.7	3283/37.1	2426/38.2	3124/38.2	3209/38.2	2359/36.8	3195/37.3	3257/36.4
	1	2198/38.0	3982/38.0	3141/38.1	2344/39.0	2991/38.4	3069/38.5	2264/37.5	3044/38.0	3097/36.9
	2	1883/38.7	3453/38.3	2796/38.5	2154/39.7	2669/38.8	2729/38.6	2079/38.2	2735/38.7	2726/37.9
	3	1487/39.3	2736/38.9	2304/39.0	1873/40.0	2215/38.9	2261/39.0	1803/38.7	2272/38.9	2226/38.7
	4	1117/39.8	2057/39.1	1803/39.4	1575/40.2	1772/39.3	1798/39.2	1512/39.1	1814/39.0	1751/39.9
9	0	2223/38.2	4122/37.1	3250/37.1	2220/39.5	3017/39.2	3152/39.8	2229/36.4	3104/36.7	3209/37.2
	1	2088/38.3	3882/37.8	3108/38.0	2140/40.1	2891/38.8	3007/39.0	2128/37.0	2960/38.0	3045/38.3
	2	1809/38.4	3372/38.2	2767/38.7	1978/40.7	2580/38.5	2679/38.5	1964/38.7	2660/39.3	2686/39.8
	3	1436/39.3	2678/38.9	2281/38.9	1740/40.0	2149/38.9	2222/39.0	1719/39.2	2215/40.0	2197/40.1
	4	1087/40.3	2026/39.6	1789/39.1	1475/41.4	1728/39.4	1774/39.9	1449/39.9	1778/40.6	1733/40.5
12	0	2317/38.3	4186/37.1	3269/37.2	2324/39.5	3076/39.2	3184/40.0	2308/36.4	3154/36.8	3236/37.3
	1	2172/38.3	3937/37.8	3121/38.1	2243/40.2	2944/38.8	3036/39.0	2209/37.1	3004/38.1	3069/38.4
	2	1873/38.5	3418/38.3	2785/38.7	2074/40.8	2631/38.5	2708/38.5	2043/38.7	2700/39.4	2710/39.9
	3	1486/39.3	2704/38.9	2294/39.1	1813/40.1	2185/39.1	2245/39.3	1778/39.2	2244/39.0	2214/39.4
	4	1115/40.4	2047/39.6	1800/39.2	1531/41.5	1755/39.5	1788/40.0	1495/40.0	1800/40.6	1744/40.6
M-D		2011/37.2	2148/37.1	2148/37.1	2028/36.7	2229/38.1	2335/36.7	1936/36.3	2149/35.1	2339/37.7
M-R		1007/36.1	1103/36.6	1178/36.8	1097/39.1	1148/38.0	1216/37.1	1043/37.9	1129/36.3	1214/36.1

Table 6.3: Compression ratios and PSNR for 3 different datasets with varying bit-wise compression of key frame residues R , residue differences D,varying block sizes=25/50/100, bone noise=5, MPEG encoding of the residues (M-R) and MPEG encoding of the input depth maps (M-D). The first number is the compression ratio and the second the PSNR.

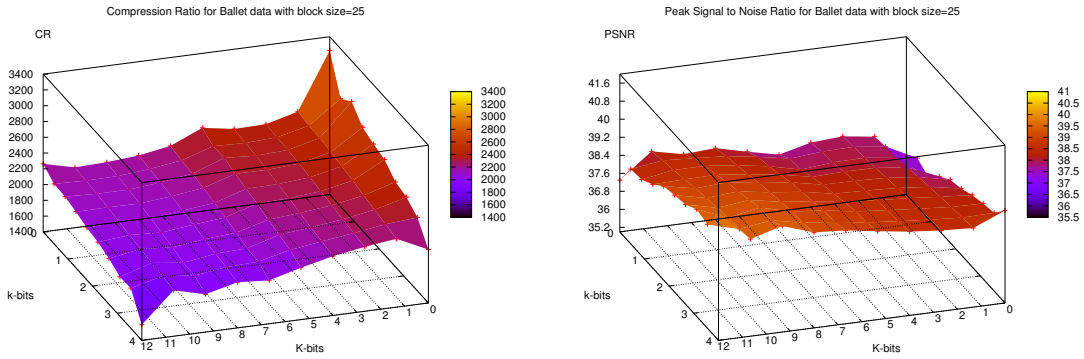

Figure 6.20: Results for Ballet Dataset with block size=25, bone noise=3, and K=0,5-14, k=0-4

