OMNIDIRECTIONAL VIEW AND MULTI-MODAL STREAMING IN 3D TELE-IMMERSION SYSTEM

BY

ZHENHUAN GAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Klara Nahrstedt

ABSTRACT

3D Tele-immersion (3DTI) technology allows full-body, multi-modal content delivery among geographically dispersed users. In 3DTI, user's 3D model will be captured by multiple RGB-D (color plus depth) cameras surrounding user's body. In addition, various sensors (e.g., motion sensors, medical sensors, wearable gaming consoles, etc.) specified by the application will be included to deliver a multi-modal experience.

In a traditional 2D live video streaming system, the interactivity of end users, choosing a specified viewpoint, has been crippled by the fact that they can only choose to see the physical scene captured by a physical camera, but not between two physical cameras. However, 3DTI system makes it possible rendering a 3D space where the viewers can view physical scene from arbitrary viewpoint.

In this thesis, we present systematic solutions of omnidirectional view in 3D tele-immersion system in a real-time manner and in an on-demand streaming manner, called *FreeViewer* and *OmniViewer*, respectively. we provide a complete multi-modal 3D video streaming/rendering solution, which achieves the feature of omnidirectional view in monoscopic 3D systems.

To my mother, for her love and support.

ACKNOWLEDGMENTS

First and foremost, I would like to express my appreciation to my adviser Prof. Klara Nahrstedt, without whose constant guidance and support, this work would not have been possible. She is an extremely motivating and helpful mentor in multimedia research. Under her guidance, I have the opportunity to be introduced to the multimedia research and successfully finish the thesis.

I am also grateful for the support from all the MONET group members. I really enjoyed all the conversations with them in the past three years, including Shannon Chen, Haiming Jin, Hongyang Li, Siting Chang, Wenyu Ren, Rauol Rivas, Ahsan Arefin, Rehana Tabassum, Scott Huang, and Phuong V. Nguyen.

Above all, I would also like to thank my mother, whose love and blessings have enabled me to come this far in life. She has always been supportive on every decision I made.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTTP	HyperText Transfer Protocol
DASH	Dynamic Adaptive Streaming over HTTP
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	eXtensible Mark-up Language
REST	REpresentational state transfer
API	Application Program Interface
GUI	Graphical User Interface
3DTI	3D Tele-Immersion
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
SQL	Structured Query Language
RDBMS	Relational DataBase Management System
ACID	Atomicity, Consistency, Isolation, Durability

CHAPTER 1

INTRODUCTION

During the past decade, the potential of 3D tele-immersion (3DTI) applications has gained the attention from both academia and industry. While most commercial 3D systems are specialized for sole purpose (e.g., Oculus), the development of 3DTI system is currently aiming at multi-purpose, multi-sites, and multi-modal platform in order to enable a variety of user activities including e-learning, remote therapy, and interactive gaming. However, there are still a lot of unsolved challenges in the domain of 3DTI system including omnidirectional view and multi-modal 3D video streaming. In this chapter, section 1.1 gives an overview of the current 3DTI and relevant 3D systems. Besides, current video streaming technologies are also summarized in section 1.1. Section 1.2 refines the initial motivation and focuses our research questions on omnidirectional view specifically in 3DTI systems. Section 1.3 summarizes the approaces to address the problems in both real-time rendering and on-demand streaming manner in sections 3 and 4. The main contributions of this master thesis are summarized in section 1.4. Section 1.5 gives the organization of this master thesis.

1.1 Overview

3DTI system enables remote users to view the local scene and even to interact with local users in a virtual 3D space, which has promoted lots of potential applications such as remote therapy [1]. In general, a 3DTI system consists of several sites distributed at different physical locations. All the sites transmit the 3D data captured by multiple 3D cameras over either a dedicated network or a shared network such as Internet. After each site gathers the data from all the other sites, it render everything in a common virtual space.

Also, video streaming has become very popular and it takes up the majority of the Internet traffic. Different multimedia transport protocols are developed and are used. Previously, people tended to rely on RTP/RTSP protocols on top of UDP to sacrifice data integrity for low latency because retransmission of TCP could cause longer delay. With the networking technology advances and the advantages of HTTP transport, many content providers have resorted to HTTP on top of TCP to deliver multimedia service for multiple reasons. The use of HTTP could leverage the off-the-shelf web servers for video delivery by reusing existing HTTP cache infrastructures on the Internet. The idea of using HTTP protocol to stream video content plus the need of adaptation leads to the idea of splicing the video into small segments with different qualities, concatenated for playback. The family of methods using this idea is called "Dynamic Adaptive Streaming over HTTP", which has been standardized by MPEG [2].

Recently, the 3D system has seized the attention of the public. Users are expecting to consume A/V services as part of multi-modal media services with additional content such as artificial graphics, textual meta-data or even haptic information. 3DTI is one example of current 3D systems. Similarly, a multi-modal 3D streaming system shares many commonalities with 3DTI. To the best of our knowledge, there is no monoscopic 3D streaming standard which can fit the streaming and rendering part of 3D systems like 3DTI, not to mention multi-modal 3D.

1.2 Problem Statement

Though there are lots of related works on 3DTI systems, along with the popularity of 3D systems, there are still some key problems unsolved. We believe the following questions are increasingly attractive and are worth investigation:

1. In a traditional 2D video system, the interactivity of the end users choosing a specified viewpoint has been crippled by the fact that they can only choose to see the scene captured by which camera, but not the scene between two physical cameras. However, essentially 3D system doesn't have this one-view limitation which means 3D system should achieve omnidirectional view. Therefore, how to achieve omnidirectional view in 3DTI system seamlessly?

2. Multi-site 3DTI technology requires full-body, multi-modal content delivery among geographically dispersed users. Nevertheless, for 3D technologies, there is no standard for 3D video format or codec, not to mention multi-modal 3D video content. We are wondering, how to efficiently stream multi-modal 3D data still with high quality?

3. Many applications such as physiotherapy necessitate on-demand 3D video streaming. The on-demand streaming requires persistent data storage compared to the real-time streaming. Due to the complexity of multi-modality, how to manage, store and retrieve multi-modal 3D data in an efficient way?

1.3 Our Approach

To answer these questions, we designed two systems: FreeViewer and OmniViewer, targeting mainly on omnidirectional view problem and multi-modal 3D video streaming problem, respectively.

Through depth estimation and interpolation among multiple 3D video streams captured from different angles, 3D system makes it possible that rendering a 3D space where the viewers can view from arbitrary viewpoint as if there existed a virtual camera at that viewpoint with the same view angle.

One possible solution to realize omnidirectional viewing of a physical scene is to create a complete 3D model by aggregating the 3D streams from the cameras all of which can cover 360°view. The viewing user is then able to choose arbitrary viewpoint to see any side of this complete 3D model. However, creation and delivery of such a complete 3D object makes bandwidth and computation requirements inevitably high. Hence, we need to look for alternative solutions to decrease the computational and networking requirements. For example, in 3DTI system, the back side of the model, which is invisible to the viewer, is redundant. The computation related to the data of the back could be avoided to reduce latency. We found that a viewer can only see one side of the object which is covered by at most two 3D cameras. This means to render the scene from any viewpoint at any time, our system only needs the streams from two cameras. Thus, in our FreeViewer system, we choose at most two camera streams to render the final scene to viewer to save computation power and bandwidth usage. The detailed system design is presented in chapter 3.

After the omnidirectional viewing problem, we stills need a complete end-to-end multi-modal 3D content streaming solution to realize 3D content delivery from server to user in an on-demand manner. We tackle this problem by dividing the problem into recording, storing and streaming, and leveraging 2D codec and current DASH standard to solve the problem within 3D context.

1.4 Contributions

In summary, the contributions of this thesis are as follows.

- *Achieves omnidirectional view conducted by wearable device or user input when streaming.* Through our FreeViewer system, we tackle the omnidirectional viewing problem and show the attractiveness of view-changing capabilities for a user via her wearable device such as the Google Glass in multi-camera 3DTI environment. Compared to complete 3D model reconstruction, FreeViewer/OmniViewer can achieve the same omnidirectional viewing and the same 3D video quality with less computation and network bandwidth, which is highly favorable under resource-limited scenarios. Also, Omniviewer offers omnidirectional view when streaming from any view at any time via proper GUI.

- *Leverages 2D video en-/decoding technique to store and retrieve 3D video segments (color and depth).* Since there is no standard 3D video format and 3D video codec, we leverage 2D video codec to compress 3D video in the form of a bundle of multiple RGB and Depth frame pairs. Different from 3D mesh compression, we compress the 3D video into 2D frames with a substantial compression ratio the codec of which is also compatible with DASH standard.

- *Proposes a comprehensive solution for multi-modal 3D content storage with NoSQL database.* In DASH, multi-modal data are partitioned into multiple file segments and delivered to a client via HTTP. To support on-demand multi-modal 3D video streaming via DASH, the large amount data should be properly stored for fast retrieval and management. We propose a NoSQL-based solution to store the huge amount of small files in each 3D video session and efficiently retrieve these data from NoSQL database as a dynamic file server.

- *Realizes Adaptively multi-modal 3D streaming.* In data-intensive 3D applications, such as telediagnosis, remote physiotherapy and e-learning, the data are always multi-modal and highly synchronized. Our system extends current DASH standard to support synchronized multi-modal data streaming besides audio and video.

1.5 Outline

The remainder of this thesis is organized as follows.

Section 2 introduces the fundamentals and related work of DASH[1], NoSQL database and 3DTI systems. Section 2.1 explains various important concepts in DASH such as streaming mechanism and MPD file. Section 2.2 presents the appearance and development of NoSQL database. Section 2.3 mainly introduces related work about 3DTI system with emphasis on the delivery part.