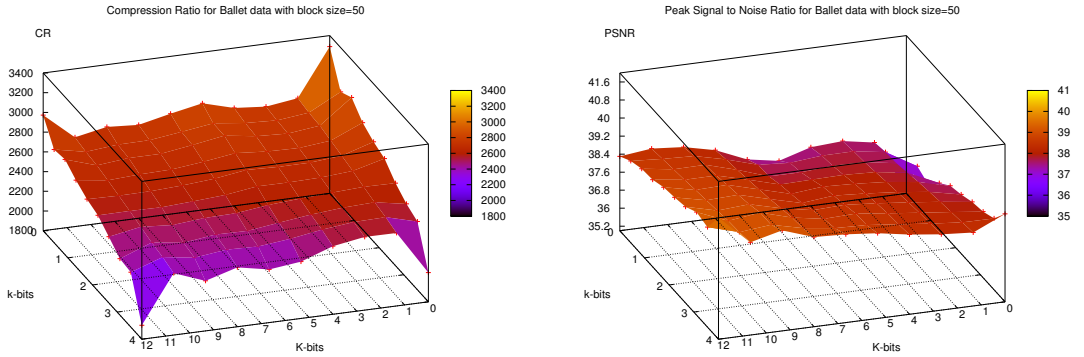

Figure 6.21: Results for Ballet Dataset with block size=50, bone noise=3, and K=0,5-14, k=0-4

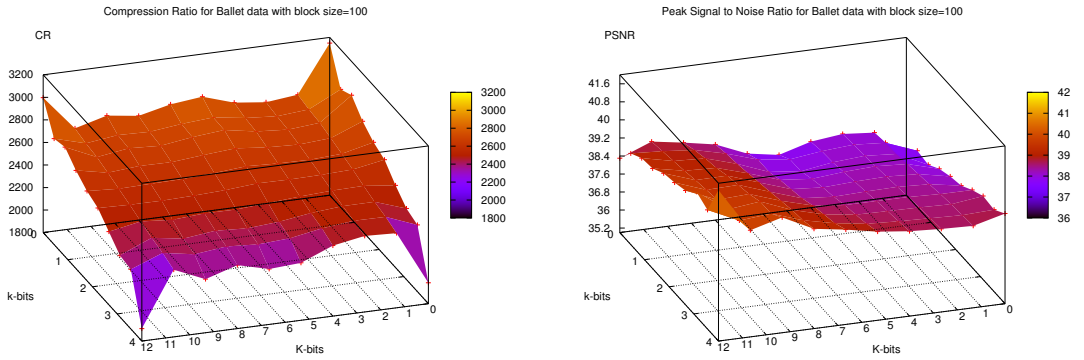

Figure 6.22: Results for Ballet Dataset with block size=100, bone noise=3, and K=0,5-14, k=0-4

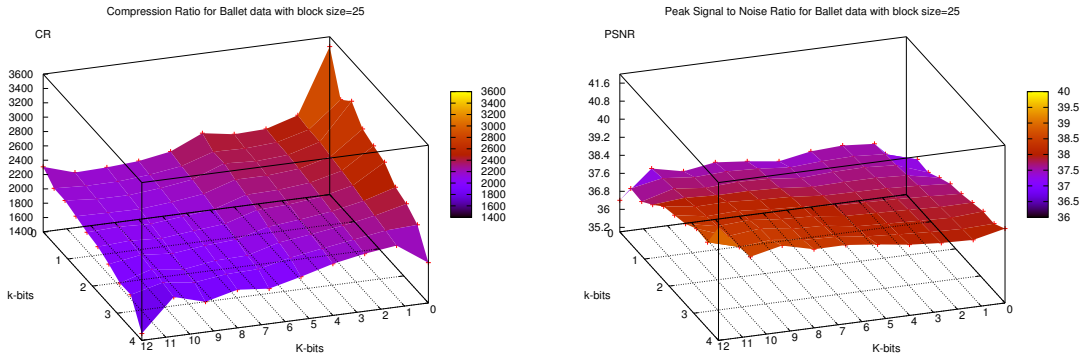

Figure 6.23: Results for Ballet Dataset with block size=25, bone noise=5, and K=0,5-14, k=0-4