Chapter 3 aims to present the design and architecture of a real-time 3D rendering system supporting omnidirectional view, *FreeViewer* [3]. Section 3.1 to section 3.4 explain different module of the whole system, respectively.

Chapter 4 introduces OmniViewer [4], an on-demand multi-modal 3D video streaming system. we discuss what is missing in current DASH implementation to support multi-modal 3D streaming and how OmniViewer fills the hole to achieve a completely functional 3D DASH framework. Section 4.1 explains the recording procedure. Section 4.2 describes how we change the standard MPD file to make it fitting with multi-modal 3D DASH. Section 4.3 explains why we choose MongoDB as our storage database and presents the design of multi-modal 3D DASH dynamic file server. Section 4.4 introduces the design of multi-modal 3D DASH player. Section 4.5 and 4.6 present the a basic evaluation of proposed 3D encoding method and a proof-of-concept implementation of the OmniViewer system, respectively.

[1]In the following, we refer to MPEG DASH when we use DASH, unless otherwise expressed.

Chapter 5 concludes the entire master thesis with the highlights of our work and potential future scope of this work.

CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1 Dynamic Adaptive Streaming over HTTP (DASH)

DASH is an adaptive bit rate streaming technology where a multimedia file is partitioned into one or more segments and delivered to a client using HTTP protocol. It needs a *Media Presentation Description* (MPD) file used by the server to tell the clients segment information (timing, URL, media characteristics) to help them fetch the right media content. The MPD syntax is defined in XML. Typically before the start of streaming, the MPD file has to be downloaded using HTTP to acquire all the media information.

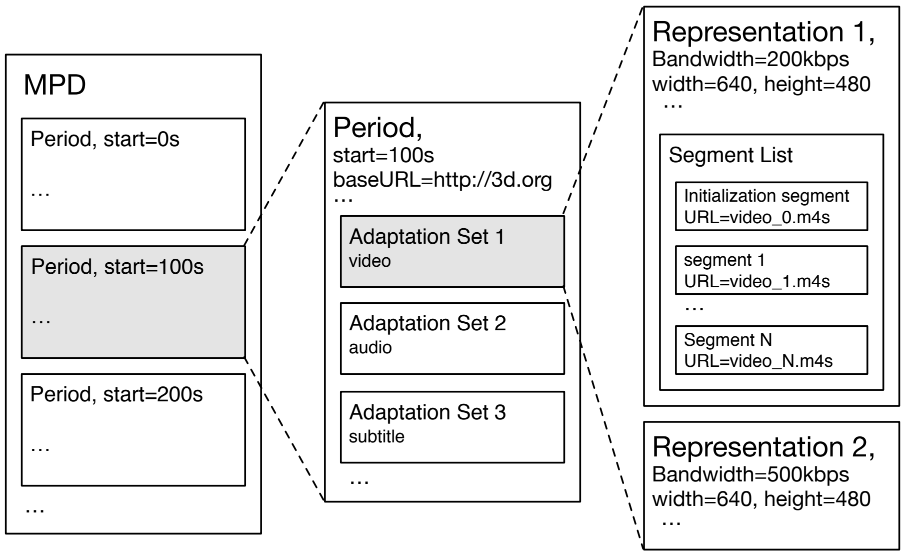


Figure 2.1: Media presentation description data model

The layout of MPD file shown in Figure 2.1 illustrates the data model of DASH. The MPD layout is composed of three layers, *Periods*, *Adaptation Sets* and *Representations*. MPD defines the video sequence with one or more consecutive periods that break up the video from start to finish. Each period contains one or more adaptation sets that contain the content of one-modality data, video, audio, etc. Each adaptation set contains multiple representations, which represent the media streams with different qualities in terms of bit rates, frame rates or video resolutions. Finally, each representation contains a series of consecutive media segments with HTTP URLs. Those URLs are either explicitly described in a *Segment List* or described through a URL template from which the client can derive a series of valid segments URLs. Although the specification provides specific guidance and formats for use with two types of containers: ISO base media file format (e.g, MP4 file format) or MPEG-2 TS, segments can contain any media data only if the client knows how to decode the media data from the segments.

There has been some previous work on improving DASH across multiple dimensions. Mok et al. [5] introduced Quality of Experience into DASH system to modify the adaptation logic. Andelin et al. [6] examine the impact of Scalable Video Coding on the client's quality selection policy. Hartung et al. [7] extend the current DASH standard to incorporate content protection techniques. Concolato et al. [8] prove the possibility of using DASH to serve as real-time media-synchronized service such as multi-modal content streaming, since DASH is codec agnostic.

2.2 NoSQL Database

The need of data management automation leads to the invention of relational database, based on the relational data model emerged with E.F.Codds 1970 paper [9], which makes data modeling and application programming much easier than in the past. Traditional tasks like recording transactions in business, science, and commerce, can be efficiently automated by the help of programming and relational database. Moreover, the relational model is well-suited to client-server programming and have proved to be the dominant database technology for storing massive structured data in almost any application. Nowadays, many organizations including companies and institutes collect tremendous amounts of customer, scientific, sales, and other forms of data. In current web application trend, the big data storage has to fulfil many requirements including high volume, high scalability, high availability, fast access and so on. In the late 90s, large web companies emerged with dramatic increases in scale on many dimensions. Such as:

- The population of users skyrocketed because applications became accessible via the web and mobile technologies.
- The amount of data collected and processed soared with the increasing users and ways to collect multi-modal data through crowd-sourcing, and it became easier and increasingly valuable to capture all kinds of data.
- The amount of unstructured data exploded.

Traditionally, most of these organizations have stored data with structured data model in relational databases for subsequent access and analysis, which seems to be the only and right choice. However, the high demand of current data storage system pushes a growing number of developers and users to turn to various types of non-relational databases, which now frequently called NoSQL databases. The term NoSQL was first coined in 1998 by Carlo Strozzi for his relational database management system (RDBMS), Strozzi NoSQL [10]. However, Strozzi made up the term simply to distinguish his solution, which still adheres to relational data model from other RDMBSs solutions which utilize SQL. The term NoSQL just emphasizes that the SQL interface is not necessary, but optional. Recently, the term NoSQL (meaning not only SQL) has come to describe a large class of databases which does not have properties of traditional relational databases and which are generally not queried with SQL (structured query language). Non-relational databases include hierarchical, graph, and object-oriented databases, which have been around since the late 1960s. However, new types of NoSQL databases are being developed. And only now they are beginning to gain market traction. Different NoSQL databases take different approaches. What they have in common is that they are not relational. Besides, they are easier to work with for the many developers not familiar with the structured query language.

There are a few reasons why people looked for alternate solutions for RDBMS. The rich feature set and the ACID properties implemented by RDBMS might be too complex to support particular applications and use cases in an efficient way. They are good, but when it comes to a specified use case, the database system could be optimized instead of serving general purpose. Currently, the principal problem of RDBMS is the trade-off between the fast growing data volume and the very expensive cost associated with scaling of the RDBMS.

In contrast, most NoSQL data stores are designed to scale well horizontally so users could thus scale a single database by running on multiple virtual servers in the cloud rather than by having to run it on a single powerful machine. Also, the one-size-fits-all notion does not fit the need of database of current application scenarios and it is better to build systems based on the nature of the application and its load. And, the need is constantly changing. The RDBMS were designed in 1980s for large high-end machines and centralized architecture. However, today's cloud technology tends to make everything distributed in case of single point of failure and lower the cost by using commodity hardware in a distributed environment. Also, today's data is not rigidly structured and does not require dynamic queries.

Numerous companies and organizations have developed NoSQL databases. Table 2.1 summarizes the most popular NoSQL databases under different categories. The most influential champions are primarily Web 2.0 companies with huge, growing data and infrastructure needs such as Amazon and Google. They developed the Dynamo [11] and Big Table [12] NoSQL databases, which have inspired many of today's NoSQL applications. It is certain that the NoSQL databases are one of the byproducts of the Web 2.0 era, which were really used only at the time when the designers of web services with very large number of users discovered that the traditional relational database management system are fit either for small but frequent read/write transactions or for large batch transactions with rare write transactions, and not for heavy read/write workloads, which is often the case for these large scale web services, such as Google, Amazon, Facebook and so on.

Table 2.1: Most popular NoSQL databases

Types	**Examples**
Document databases	MongoDB, CouchDB
Key-Value Stores	Redis, MemCache, Dynamo
Graph stores	Neo4j, InfiniteGraph
Wide-column stores	Cassandra, Hbase, Bigtable

It seems that some of the major relational database management system producers are learning something from this evolution. Therefore, most of them have taken actions to develop the NoSQL databases. For example, Amazon introduced its Dynamo distributed NoSQL system for internal use. Amazon was one of the first major companies to store much of its important corporate data in a non-relational database; Microsoft introduced some NoSQL type features such as snapshot isolation, although used at a single table level, into its newer relational database management system product in Cloud Azure.

In general, compared to RDBMS, NoSQL has many advantages in the following aspects:

- **High concurrent and high speed I/O.** NoSQL database were designed to meet the needs of high concurrent reading and writing with low latency, at the same time, which is not easy for RDBMS.

- **Efficient big data storage and access requirements.** Large applications such as social networking site (SNS) and search engines require a database system capable of storing PB-level data storage with fast access time.

- **High scalability and high availability.** With the increasing amount of data, the database needs to be able to scale horizontally easily, and ensure rapid uninterrupted service.