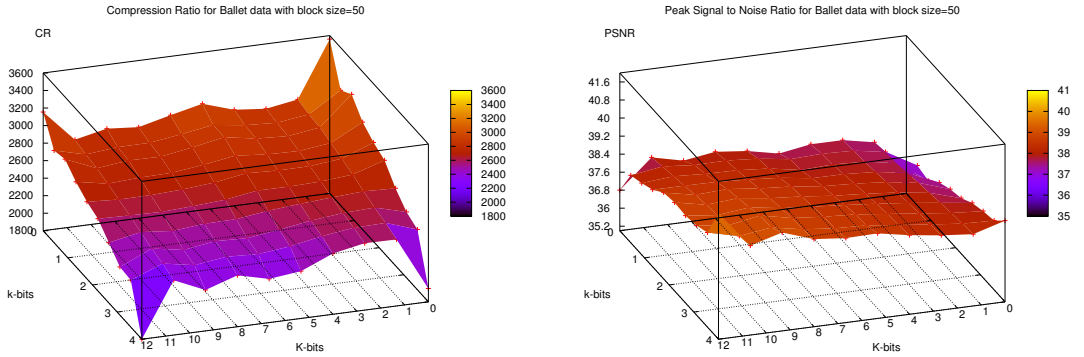

Figure 6.24: Results for Ballet Dataset with block size=50, bone noise=5, and K=0,5-14, k=0-4

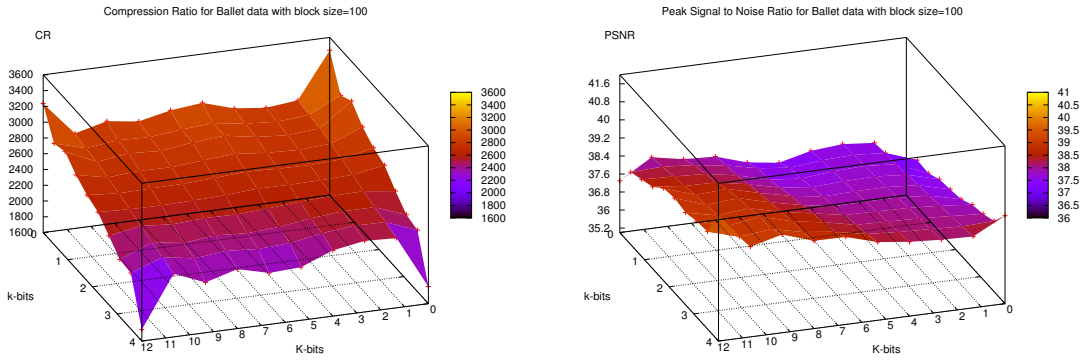

Figure 6.25: Results for Ballet Dataset with block size=50, bone noise=5, and K=0,5-14, k=0-4

Chapter 7

Conclusions and Future Work

In this thesis, we presented various algorithms for compression of time-varying multiview depth movies. Time varying depth and image sequences, called Depth Movies, can extend Image based rendering and modelling (IBMR) to dynamic events. Depth movies capture time varying geometry of a dynamic scene and are often streamed to a distant location for immersive viewing. The applications of such systems include virtual-space tele-conferencing, remote 3D immersion, 3D entertainment, etc. Immersive display applications and transmission of depth images for each frame requires time-varying sequences of depth images from multiple cameras in order to get the $2\frac{1}{2}$D information of the scene. Multiview image compression and video compression have been well studied earlier, but there has been no study about dynamic depth map compression. The dynamic depth map data is heavy and need efficient compression schemes. Our thesis contributes towards developing dynamic depth map compression algorithms for efficient transmission in a server-client 3D teleimmersive environment.