- **Lower costs.** With the dramatic increase in data volume, hardware costs, software costs and maintainence costs, have all increased. The better scalability of NoSQL database greatly lowers the database cost.

2.3 3DTI System

2.3.1 Cyber Collaborations through 3DTI

3DTI systems aim towards multi-purpose, multi-sites, and multi-modality to enable a wide variety of user activities [13][14][15]. In [16] 3DTI is proposed to be the medium for training and simulation in critical/hazard domains like military training and emergency healthcare. Educational 3DTI application like archeology is also proposed in [17]. In [18], 3DTI platform for performance broadcasting is proposed. The authors envision performer crew to be physically dispersed and interact remotely in the virtual world.

2.3.2 3DTI Content Archiving

Due to the high bitrate of 3DTI, various archiving schemes for compression and content analysis were proposed. In [17], compression module based on frame synthesis is proposed to lowers the bitrate of 3DTI systems. In [17], the module is paired with activity recognition to achieve dynamic bitrate adaptation. Other compression schemes for mesh-based 3DTI content are proposed in [19], which concentrate on independent 3D image compression without inter-frame coding. Analysis on 3DTI data using metadata is proposed in [20]. The authors achieve high activity detection accuracy via metadata analysis to avoid computationally expensive deep content analysis.

2.3.3 3DTI Content Delivery

Delivery of 3DTI content is not trivial due to its bandwidth consumption and soft-real-time requirement. There are two categories in 3D visual technology. Monoscopic 3D (M3D), used in traditional 3D computer graphics, creates images based on a 3D coordinate system and then displays these images onto a flat 2D device. Stereoscopic 3D (S3D), with two views of the visual scene from which the human brain extracts the depth information to perceive a stereoscopic view in 3D. There are some previous works on S3D video streaming. In [17], the authors propose a prioritization scheme for 3DTI in bandwidth limited environment. Streams are prioritized based on their shooting angles and viewers preferences. However, the authors focus on the architecture design and do not propose a streaming or delivery protocol. Pehlivan et al. [21] design an end-to-end stereoscopic video streaming system that selects transmission of mono or stereo video adaptively. Diab et al. focus on optimizing the storage in S3D streaming systems. Anahita [22] provides automatic depth adjustments to customize 3D videos and maximizes their perceived quality. In [23], a DASH-based offline streaming for 3D streams is proposed. The authors develop adaptation mechanism based on quality balancing between two requested streams and allocate bandwidth to their delivery accordingly to achieve a better quality if experience. There are also other existing multi-view client-server systems (e.g.,[24]), where a scene can be displayed from different viewpoints, which is not supported in S3D but supported by M3D. Furthermore, Venkatraman et al. [25] propose a new protocol, *MPEG Media Transport* (MMT), for 3DTI system specially, but not compatible with current protocols.

In summary, there has been significant interest from the academia and industry in streaming S3D videos. However, the research about M3D video streaming is still not well understood.

CHAPTER 3

REAL-TIME 3D RENDERING SYSTEM

FreeViewer is a 3DTI rendering system that supports omnidirectional view with low requirement of computation power and bandwidth, which is suitable for real-time p2p system such as 3DTI system on local rendering. Figure 3.1 illustrates an example of the deployment of FreeViewer. There are 4 Kinect cameras deployed in the area with the same angular difference and the same distance from the center of the capturing area. Kinect camera captures 640x480 RGB frames with D (depth) frames of the same resolution. Each pixel in the depth frame represents a depth value (distance between camera and the object) in the range between 0.4 and 4.5 m. Thus, we capture RGB and D streams as two separated but synchronized videos. A performer stays in the capturing area with a varying orientation. The objective of FreeViewer is that no matter which direction the performer is facing, it always renders the scene of the performer's front side as if there were a virtual camera faced by the performer. In fact, the scene "captured" by the virtual camera is the result of interpolation and merging of the 3D streams from 2 neighbouring cameras. FreeViewer architecture (3.3) includes several important functional services:

- 3D View Tracking Service takes the viewing direction from user through a specific user interface (Google glass in our case) via a wireless channel and appropriate frequency which determines the final rendering frame rate.

- Capturing Service takes the viewing direction and choose to collect 3D data from which two cameras and begin the capturing process.

- 3D Point Cloud Processing Service performs calibration and merging of the 3D data collect from two neighboring cameras by coordinate transformation.

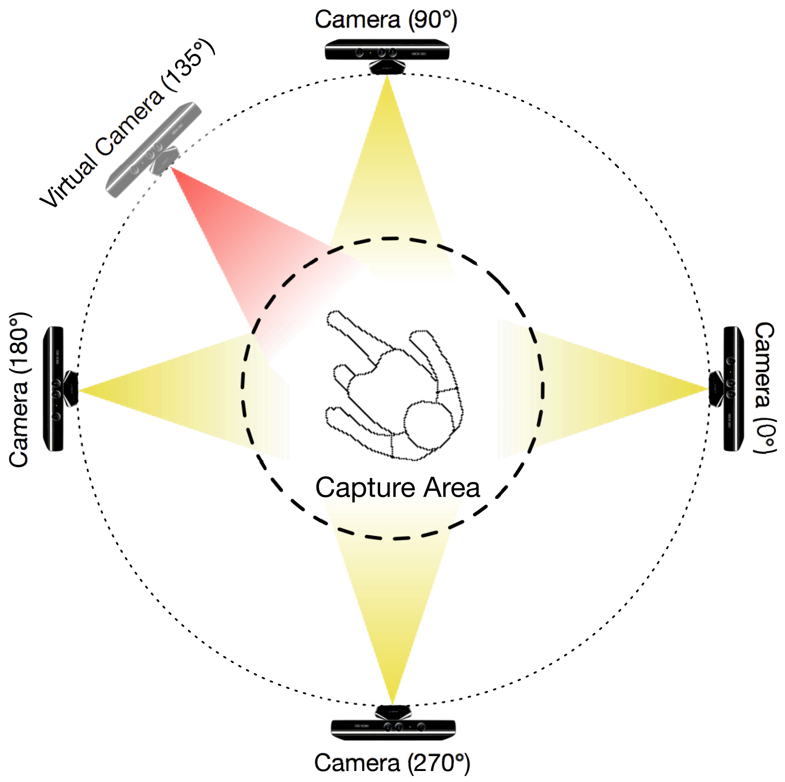


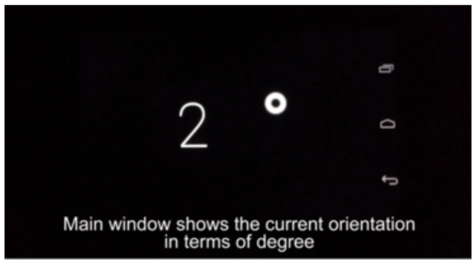
Figure 3.1: The bird view of FreeViewer deployment

- Rendering Service takes the merged 3D point cloud and viewing direction to perform the final rendering process by projecting the 3D scene onto a 2D display.

3.1 3D View Tracking Service

3D View Tracking is an advanced service that performs cooperation between wearable device and orientation monitoring module to capture user's head direction.

Traditional user interaction solutions for 3DTI systems are often naive and simply relying on standard input devices(mouse, keyboard) to control visual devices and receive visual feedback. We present a user interaction solution based on wearable devices. That is, we use Google Glass as the input interaction device to enable tracking the rotational movement of the user's head. The tracking service allows users to interact with the system in



(a) Current facing direction

(b) Connecting to server

Figure 3.2: Screen shot of our Google Glass view tracking app

a natural way without any additional manual operation such as "press some keys" or "drag the mouse". We choose Google Glass because of its portability and easiness to estimate the facing orientation compared to smartphones, with the help of the software-based rotation vector sensor which derives its data from hardware sensors via internal processing procedure. The rotation vector represents the orientation of Google Glass from which we can derive the position and view angle of the virtual camera conveniently.

3D View Tracking Service also maintains a socket connection with Google Glass to receive the most recent sensor data and then feeds them to capturing service and rendering service.

We developed an orientation tracking app on Google Glass which can show current facing direction in terms of different toward North. Users are able to see current facing direction as well as to connect to the tracking service on Freeviewer server through a TCP connection. Figure 3.2a and 3.2b show the screen shot of our Google Glass view tracking app by mapping the screen to an android smart phone via Bluetooth.

3.2 Capturing Service

As previously stated, in a 3D virtual space at a specified viewpoint, a viewer can only see one side of an object which can be covered by at most two 3D cameras. It means in order to render the scene from any viewpoint at any time, the streams from two cameras are enough. Given the orientation returned by the Google Glass, the capturing module determines the streams from which cameras to acquire according to the known physical deployment of the cameras. For example, if the current facing direction is 45°, then we

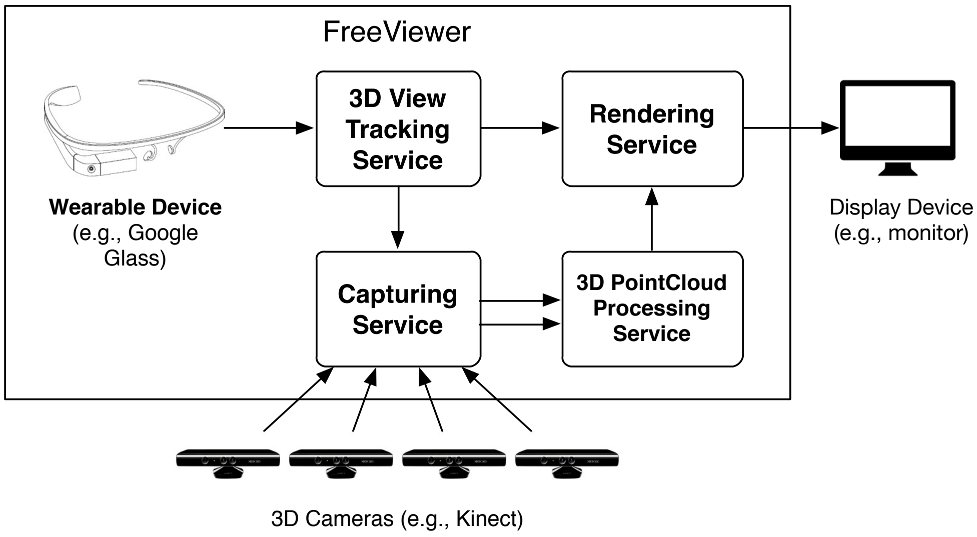

FreeViewer
3D View Tracking Service
Rendering Service
Wearable Device (e.g., Google Glass)
Display Device (e.g., monitor)
Capturing Service
3D PointCloud Processing Service
3D Cameras (e.g., Kinect)

Figure 3.3: FreeViewer architecture

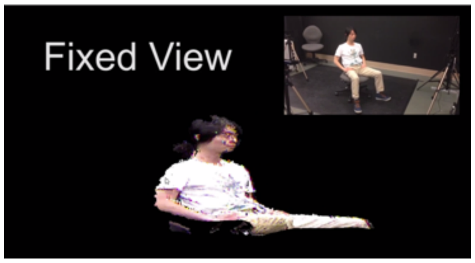
can choose Kinect(0°) and Kinect(90°). Then capturing service synchronizes the color and depth streams from both of the chosen cameras and send the frames to the processing service to create a partial 3D model.

In fact, our deployment of 4 cameras like figure 3.1 not only reduces the bandwidth consumption of bandwidth resources but also practically relieve the interference effect [26] caused when multiple structured light depth cameras (e.g., Kinect) point at the same part of a scene.

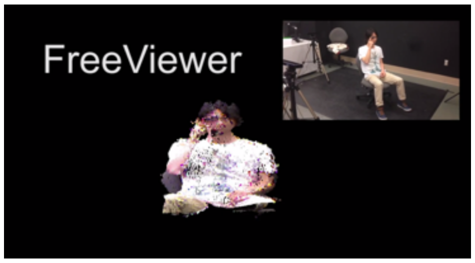
3.3 3D Point Cloud Processing Service

The 3D cloud point processing service aims to use the color stream and the depth streams from the 3D cameras to generate two 3D model (Point Clouds [27]) and to merge them in to a complete model so that the intersecting areas between them could overlap perfectly. The problem of such consistent alignment of various 3D point cloud data views is known as registration.

However, the most famous registration algorithm, iterative closet point algorithm which finds matching pairs between point clouds and creates a rigid body transformation that minimizes the distance between them until convergence, is too slow to support real-time processing. Since it is an iterative algorithm with multiple rounds until convergence, it is much suitable for offline data processing which doesn't have a strict delay constraint. For-



(a) Fixed View



(b) Omnidirectional view

Figure 3.4: A comparison between fixed view and omnidirectional view

tunately, in our deployment, we have already known the physical positions and orientations of all the camera and they are all fixed during the system running. So the calibration parameters themselves already provides fairly precisely the necessary transformation to refer all points in one frame.

Thus, we adopt the preconfiguration and calibration techniques to obtain the coordinate transformation relationship between a main camera and the other cameras. Then after the system starts to run, the processing service first performs transformation computation to convert each pixel on color/depth frame to a 3D point with color (R, G, B) and coordinate (x, y, z) in 3D space and then transforms both the 3D point clouds from the camera-based coordinate system to a global coordinate system, where the two point clouds align with each other properly.

Finally, this module delivers the merged point cloud, which is also a partial 3D model, to the rendering service.

3.4 Rendering Service

The rendering service takes two inputs, the performers current orientation, coming from the 3D view-tracking service, and the correspondent partial 3D model in terms of merged point cloud. Based on the users orientation and the geometric relationship between the deployed cameras, the rendering service calculates the virtual cameras viewpoint and view angle. Then it uses standard OpenGL API to render the synthesized scene on the display device by drawing all the points in the scene and projecting the whole 3D scene onto the final 2D view (monoscopic 3D rendering). Also, in order to provide truly immersive visual feedback, the display device does not have to be a standard monitor. It can be a head-mounted display or even another Google Glass with OpenGL ES support. Figures 3.4a and 3.4b show a comparison between fixed view and Omnidirectional view in FreeViewer, where we can find that with FreeViewer, we always are able to view the front side of the target.

CHAPTER 4

ON-DEMAND MULTI-MODAL 3D STREAMING SYSTEM

In Section 2, we gave an overview of the standard 2D DASH and current research of 3D video streaming. Now we discuss the overall architecture of the OmniViewer system that has modifications needed for the support of multi-modal 3D DASH. As illustrated in Figure 4.1, the architecture consists of three main parts: a 3D content recording subsystem to prepare 3D media data, a 3D content storage/web server to serve the MPD file and the data, and the OmniViewer 3D DASH player to process the 3D content segments delivered and to playback the multi-modal 3D content (3D video, audio, other sensory data). We will give more details of each part in the following subsections.

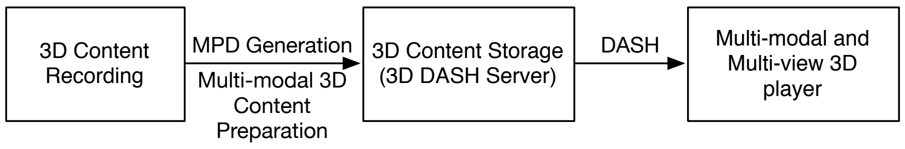

3D Content Recording
MPD Generation
Multi-modal 3D Content Preparation
3D Content Storage (3D DASH Server)
DASH
Multi-modal and Multi-view 3D player

Figure 4.1: The architecture of Omniviewer 3D DASH system

The deployment of OmniViewer is similar to FreeViewer except that OmniViewer has to collect multi-modal data including video, audio, skeleton and maybe other sensor data such as EMG data. Skeleton stream contains 3D positions of users joints (i.e., shoulder, knee, hip, etc.) at every time instance. This information is extracted from RGB-D frames by Kinect API.

4.1 Multi-modal 3D Content Recording

Compared to 2D video, a main challenge of 3D video is the existing form of 3D video. To the best of our knowledge, there is no mature M3D video format, which can be regarded as a series of 3D objects with timestamps, let

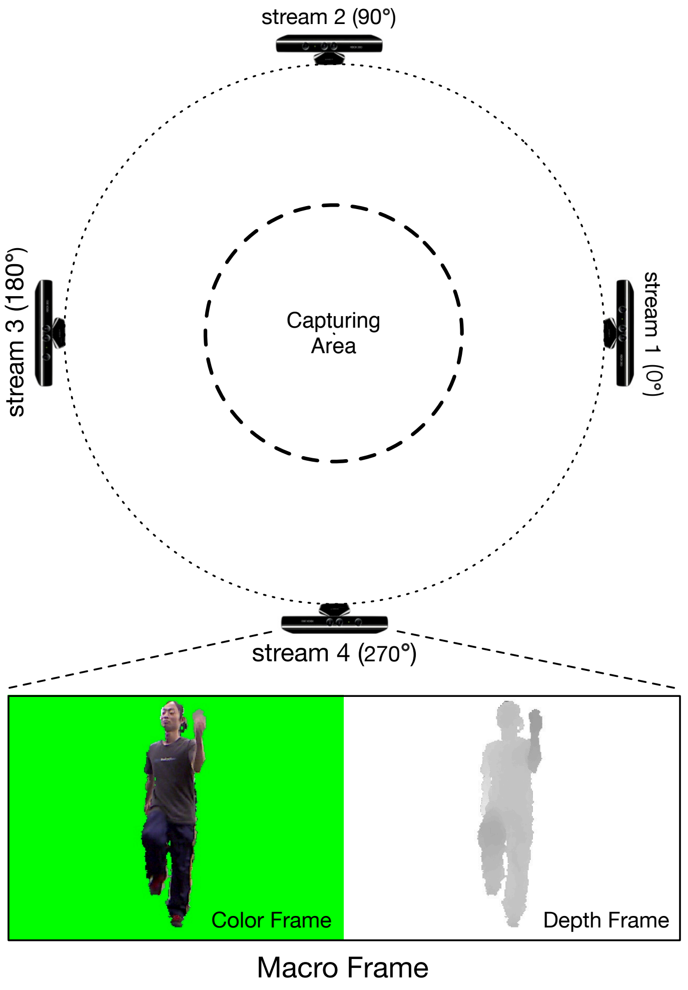


Figure 4.2: The deployment of OmniViewer recording site

alone en-/decoding techniques. Furthermore, the way to capture 3D object continously in the form of 3D video to cover omni-directional view is another challenge.