Our work explored the compression of depth movies of human actors using a parametric proxy model for the underlying action. We use a generic articulated human model as the proxy to represent the common human model in action and the various joint angles of the model to parametrize the proxy for each time instant. The proxy represents a common prediction of the scene structure. The difference or residue between the captured depth and the depth of the proxy represents the scene prediction error, which is encoded to exploit spatial and temporal coherence. We experimented with bit-wise compression of the residues and analyzed the quality of the generated 3D scene. Differences in residues across time, difference-coded frames, are used to exploit temporal coherence. Also, intra-frame coded frames and difference-coded frames provide random access and high compression. We presented results on several synthetic and real actions to demonstrate the compression ratio and resulting quality using a depth-based rendering of the decoded scene. In summary, we simulated a 3D tele-immersive or teleconferencing system using depth Movies, representing them with a simple proxy, compressing it using residues and reconstructed it at the client side. Our compression scheme resulted in a compression ratio of 4919/3058/2735/1800 at PSNR levels of 35.0/36.9/38.7/40.6. Our scheme also provides trade-off in bitrate and quality.

Today, 3D tele-presence and teleimmersive environments are designed to overcome disadvantages of desktop videoconferencing, that gives and artificial presence of the remote location, and to establish a life-like conference sessions that bring people at the remote locations together as if face-to-face. In most basic terms, it transmits a life-size 3D image of an environment to the remote client, creating the perception of the client's presence in the room by transmitting and reconstructing both the environment and the voice at the remote site. Our system represents the dynamic 3D presence as multiview depth movies, captures them, efficiently compresses them taking care of compression

ratios and qualities for a real-time transmission and then renders them back at the remote site with real-time depth image based rendering algorithms. Thus, the most basic application of our system is 3D teleimmersion.

3D Tele-immersion systems are these days being used remote educational institutions, remote surgical operational training, corporate remote meetings, research labs, etc. Our application can eventually be effectively used in the same manner.

We started with an idea of representing any generic dynamic scene with rigid/ non-rigid, dynamic/ static object(s) using a common proxy model(s). In our thesis, we have concentrated on the most difficult thing to represent in a scene, i.e., a human being, but as future work, we would like to increase the scope of the work to animals like dogs, cows, etc. The work differs only at generating a different hierarchical proxy model. Also as future work, we can include different number and different kinds of objects in a scene. Thus, the idea that we have presented, can easily be taken as the further work to enhance the scope, by including animals, and similar non-rigid objects to capture the essence of a true tele-conferencing system. Extending it to non-human generic scenes.

Appendix 1

The work done during my masters has been disseminated to the following conferences:

1. Pooja Verlani, Aditi Goswami, P. J. Narayanan, Shekhar Dwivedi, Sashi Kumar Penta **Depth Images: Representations and Real-Time Rendering**. *Proceedings of Third International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT 2006), 14-16 June 2006, Chapel Hill, North Carolina, USA, 962-969.*

2. Pooja Verlani, P. J. Narayanan **Parametric Proxy-Based Compression of Multiple Depth Movies of Humans**. *Proceedings of Data Compression Conference 2008, 25-27 March 2008, Snowbird, UT, USA, P:550.*

3. Pooja Verlani, P. J. Narayanan **Proxy-Based Compression of $2\frac{1}{2}$D Structure of Dynamic Events for 3D teleconferencing**. *Proceedings of Fourth International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT 2008), 18-20 June 2008, Georgia Institute of Technology, Atlanta, GA, USA.*

Appendix 2

We would like to thank Michael Waschbusch from ETH-Z for providing the Doo-Young sequence for real depth movies as shown in 7.1 The data was captured from 15 cameras for 250 frames as shown in Figure 7.2.

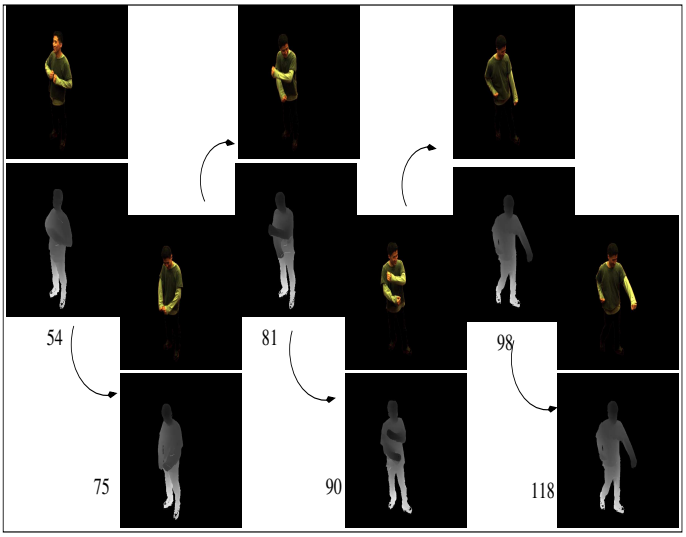

Figure 7.1: Doo-Young Dataset Depth Movie from ETH-Z.