In order to solve these challenges, we propose the following scheme to record multi-modal 3D contents. As illustrated in Figure 4.2, we adopt four 3D cameras (Kinects), deployed in the area with the same angular difference (90°) and the same distance from the center of the capturing area to cover an omnidirectional view and to minimize the interference effect [26] between different 3D cameras. Also, since it is hard to manually place the camera with a perfect orientation and at an accurate location, an offline calibration is needed and the resulting calibration matrix (M_{cal}) should be recorded for the OmniViewer player to use and to merge point clouds later. In order to use DASH as the streaming standard, the recording part prepares media data for DASH-compliant representation. We represent each 3D camera video as a 2D video, where each 2D frame consists of a pair of color (RGB) and depth (Gray) frames with the same resolution, stitched together horizontally to be a uniform macro frame (RGB-D). Considering we only need the target object, human body in our case, we extract human body from both color and depth frames by filling the background with a sentinel color, e.g., green and white, to reduce the encoding complexity. The macro frame in Figure 4.2 shows that the gray color will turn dark when the performer gets closer to the camera. We compress the four 2D videos using standard 2D video codec, eg., H.264, with different bitrates. As a result, our multi-view 3D video is represented as a combination of 4 2D RGB-D videos. Because popular codecs mostly are lossy, the encoding of depth frame degrades the accuracy of depth information. Thus, the higher bitrate each 2D RGB-D video has, the better quality of 3D model we can achieve. We will discuss the influence of depth frame quality on the generated point cloud in section 4.5. For audio, we only need one stream and record the audio data from microphone array of one 3D camera.

For multi-modal sensor data recording, we use skeleton and heartbeat data as examples, which are acquired by 3D camera itself and onbody heartbeat sensor, respectively. Since DASH is en-/decoder agnostic, we can choose any codec or format to convey these multi-modal sensor data. We choose JSON (JavaScript Object Notation), a lightweight data-interchange format prevailing on the Web application, as the multi-modal media file format to represent the other sensor data except audio and video. An example of the JSON file, encoding skeleton data, is as follows:

```
{
  "type": "skeleton",
  "rate": 24,
  "data": [[0.034, 0.122, 2.354],...]
}
```

All multi-modal 3D content is tightly synchronized by connecting sensors to the same PC through wired or local wireless channels, and RGB-D, skeleton, and audio are captured by devices connected to the same PC, synchronization between them is done by standard buffering and alignment.

4.2 DASH-compliant Data Representation

As introduced in section 2, DASH needs a MPD file to describe multi-modal content at the content server. The multi-modal content can be represented as a multiplexed MPEG-2 TS or ISO Base Media File, usually containing both video and audio tracks, or can be a single-modality file, for example containing only video or only audio in a specific language. An adaptation set is a set of multiple representations. At any time, the DASH player is suppposed to play at most one presentations from each adaptation set. In this paper, we manipulate a multi-modal 3D DASH MPD file containing only single-modality representation since the tight coupling in multiplexed media file fails to offer the flexibility pertaining to track selection. A MPD file of multi-modal 3D DASH is important, since we need to work with four video streams, one audio stream, and other sensory streams.

The major difference between multi-modal 3D DASH and traditional 2D DASH is the inclusion of multiple 2D video tracks and other multi-modal data streams. In MPD, each adaptation set contains the content of some single-modality data, video, audio, etc. Thus, we have to configure the MPD file with at least four adaptation sets, all of which contain video streams, i.e., videos recorded by the 4 3D cameras. Plus, audio or other media data are assigned to an individual adaptation set. We label the first four adaptation sets with identifiers 1, 2, 3 and 4 for the four video tracks, and the rest of adaptation sets for audio or other media data is labeled with identifiers 5 or more. The OmniViewer DASH player is able to recognize media type by examining the *mimeType* property of each presentation. For example, "video/mp4" for video, "audio/mpeg" for audio, or as presented in this work, "application/json" for multi-modal sensor data, such as skeleton or hearbeat sensor data.

Another challenge is synchronization. The multi-modal 3D data provider knows the frame rate of the 3D video. However, the sampling rate of different sensors may be different. Thus, we segment each media into continuous segments with the same duration, but possibly with different rates. Finally, we generate the same number of segments for each media, the order of which match each other in the *SegmentList* tags in the MPD file. When playback starts, the player is able to play different media data at the rate denoted in the segment. For example, if the "rate" is set as 24 in the JSON file,

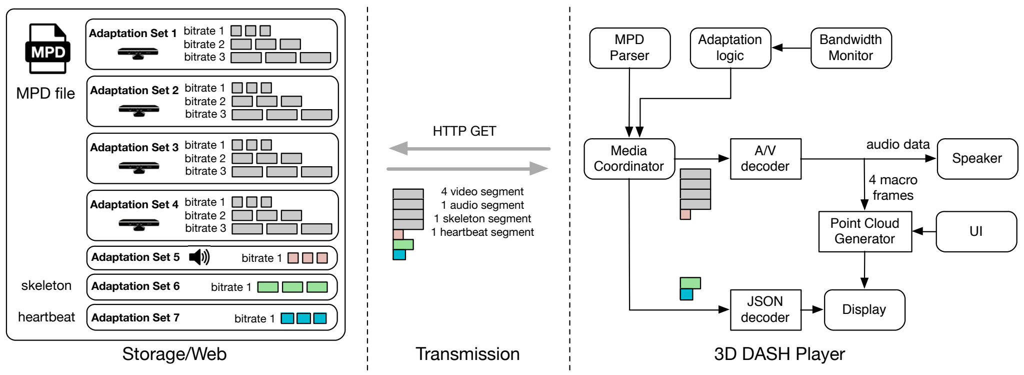


Figure 4.3: MPD file and the Multi-modal 3D DASH Player

the player will play the data of one frame every 1/24 second. Then all the data would be synchronized at the player side. It should be noticed that due to variable GOP (group of pictures) size, introduced by the encoder, sometimes the video or audio segments can not be segmented with perfectly same duration. It would not be a problem only if the player knows the playback rate of each media because if the buffer of one media is empty, the player will try to download the next one and the playback will pause if not all the media data is available.

Therefore, in our example, there will be seven adaptation sets, four for video, one for audio, one for skeleton data and one for heartbeat sensor data. The OmniViewer player knows how to decode these data, respectively. An example of MPD file structure is illustrated in Figure 4.3. For each stream, it can have multiple sets of data with different bitrates.

4.3 Multi-modal 3D DASH Content Storage

Like MPEG-DASH, OmniViewer works by splitting the data into a sequence of small HTTP-based file chunks, each segment containing a short interval of playback time of single-modal content. all the segments of different content represents a fixed interval of playback of the whole multimedia work, such as a patient session in remote physiotherapy. for each file chunks of each modality, there are a variery of different chunks with different qualities, i.e., alternative chunks with different bitrates are also available. When the session is played by our OmniViewer player, according to current network conditions, the player should automatically selects from the alternatives the next session chunks to download and playback to achieve the best quality of exprience of users. Hence, the player is able to stream the 3D video seamlessly and adaptively without stalls.

In the OmniViewer system, 3D DASH content are stored as a lot of small chunked files for video and audio, or json files for sensor data. For one session, we usually have one mpd file, and thousands of these small file segments, which are hard to maintain and even query. In the standard MPEG-DASH, all the chunk files are served in a standard static HTTP server. Since each one of the media segments and the init segments are each stored in separate files and the MPD refers to relative URLs for those files. The player always first downloads the MPD file and then chooses to download the segments based on the URLs provided by the MPD file. Also, care should be taken to configure the static file HTTP server to return the correct Content-Type for the MPD file and segment files.

When the amount of sessions recorded increases, the organization and management of so many small files becomes a challenging problem. In this section, we primarily deal with 3 questions:

- How to upload 3D video session simply and efficiently?
- How to query 3D video session content faster and easier?
- How to retrieve 3D data in a dynamic way?

To answer the questions above, we propose a dynamic DASH server powered by Node.js with MongoDB as backend database to store all the 3D multi-modal video content. The dynamic DASH server has the following features:

- Supporting RESTful API to upload 3D data through HTTP post request.
- All the data are stored in NoSQL database (MongoDB) with other metadata for single or batched query and delete operations.
- All the files are served not as a static file but binary data retrieved from database dynamically.

4.3.1 3D DASH Database Selection

After the recording of multi-modal 3D data, we have to find an elegant method to upload these data to the cloud or data server where the data can be safe and secured with redundancy in case of loss and damage. Usually, people just place all the segment files manually in appropriate server folders, and then feed the player with a link to look up files specifying the name and relative URL of these folders. Usually different folders contain data of different sessions.

For small DASH servers, such a solution is acceptable. But for a complete multi-modal 3D system includes recording, storage and playback, we require every procedure can be executed efficiently and automatically. Manual data transfer is acceptable for rare uploads. However, if the data uploads happen frequently, this solution is not acceptable since it requires a lot of human labor and is not scalable. Since the data size is soaring while the hard disk size of a specific web server is limited. When the hard disk is full, there will be no place to hold additional data.

In our use case, we observe two facts.

1. we have to store and a lot of files and retrieve them frequently.

2. These files are all very small (under 16MB).

For data storage, the first solution arises in our head is database. As we discussed the background, compared to the RMDBS, in our use case, the database system could be optimized instead of serving general purpose. In 3D DASH, our requirement includes efficient storage and retrieval of a large amount of small files and scalability. NoSQL data stores are designed to scale well horizontally so users could thus scale a single database by running on multiple virtual servers in the cloud rather than by having to run it on a single powerful machine. Also, it is simple to support particular operation of one special use case in an efficient way. As we stated, there are mainly three categories of NoSQL database: Key-value Databases, Document Databases and Column Family Databases. Among these three categories, in order to have more freedom in how we query the data and optimize for read-heavy work (our 3D DASH has more read operations than write operations), we choose MongoDB, a popular NoSQL document database as our 3D content storage database.

MongoDB (from “humongous”) is an open-source document database. The main abstraction and data structure in MongoDB is a document. Documents consist of named fields that have a id and multiple values. The field values can be scalar (string, numeric, or Boolean) or compound (a document or array). While relational databases are designed for structured and interdependent data; key-/value-stores operate on uninterpreted, isolated key-/value-pairs; document stores like MongoDB are designed for data (contained in documents) which correspond to schema-free document and only have some flexible structure known to applications as well as the database itself. The flexibility here means the schema has to be known by application by defining it at the application level which is ignorant to the database. but some part of the schema could be optional other than all the fields are necessary. The advantages of this approach are that first schema migrations are unnecessary, which introduces a lot of overhead in the relational databases; secondly, compared to key-/value-stores data can be evaluated and queried more sophisticatedly. So, collections comprise the only schema in MongoDB, and secondary indexes must be explicitly created on fields in collections. MongoDB also provided indexes on collections and supports map-reduce for complex aggregations across documents. MongoDB supports dynamic queries with automatic use of indices, like RDBMSs. MongoDB allows specifying indexes on document fields of a collection. The information gathered about these fields is stored in B-Trees and utilized by the query optimizing component to to quickly sort through and order the documents in a collection thereby enhancing read performance.

MongoDB does automatic sharding, the process of storing data records across multiple machines, to meet the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provides an acceptable read and write throughput. Sharding solves the problem with horizontal scaling with automatic fail over and load balancing. With sharding, you add more machines to support data growth and the demands of read and write operations.

MongoDB uses asynchronous replication for redundancy and fail over. A replica set is a group of mongod (the primary daemon process for the MongoDB system) instances that host the same data set. One mongod, the primary, receives all write operations. All other instances, secondaries, apply operations from the primary so that they have the same data set. The primary accepts all write operations from clients. A replica set can have only one primary. To support replication, the primary records all changes to its data sets in its operations log. The secondaries replicate the primary operations log and apply the operations to their data sets such that the secondaries data sets can be consistent with the primary data set. If the primary fails, the replica set will elect a secondary to be primary. It does not provide the global consistency of a traditional DBMS, but a local consistency on the up-to-date primary copy of a document.

MongoDB provides also atomic updates on fields that helps concurrent access to the same document.

4.3.2 3D DASH Content Upload and Download

A record in MongoDB is a document, a data structure consisting of multiple field and value pairs which are similar to JSON formats. The document can be nested that value part can be other document, array or arrays of documents. For 3D DASH, each file is stored as a document in MongoDB. The structure looks like as follows:

```
{
    "_id": ObjectId("563a2acdadf55ee91f6c39a2"),
    "size": 63031,
    "name": "seg_macro_2_500k40.m4s",
    "user_id": 4,
    "session_id": 10,
    "content_type": "application/octet-stream",
    "data": <Binary Data>,
    "date_created": ISODate("2015-11-04T15:57:01.093Z")
}
```

The structure of 3D DASH file document is really simple and concise. MongoDB uses BSON, a binary serialization format used to store documents and make remote procedure calls in MongoDB. BSON supports a lot of data types as values in documents, including *Double*, *String*, *Date* and even *Binary data*. It is worth noting that the maximum BSON document size is 16MB. The maximum document size helps ensure that a single document cannot use excessive amount of RAM or, during transmission, excessive amount of bandwidth. To store documents larger than the maximum size, MongoDB provides GridFS, a specification for storing and retrieving files that exceed the BSON-document size limit. However, in 3D DASH, even with high bitrate, a video segment is much smaller than 16MB, not to mention MPD file. So a single document is enough for a single file storage. GridFS is beyond the scope of this thesis, so we will not discuss it here.

The *_id* field defines the unique identifier for each document that act as a primary key. *size* and *name* shows the size of the file stored in this document. Both *user_id* and *session_id* indicate which user and which session of this file belongs to and we assign them with unique numerical identifiers. The real binary data of this file are kept in *data* field. Also, it includes the timestamp when this document is created and the *Content-Type* of the file. If it is a MPD file, the *Content-Type* should be "application/xml".

On the server side, we design a RESTful API to process both upload and download requests. REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services. RESTful API has many advantages including firewall traversal and client ignorance. The REST style emphasizes that interactions between clients and services is enhanced by having a limited number of operations. Flexibility is provided by assigning resources their own unique URLs. Because each verb has a specific meaning (GET, POST, PUT and DELETE), RESTful API avoids ambiguity.

In 3D DASH, an example of our RESTful endpoint structure for upload and download are illustrated in figure 4.4:

$$\underbrace{\textbf{http://3d-dash.net/}}_{\text{domain}}\underbrace{\textbf{user\_4/}}_{\text{user}}\underbrace{\textbf{session\_16/}}_{\text{session}}\underbrace{\textbf{seg\_macro\_2\_500k40.m4s/}}_{\text{file}}$$

Figure 4.4: RESTful API endpoints structure example

Figure 4.4 illustrate the endpoint structure of download, split by slash. Actually the upload endpoint is similar but does not have the file part, i.e. only domain, user and session parts.

Domain gives the base URL of the server, which can be either the public IP address or human-readable domain name with appropriate port number (if applicable).

User identifies which user this request is targeting.

Session identifies for a specific user designated in user part, which session this request is aiming at.

File is only used in download request, indicating the file name of the file that the HTTP GET request is requesting.

When a user wants to upload a file, she will send a HTTP POST request with a multi-part encoded file and its file name to the endpoint. The server receives this POST request with corresponding file name and also knows the user ID and session ID by parsing the endpoint route parameters. For example, in figure 4.4, the server could parse the second and third part of the endpoint URL to know the request is for user with ID 4 and her session with ID 16. When the server gets all the data, it will analyze the file, generate a document for this file including all the fields we stated before and save the document into our MongoDB database for persistent storage.

When a user or player tries to download a file, though the behaviour of our 3D DASH server looks like a normal static file server. The principle behind it is totally different. Actually our 3D DASH server is able to be seen as a dynamic file server since it doesn't store any file by itself. All the files are stored in our MongoDB database in the form of document with additional metadata such as size and name. The download request is a normal HTTP GET request aiming at a file identified with a file name in the endpoint URL. When the server get this GET request, it will go through the following steps:

1. Parse the URL to get all the parameters required, user ID, session ID, and file name.
2. Query the MongoDB database with these three parameters.
3. If the file is not found, return a response with status 404 (not found). Otherwise, create a buffer to hold the binary data of the file stored in the document found, and then stream the data as the response back to the user.

In such a way, the server can reads any file from database as requested and stream it to the user dynamically. The server itself is similar to a file broker between database and user. As the data grows, the database easily scales with the sharding technique as we introduced before.

4.4 Multi-modal 3D DASH Player

Different from 2D DASH, 3D DASH player shown in Figure 4.3, has to render the video in a 3D environment created in an OpenGL [28] context. When playback begins, the player first downloads the 3D DASH MPD file and parses it to get the information of all the periods, adaptation sets and representations. As we agreed, the first four adaptation sets with the video content are for 3D video rendering, the adaptation sets left contain other media data except for video. Also, the player has at least two decoders, one for A/V content, one for the other sensory data.

The player opens one media coordinator for 3D video to manage four video streams and one for each other modality. The video session manager will download all segments of the current sequence before downloading the segments of the next sequence in case of bandwidth hogging. For example, the media coordinator will not begin to download any one of the second segments of the four video streams until the first segments of all the 4 video streams have been downloaded already.

When all multi-modal data for the current frame are available, it is time to render. The primary task for the player it to transform 4 macro frames decoded by A/V decoder into a full 3D model (point cloud). The process mainly has three steps.

1. **Frame selection.** In a 3D virtual space at a specified view angle, a viewer can only see one side of the performer which can be covered by at most two 3D cameras. This means in order to render the scene from any view angle at any time in OmniViewer, OmniViewer player only needs the macro frames from two cameras that depend on the view angle, set by the viewer.

2. **Conversion from 2D pixel to 3D point.** The advantage of 3D over 2D is the free view, so we need the conversion from 2D videos to 3D video at the receiver side to render the M3D video from any view angle designated by user. Once the player gets a macro frame decoded, it can convert the pixels in 2D macro frame, representing the captured human body, into a partial point cloud, combining the RGB data from the left-half frame and the corresponding depth value from the right-half frame. As we stated before, the depth value represents

the distance d of the point constructed by that pixel. However, if we denote the 2D coordinate of the pixel as (i, j), where the coordinate of the pixel at the top left corner is $(0, 0)$, we need to find a way to convert the 2D matrix coordinate (i, j) and depth value d to the camera-view coordinate (x_v, y_v, z_v). Since we are using Kinect as our 3D camera, we use the following formula which comes from the official Kinect SDK [29] to perform the conversion:

$$\begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \begin{bmatrix} (j - \frac{w}{2})\frac{320}{w}\alpha d \\ -(i - \frac{h}{2})\frac{240}{h}\alpha d \\ -d \end{bmatrix} \tag{4.1}$$

where h and w are the height and width of color/depth frame (480 and 640 in our case) and constant $\alpha = 3.501 \times 10^{-3}$. Then we get the 3D point coordinate in the camera-view coordinate system with camera at the origin shooting towards $+z$ axis.

3. **Coordinate transformation.** It should be noted that each camera has its own camera-view coordinates, based on their perspective and position. In order to merge the point clouds of all the cameras together by their geometric relationship, we have to do coordinate transformation of each point in every camera-view coordinate system into the uniform world coordinate system, where the center of capture area is at the origin, camera 0°is at $(0, 0, R)$ and camera 90°is at $(R, 0, 0)$, etc. Then according to the camera position, we can transform the coordinates of each point to the world coordinate (x, y, z) as follows:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & R\sin\theta \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & R\cos\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} M_{cal} \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} \tag{4.2}$$

where R denotes the distance between the center of capture area and any camera and θ denotes the shooting angle of each camera (0°, 90°, 180°, 270°) illustrated in Figure 4.2. M_{cal} represents the fixed calibration matrix of each camera.