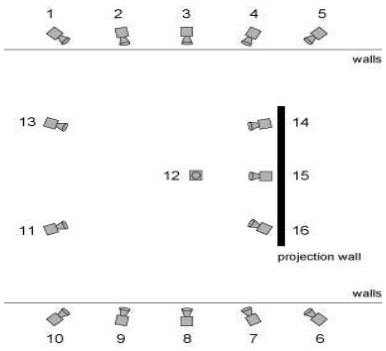

Figure 7.2: Camera Setup for Doo-Young Dataset.

For further details please visit: *http://graphics.ethz.ch/research/3dvideo/main.php?Menu=7&Submenu=0*

Bibliography

[1] Specification of a standard vrml humanoid.[online], 2007.

[2] A. Agarwal and B. Triggs. Recovering 3d human pose from monocular images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(1):44–58, 2006.

[3] A. Aubel, R. Boulic, and D. Thalmann. Real-time display of virtual humans: levels of details and impostors. *IEEE Trans. Circuits Syst. Video Techn.*, 10(2):207–217, 2000.

[4] H. L. B. Wilburn, M. Smulski and M. Horowitz. The light field video camera. *Media Processors 2002*, 4674:29–36, 2002.

[5] N. I. Badler, B. A. Barsky, and D. Zeltzer, editors. *Making them move: mechanics, control, and animation of articulated figures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[6] N. I. Badler, K. H. Manoocherhri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Comput. Graph. Appl.*, 7(6):28–38, 1987.

[7] N. I. Badler and S. W. Smoliar. Digital representations of human movement. *ACM Comput. Surv.*, 11(1):19–38, 1979.

[8] H. H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender. Understanding performance in coliseum, an immersive videoconferencing system. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(2):190–210, 2005.

[9] P. Baker and Y. Aloimonos. Complete calibration of a multi-camera network. In *OMNIVIS '00: Proceedings of the IEEE Workshop on Omnidirectional Vision*, page 134, Washington, DC, USA, 2000. IEEE Computer Society.

[10] M. Billinghurst, A. D. Cheok, S. Prince, and H. Kato. Ieee computer graphics & applications: Projects in vr - real world teleconferencing. *IEEE Distributed Systems Online*, 4(2), 2003.

[11] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, 12(3):286–292, 1978.

[12] J. Bloomenthal. Hand crafted: Modeling, visualization and animating implicit surfaces. *ACM SIGGRAPH Course Notes*, 25, 1993.

[13] M. Bourges-Sévenier, E. S. Jang, G. Lafruit, and F. Morán. Introduction to the special issue on mpeg-4's animation framework extension (afx). *IEEE Trans. Circuits Syst. Video Techn.*, 14(7):926–927, 2004.

[14] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, New York, NY, USA, 2001. ACM.

[15] J. Carranza, C. Theobalt, M. A. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 22(3):569–577, 2003.

[16] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 243–252, New York, NY, USA, 1989. ACM.

[17] C. Chang, X. Zhu, P. Ramanathan, and B. Girod. Light field compression using disparity-compensated lifting and shape adaptation. *Image Processing*, 15(4):793–806, April 2006.

[18] C.-F. Chang, G. Bishop, and A. Lastra. Ldi tree: a hierarchical representation for image-based rendering. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 291–298, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[19] S. E. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series):279–288, 1993.

[20] G. K. M. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *CVPR*, pages 2714–2720, 2000.

[21] Q. Delamarre and O. Faugeras. 3d articulated models and multiview tracking with physical forces. *Comput. Vis. Image Underst.*, 81(3):328–357, 2001.

[22] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94, New York, NY, USA, 1998. ACM.