After the final transformation, the OmniViewer player will use standard

OpenGL functions to render the final monoscopic view from the view angle set by the user. Plus, the audio of current frame will be decoded and played with the audio output device. The other sensor data will be decoded by the JSON decoder and presented through proper user interface (an example will be given in the next section).

As a 3D DASH player, another important module of OmniViewer player is bandwidth adaptation. This module has to measure current available bandwidth and according to the adaptation logic, chooses the intended segments to download. The basic adaptation logic for video is to keep the bitrates of the four video streams consistent and maximize them while guaranteeing the maximum bitrates[1] of other modal data.

4.5 Evaluation

In section 4.1, we proposed to use the H.264 2D video codec to compress macro frames. The algorithms behind the codec are based on small blocks similarity between a sequence of images, which results in lossy compression. For color frame, it is acceptable since only the color is modified. However, the gray scale of the pixels of the depth frame is more sensitive than color, because it is closely related to the distance of the corresponding points in the 3D space. The change of their values would modify the generated point cloud.

The influence of compression on the conversion from 2D frame to 3D point cloud stems from 3 aspects. 1) As previously stated, the colors of the points in the color frame can be distorted. 2) The gray scale of the pixels that stands for the distance value in the depth frame can be distorted too, so the resulting point cloud may get deformed. 3) We use a sentinel gray scale to represent an invalid pixel meaning transparency. The compression can lead to the change of the sentinel gray scale, which finally produces additional noisy points that should not exist.

Any lossy 2D video compression codec causes variations in color. However, the human eye is not very sensitive to subtle variations in color. Thus, we mainly focus on the variation in gray scale in the depth frame and the increase of the number noisy points.

[1]Usually other modal data except for video only has one representation.

We use our recording deployment and four Kinect cameras to record four videos consisting of proposed macro frames for 60 seconds. Then we compress these videos using H.264 codec with 20 different bitrates, from 100kbps to 2000kbps with a step length of 100kbps. Each color frame or depth frame has a resolution of 640×480 and the resolution of the resulting macro frame is 1280×480. For the depth frame, we use 8-bit grayscale 1 to 250 to represent a distance of 2 meter to 3 meters and use 255 as the sentinal value. After the compression, if the grayscale of a pixel is equal or below 250, we regard it as a valid point and perform the conversion from 2D pixel to 3D point. Otherwise, we ignore it.

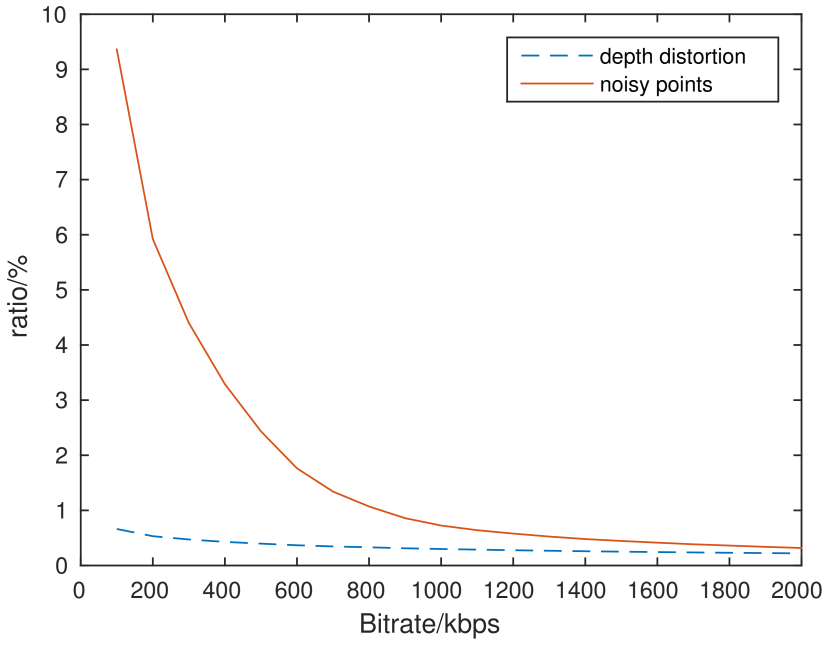
We choose 1000 consecutive frames synchronized from each video and analyze the average ratio of grayscale distortion and that of noisy points under different bitrates compared to the raw data. The result is illustrated in Figure 4.5a. From the figure, we can see that the average ratio of depth distortion caused by lossy compression is always below 1% and doesn't change much as the bitrate goes up. In contrast, the ratio of the number of generated noisy points to the number of valid points can be as high as 9.36% when the bitrate is 100kbps. With the increase of bitrate, the ratio of noisy points decreases quickly and then flattens out. When the bitrate is over 800kbps, the ratio of noisy points falls below 1%. We also applied the same configuration using VP8, a popular video compression format for web use, the result of which is shown in Figure 4.5b. When the encoding bitrate is 200kbps, the ratio of noisy points is 4.45%, better than 5.92% with H.264 under 200kbps. Overall, the two figures are similar with subtle differences.

Therefore, it is feasible to use 2D video codec to compress depth value, but too bad quality may induce too much noise in the point cloud and futhermore degrade user experience.

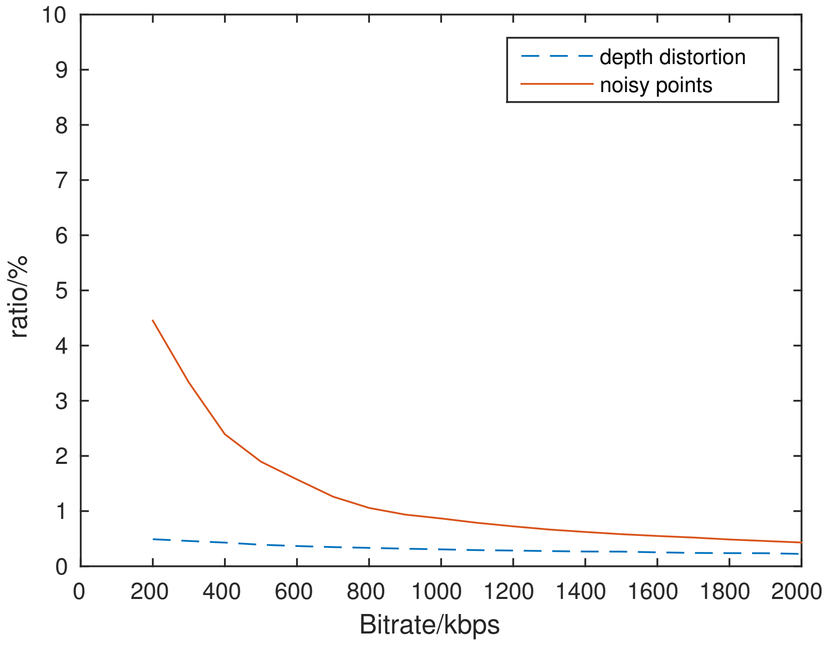
We use a PC with 8-core 3.40GHz CPU and 8GB RAM to encode these 4 RGB-D video with 1000 consecutive frames with 30 FPS each and it costs 359 seconds.

4.6 Implementation

We are concerned with multi-modal 3D DASH media generation, 3D DASH server and playback.



(a) H.264 codec compression



(b) VP8 codec compression

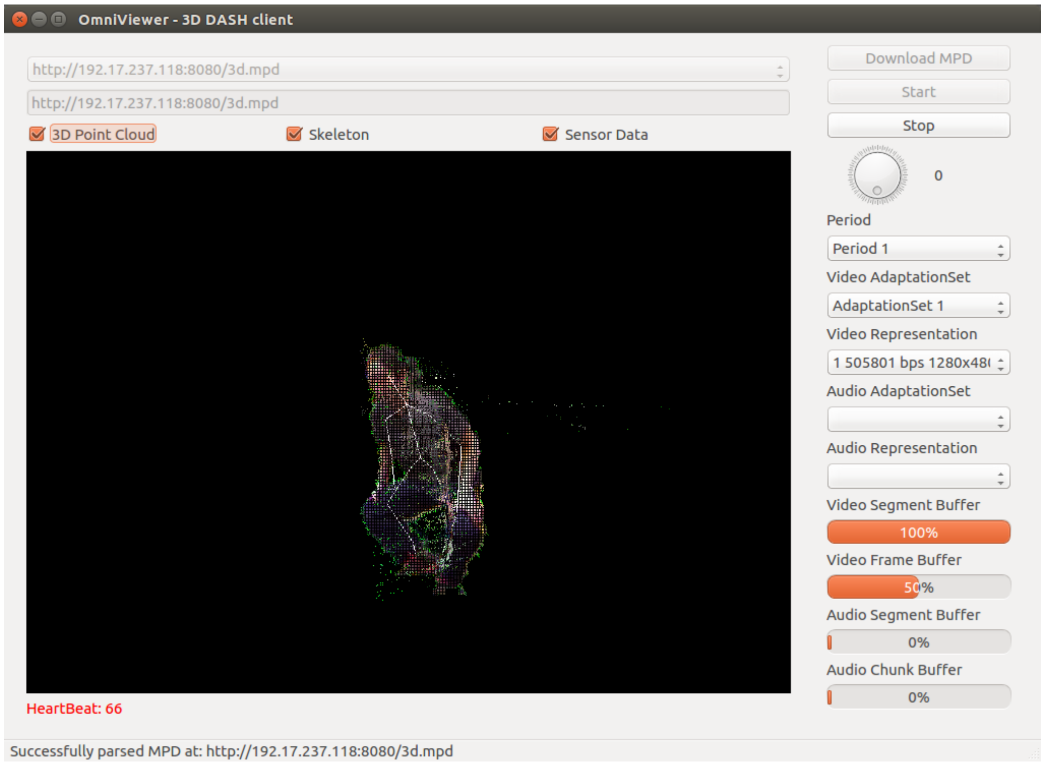
Figure 4.5: Influence of H.264 and VP8 compression on depth data

We compress four videos from macro frames using H.264 codec with different bitrates and slice them into 1s DASH segments using MP4Box [30]. For audio, we only need one stream and apply similar operation. For other media data, in our case we incorporate skeleton and heartbeat data, encoding them in separate JSON files, one file per segment including the data for each frame.

All the generated segments and customized MPD file are upload ed to and stored in a MongoDB database maintained by Mongolab [31] through a client develop in python with requests library [32]. The server acts as a dynamic file server implemented with Express, a web application framework based on Node.js [33] web framework and Mongoose [34], an elegant MongoDB object modeling for Node.js.

We have implemented OmniViewer player based on the open-source sample player that comes with libdash [35], an open-source C++ library which implements the full MPEG-DASH standard defined in [2]. We test our implementation on another laptop installing ubuntu 14.04, which has 4GB RAM and 4-core 2.60GHz CPU. The average time (1000 3D frames) of the conversion from two 2D macro frames to a merged 3D point cloud is 40.5 milliseconds, which could be improved by a more powerful machine. Under a local 100Mbps wireless network, the initial delay from the first request to playback is about 400 milliseconds with the same machine.

A screen shot of OmniViewer player on Ubuntu 14.04 is shown in Figure 4.6. The GUI uses a dial to let viewer set the preferred view angle. It also has three check boxes to help viewer watch the 3D point cloud, skeleton and sensor data. The 3D point cloud and skeleton are rendered in OpenGL, while the sensor data is shown in a textbox.



Successfully parsed MPD at: http://192.17.237.118:8080/3d.mpd

Figure 4.6: The screenshot of Omniviewer player

CHAPTER 5

CONCLUSION

In this thesis, we explore omnidirectional view problem and multi-modal streaming in 3D Tele-immersion system in two parts: real-time rendering system and on-demand streaming system.

For real-time rendering system, we propose *FreeViewer* that allows viewers to see arbitrary side of the performer by intelligently choosing the streams of a subset of cameras and changing viewpoint in a 3D virtual space. It shows the attractiveness of view-changing capabilities for a user via her wearable device such as the Google Glass in multi-camera immersive environment. The view-tracking technique is based on the sensor data from wearable devices. Compared to complete 3D model reconstruction, FreeViewer can achieve the same omnidirectional view and the same 3D video quality with less computation and network bandwidth, which is highly favorable under resource-limited scenarios.

For on-demand streaming system, we propose *OmniViewer*, a multi-modal 3D DASH system, that supports omnidirectional view, adaptive, multi-modal 3D content streaming based on MPEG DASH. OmniViewer does not require change in DASH standard and proposes to use multiple 2D videos to represent one 3D video. Also, OmniViewer gives an example to include multi-modal data into DASH. OmniViewer combines DASH, 3D and multi-modal data together, which expands the current scope of both DASH and 3D systems. Besides recording and delivery, we propose a new way to store and serve multi-modal 3D DASH content, which is also suitable for other read-heavy file system that has a lot of small files to be stored. We leverage document-based NoSQL and RESTful API to realize a dynamic file server with all the file stored in NoSQL database, which also provides fast query and retrieval. The design of the 3D DASH server makes OmniViewer more scalable. The versatility and flexibility of OmniViewer can be leveraged to fit many use cases, such as remote physiotherapy, entertainment and other 3D activities.

For the future work, we may research other related but challenging problems such as skeleton merging and view rendering offloading for mobile device at the server side.

REFERENCES

[1] K. Nahrstedt, “3d teleimmersion for remote injury assessment,” in *Proceedings of the 2012 International Workshop on Socially-aware Multimedia*, ser. SAM ’12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2390876.2390884 pp. 21–24.

[2] “Information technology - dynamic adaptive streaming over http (dash) - part 1: Media presentation description and segment formats,” International Organization for Standardization, Geneva, Switzerland, Tech. Rep. ISO/IEC DIS 23009-1, 2012.

[3] Z. Gao, S. Chen, and K. Nahrstedt, “FreeViewer: An Intelligent Director for 3D Tele-Immersion System,” in *Proceedings of the ACM International Conference on Multimedia*, ser. MM ’14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654873 pp. 755–756.

[4] Z. Gao, S. Chen, and K. Nahrstedt, “Omniviewer: Enabling multi-modal 3d dash,” in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM ’15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2733373.2807971 pp. 801–802.

[5] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, “Qdash: A qoe-aware dash system,” in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys ’12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155558 pp. 11–22.

[6] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, “Quality selection for dynamic adaptive streaming over http with scalable video coding,” in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys ’12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155580 pp. 149–154.

[7] F. Hartung, S. Kesici, and D. Catrein, "Drm protected dynamic adaptive http streaming," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943589 pp. 277–282.

[8] C. Concolato, J. Le Feuvre, and R. Bouqueau, "Usages of dash for rich media services," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943587 pp. 265–270.

[9] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, June 1970. [Online]. Available: http://doi.acm.org/10.1145/362384.362685

[10] "Nosql: a non-sql rdbms," http://www.strozzi.it/cgi-bin/CSA/tw7/I/en US/nosql/.

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294281 pp. 205–220.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323 pp. 15–15.

[13] G. Kurillo and R. Bajcsy, "3d teleimmersion for collaboration and interaction of geographically distributed users," *Virtual Reality*, vol. 17, no. 1, pp. 29–43, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10055-012-0217-2

[14] S. Schulte, S. Chen, and K. Nahrstedt, "Stevens' power law in 3d tele-immersion: Towards subjective modeling of multimodal cyber interaction," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654998 pp. 1133–1136.

[15] H. Fuchs, A. State, and J.-C. Bazin, “Immersive 3d telepresence,” *Computer*, vol. 47, no. 7, pp. 46–52, 2014.

[16] A. Sadagic, M. Klsch, G. Welch, C. Basu, C. Darken, J. P. Wachs, H. Fuchs, H. Towles, N. Rowe, J.-M. Frahm, L. Guan, R. Kumar, and H. Cheng, “Smart instrumented training ranges: bringing automated system solutions to support critical domain needs,” *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 2013. [Online]. Available: http://dms.sagepub.com/content/early/2013/01/25/1548512912472942

[17] S. Chen and K. Nahrstedt, “Activity-based synthesized frame generation in 3dti video,” in *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, July 2013, pp. 1–6.

[18] S. Chen, Z. Gao, K. Nahrstedt, and I. Gupta, “3dti amphitheater: Towards 3dti broadcasting,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 2s, pp. 47:1–47:22, Feb. 2015. [Online]. Available: http://doi.acm.org/10.1145/2700297

[19] R. Mekuria, M. Sanna, S. Asioli, E. Izquierdo, D. C. A. Bulterman, and P. Cesar, “A 3d tele-immersion system based on live captured mesh geometry,” in *Proceedings of the 4th ACM Multimedia Systems Conference*, ser. MMSys ’13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2483977.2483980 pp. 24–35.

[20] A. Jain, A. Arefin, R. Rivas, C.-n. Chen, and K. Nahrstedt, “3d teleimmersive activity classification based on application-system metadata,” in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM ’13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2502081.2502194 pp. 745–748.

[21] S. Pehlivan, A. Aksay, C. Bilen, G. Akar, and M. Civanlar, “End-to-end stereoscopic video streaming system,” in *Signal Processing and Communications Applications, 2006 IEEE 14th*, April 2006, pp. 1–4.

[22] K. Calagari, K. Templin, T. Elgamal, K. Diab, P. Didyk, W. Matusik, and M. Hefeeda, “Anahita: A system for 3d video streaming with depth customization,” in *Proceedings of the ACM International Conference on Multimedia*, ser. MM ’14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654899 pp. 337–346.

[23] A. Hamza and M. Hefeeda, “A dash-based free viewpoint video streaming system,” in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, ser. NOSSDAV ’14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2578260.2578276 pp. 55:55–55:60.

[24] J.-G. Lou, H. Cai, and J. Li, "A real-time interactive multi-view video system," in *Proceedings of the 13th Annual ACM International Conference on Multimedia*, ser. MULTIMEDIA '05. New York, NY, USA: ACM, 2005. [Online]. Available: http://doi.acm.org/10.1145/1101149.1101173 pp. 161–170.

[25] K. Venkatraman, S. Vellingiri, B. Prabhakaran, and N. Nguyen, "Mpeg media transport (mmt) for 3d tele-immersion systems," in *Multimedia (ISM), 2014 IEEE International Symposium on*, Dec 2014, pp. 279–282.

[26] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim, "Shake'n'sense: Reducing interference for overlapping structured light depth cameras," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208335 pp. 1933–1936.

[27] "Point cloud library," http://pointclouds.org/.

[28] "Opengl (open graphics library)," http://www.opengl.org/.

[29] "Kinect for Windows SDK," http://msdn.microsoft.com/en-us/library/hh855347.aspx.

[30] "MP4Box," http://gpac.wp.mines-telecom.fr/mp4box/.

[31] "mongolab," http://mongolab.com/.

[32] "Requests: Http for humans," http://docs.python-requests.org/.

[33] "Node.js," http://nodejs.org/.

[34] "mongoose," http://mongoosejs.com/.

[35] "Libdash," http://github.com/bitmovin/libdash.