[23] C. Fehn, E. Cooke, O. Schreer, and P. Kauff. 3d analysis and image-based rendering for immersive tv applications. *SPIC, SIGNAL PROCESSING: IMAGE COMMUNICATION*, 17(9):705–715, October 2002.

[24] C. Fehn, P. Kau, M. de Beeck, F. Ernst, W. Ijsselsteijn, M. Pollefeys, E. Ofek, and L. V. Gool. An evolutionary and optimised approach on 3d-tv, 2002.

[25] D. M. Gavrila. The visual analysis of human movement: a survey. *Comput. Vis. Image Underst.*, 73(1):82–98, 1999.

[26] D. M. Gavrila and L. S. Davis. 3-d model-based tracking of humans in action: a multi-view approach. In *CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, page 73, Washington, DC, USA, 1996. IEEE Computer Society.

[27] M. Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *SIGGRAPH Comput. Graph.*, 19(3):263–270, 1985.

[28] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM.

[29] D. Gotz, K. Mayer-Patel, and D. Manocha. Irw: an incremental representation for image-based walkthroughs. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 67–76, New York, NY, USA, 2002. ACM.

[30] N. Greene. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, 1986.

[31] M. H. Gross, S. Würmlin, M. Näf, E. Lamboray, C. P. Spagno, A. M. Kunz, E. Koller-Meier, T. Svoboda, L. J. V. Gool, S. Lang, K. Strehlke, A. V. Moere, and O. G. Staadt. blue-c: A spatially immersive display and 3d video portal for telepresence. *ACM Trans. Graph.*, 22(3):819–827, 2003.

[32] J. P. Grossman and W. J. Dally. Point sample rendering. In *Proceedings of the 9th Eurographics Workshop on Rendering Techniques*, pages 181–192, (Vienna, Austria), 1998. Springer.

[33] P. Hanrahan and D. J. Sturman. Interactive animation of parametric models. *The Visual Computer*, 1(4):260–266, 1985.

[34] G. M. I. Cohen and H. Gu. Inference of 3-d human body posture from multiple cameras for vision-based user interface. *World Multiconf. on Syst., Cybern. Informatics*, 2001.

[35] I. Ihm, S. Park, and R. K. Lee. Rendering of spherical light fields. In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, page 59, Washington, DC, USA, 1997. IEEE Computer Society.

[36] A. Jagmohan, A. Sehgal, and N. Ahuja. Compression of lightfield rendering data using coset codes. *Asilomar Conference on Signal and communications*, 2003.

[37] B. Javidi and F. Okano. *Three-Dimensional Television, Video and Display Technology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[38] T. Kanade and P. Narayanan. Virtualized reality: Perspectives on 4d digitization of dynamic events. *Proceedings of IEEE Computer Graphics and Applications*, 27(3):32–40, 2007.

[39] T. Kanade, P. Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE MultiMedia*, 4(1):34–47, 1997.

[40] P. Kauff and O. Schreer. An immersive 3d video-conferencing system using shared virtual team user environments. In *CVE '02: Proceedings of the 4th international conference on Collaborative virtual environments*, pages 105–112, New York, NY, USA, 2002. ACM.

[41] K. Komatsu. Human skin model capable of natural shape variation. *The Visual Computer*, 3(5):265–271, 1988.

[42] R. Krishnamurthy, B. Chai, H. Tao, and S. Sethuraman. Compression and transmission of depth maps for image-based rendering. In *ICIV01*, pages III: 828–831, 2001.

[43] S. Kum and K. MayerPatel. Intra-stream encoding for multiple depth streams. In *Proceedings of International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 62–67, May 2006.

[44] S.-U. Kum and K. Mayer-Patel. Real-time multidepth stream compression. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(2):128–150, 2005.

[45] E. Lamboray, S. Würmlin, and M. Gross. Real-time streaming of point-based 3d video. In *VR '04: Proceedings of the IEEE Virtual Reality 2004 (VR'04)*, page 91, Washington, DC, USA, 2004. IEEE Computer Society.

[46] J. Lengyel. The convergence of graphics and vision. *Computer*, 31(7):46–53, 1998.

[47] L. Levkovich-Maslyuk, A. Ignatenko, A. Zhirkov, A. Konushin, I. K. Park, M. Han, and Y. Bayakovski. Depth image-based representation and compression for static and animated 3-d objects. *IEEE Trans. Circuits Syst. Video Techn.*, 14(7):1032–1045, July 2004.

[48] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM.

[49] N. Magnenat-Thalmann and D. Thalmann. *Synthetic actors in computer-generated 3D films*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[50] M. Magnor, P. Eisert, and B. Girod. Multi-view image coding with depth maps and 3-d geometry for prediction. *Proc. SPIE Visual Communication and Image Processing (VCIP-2001),* San Jose, USA, pages 263–271, jan 2001.

[51] M. Magnor and B. Girod. Data compression for light field rendering. *IEEE Trans. Circuits and Systems for Video Technology*, 10(3):338–343, 2000.

[52] M. A. Magnor, P. Ramanathan, and B. Girod. Multi-view coding for image-based rendering using 3-d scene geometry. *IEEE Trans. Circuits Syst. Video Techn.*, 13(11):1092–1106, 2003.

[53] W. R. Mark. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. PhD thesis, University of North Carolina, 1999.

[54] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Symposium on Interactive 3D Graphics*, pages 7–16, 180, 1997.

[55] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

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

[57] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina, Apr 1997.

[58] L. McMillan and G. Bishop. Plenoptic modeling: an image-based rendering system. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46, New York, NY, USA, 1995. ACM.

[59] A. Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[60] G. MILLER. Volumetric hyper-reality, a computer graphics holy grail for the 21st century? *In Proceedings of Graphics Interface '95*, pages 56–64, 1995.

[61] J. Mulligan, X. Zabulis, N. Kelshikar, and K. Daniilidis. Stereo-based environment scanning for immersive telepresence. *CirSysVideo*, 14(3):304–320, March 2004.

[62] T. Naemura, J. Tago, and H. Harashima. Real-time video-based modeling and rendering of 3d scenes. *IEEE Comput. Graph. Appl.*, 22(2):66–73, 2002.

[63] P. J. Narayanan, S. K. Penta, and S. R. K. Depth+texture representation for image based rendering. In *ICVGIP*, pages 113–118, 2004.

[64] P. J. Narayanan, P. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *ICCV*, pages 3–10, 1998.

[65] L. P. Nedel and D. Thalmann. Real time muscle deformations using mass-spring systems. In *CGI '98: Proceedings of the Computer Graphics International 1998*, page 156, Washington, DC, USA, 1998. IEEE Computer Society.

[66] T. Okoshi. Three dimensional displays. *IEEE Proceedings*, 68:548–564, 1976.

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

[68] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[69] M. Preda. Mpeg-4 animation framework extension (afx) vm 9.0. *ISO/IEC JTC1/SC29/WG11*, (Document N5245), 2002.

[70] P. Ramanathan, M. Kalman, and B. Girod. Rate-distortion optimized streaming of compressed light fields. In *ICIP (3)*, pages 277–280, 2003.

[71] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. *Computer Graphics*, 32:179–188, 1998.

[72] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[73] H. Schirmacher, M. Li, and H.-P. Seidel. On-the-fly processing of generalized lumigraphs. *Comput. Graph. Forum*, 20(3), 2001.

[74] G. M. Schuster and A. K. Katsaggelos. An optimal quadtree-based motion estimation and motion-compensated interpolation scheme for video compression. *IEEE Transactions on Image Processing*, 7(11):1505–1523, 1998.

[75] S. M. Seitz and C. R. Dyer. View morphing. *Computer Graphics*, 30(Annual Conference Series):21–30, 1996.

[76] J. W. Shade, S. J. Gortler, L.-W. He, and R. Szeliski. Layered depth images. *Computer Graphics*, 32(Annual Conference Series):231–242, 1998.

[77] H. Shum and S. B. Kang. Review of image-based rendering techniques. In *VCIP*, pages 2–13, 2000.

[78] H.-Y. Shum and L.-W. He. Rendering with concentric mosaics. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 299–306, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[79] H.-Y. Shum, J. Sun, S. Yamazaki, Y. Li, and C.-K. Tang. Pop-up light field: An interactive image-based modeling and rendering system. *ACM Trans. Graph.*, 23(2):143–162, 2004.

[80] M.-C. Silaghi, R. Plänkers, R. Boulic, P. Fua, and D. Thalmann. Local and global skeleton fitting techniques for optical motion capture. In *CAPTECH '98: Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, pages 26–40, London, UK, 1998. Springer-Verlag.

[81] A. Smolic and P. Kauff. Interactive 3-d video representation and coding technologies. In *Proceedings of IEEE*, volume 93, pages 98–110, 2004.

[82] A. Smolic and H. Kimata. Report on 3dav exploration. *ISO/IEC JTC1/SC29/WG11*, (Document N5878), 2003.

[83] S. N. Steketee and N. I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. *SIGGRAPH Comput. Graph.*, 19(3):255–262, 1985.

[84] M. Tanimoto and T. Fuji. Ray-space coding using temporal and spatial predictions. *ISO/IEC JTC1/SC29/WG11*, (Document M10410), 2003.

[85] D. Thalmann, J. Shen, and E. Chauvineau. Fast realistic human body deformations for animation and vr applications. In *CGI '96: Proceedings of the 1996 Conference on Computer Graphics International*, page 166, Washington, DC, USA, 1996. IEEE Computer Society.

[86] C. Theobalt, E. de Aguiar, M. A. Magnor, H. Theisel, and H.-P. Seidel. Marker-free kinematic skeleton estimation from sequences of volume data. In *VRST*, pages 57–64, 2004.

[87] C. Theobalt, M. Magnor, P. Schüler, and H.-P. Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 96, Washington, DC, USA, 2002. IEEE Computer Society.

[88] X. Tong and R. M. Gray. Coding of multi-view images for immersive viewing. In *ICASSP '00: Proceedings of the Acoustics, Speech, and Signal Processing, 2000. on IEEE International Conference*, pages 1879–1882, Washington, DC, USA, 2000. IEEE Computer Society.

[89] H. Towles, S. Kum, T. Sparks, S. Sinha, S. Larsen, and N. Beddes. Transport and rendering challenges of multi-stream,3d tele-immersion data. In *CVRV*, October 2003.

[90] P. Verlani, A. Goswami, P. J. Narayanan, S. Dwivedi, and S. K. Penta. Depth images: Representations and real-time rendering. In *3DPVT*, pages 962–969, 2006.

[91] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz. High-speed videography using a dense camera array. In *CVPR (2)*, pages 294–301, 2004.

[92] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. *ACM Trans. Graph.*, 24(3):765–776, 2005.

[93] A. Wilson, K. Mayer-Patel, and D. Manocha. Spatially-encoded far-field representations for interactive walkthroughs. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 348–357, New York, NY, USA, 2001. ACM.

[94] Q. Wu, K. T. Ng, S.-C. Chan, and H.-Y. Shum. On object-based compression for a class of dynamic image-based representations. In *ICIP (3)*, pages 405–408, 2005.

[95] S. Würmlin, E. Lamboray, O. G. Staadt, and M. H. Gross. 3d video recorder: a system for recording and playing free-viewpoint video. *Comput. Graph. Forum*, 22(2):181–194, 2003.

[96] M. Yamamoto, A. Sato, S. Kawada, T. Kondo, and Y. Osaki. Incremental tracking of human actions from multiple views. In *CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 2, Washington, DC, USA, 1998. IEEE Computer Society.

[97] J. C. Yang, M. Everett, C. Buehler, and L. McMillan. A real-time distributed light field camera. In *Rendering Techniques*, pages 77–86, 2002.

[98] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S.-H. Jung, and R. Bajcsy. Teeve: The next generation architecture for tele-immersive environment. In *ISM*, pages 112–119, 2005.

[99] S.-U. Yoon, E.-K. Lee, S.-Y. Kim, and Y.-S. Ho. A framework for multi-view video coding using layered depth images. In *PCM (1)*, pages 431–442, 2005.

[100] S. Yoshimoto. Ballerinas generated by a personal computer. *Visual Computer Animation*, 3(1):85–90, 1992.

[101] J. Yu, L. McMillan, and S. Gortler. Scan light field rendering. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 137, Washington, DC, USA, 2002. IEEE Computer Society.

[102] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004.