OULUN YLIOPISTO
UNIVERSITY of OULU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Ali Kämäräinen

SCALABILITY SOLUTIONS FOR 3-D MULTI-USER VIRTUAL WORLDS

Master's Thesis
Degree Programme in Information Networks
May 2014

Kämäräinen A.V.J. (2014) 3-D virtuaalimaailmojen skaalautuvuusratkaisut. Oulun yliopisto, tietotekniikan osasto. Diplomityö, 60 s.

TIIVISTELMÄ

Virtuaalimaailmat voivat käsitteenä vaihdella tähtitieteellisistä simulaatiosta massiiviin monen pelajaan verkottuneisiin roolipeleihen. Jälkimmäisissä tarvitaan geneerinen ja laajennettava alustaratkaisu, jonka tieto- ja kommunikaatiomallit riittävät mielivaltaisille virtuaalimaailmoille, joilla on jopa tuhansia yhtäaikaisia käyttäjiä.

Laskennalliset skaalautuvuusongelmat, kuten grafiikan luominen ja fysiikan simulointi kolmiulotteisessa ympäristöissä ovat ovat hyvin tunnettuja. Niistä poiketen tämä diplomityö pohtii suorituskykyongelmia monen käyttäjän virtuaalimaailman asiakas-palvelin-mallin kommunikaatiossa, kun sisältö ja toiminnallisuus voivat olla mielivaltaisia.

Olemassa olevien sovellusten ja alustojen eroista huolimatta nillä on yhteisiä tarpeita. Visuaalisia kolmiulotteisia objekteja on esitettävä, niitä on liikutettava ja ja elävöitettävä, sekä jaettava tieto osallistujien kesken synkronoiden. Tämän kaiken on tapahduttava tehokkaasti.

Jotta näissä tehtävissä onnistuttaisiin, on kiinnitettävä huomiota maailmamalliin ja protokollaan, jolla välitetään tieto sisällön muutoksista. Hyödyntämällä käytettävissä olevaa geometrista informaatiota asiakas-palvelin-kommunikaatiossa ja tiedon replikoinnissa, tieto synkronoidaan osallistujien kesken vain kun se on oleellista yksittäisen osallistujan kannalta.

Tuloksena esitetään toimiva ja käytäntöön soveltuva synkronointialueiden hallintatekniikka. Sen avulla vähennetään palvelimen kaistavaatimuksia merkittävästi, kun maailma sisältää paljon liikkuvia ja muuttuvia kohteita.

Avainsanat: entiteetti-komponentti-malli, pelimoottori, asiakas-palvelin-malli, synkronointialueiden hallintatekniikka

Kämäräinen A.V.J. (2014) Scalability solutions for 3-D multi-user virtual worlds. University of Oulu, Department of Computer Science and Engineering. Master's Thesis, 60 p.

ABSTRACT

Virtual worlds can be potentially anything, ranging from an astronomical simulation to a massively multiplayer online role-playing game. Such a wide variety of potential use cases require a generic and extensible solution when considering the data and communication models for a general-purpose platform for arbitrary virtual worlds with potentially thousands of concurrent users.

As computational scalability issues, such as graphics rendering and physics simulation in 3-D environments are already well addressed with existing solutions, this thesis considers the performance problems of multi-user virtual world client-server communication when working on large virtual worlds containing arbitrary data and functionality.

Despite the differences in the existing virtual world applications and platforms, all of them share common needs: representing visual 3-D objects, moving and animating them, synchronizing all the necessary data among the participants, and storing it for continuing use and all of this in an efficient manner.

In order to succeed in this task, attention must be paid to the scene model that represents the contents of the world, and the protocol that is used to disseminate the changes in the content. Also, by utilizing the available geometric information in the client- server communication and data replication, the data is synchronized between the participants on a need-to-know basis.

As a result, this thesis presents a working and production-ready interest manager technique that helps to reduce the server's bandwidth usage considerably when a world contains lots of moving objects.

Key words: entity-component model, game engine, client-server model, interest management

TABLE OF CONTENTS

FOREWORD

First and foremost I would like to thank my supervisor professor Olli Silvén for the relentless support and encouragement during this somewhat prolonged process. I would also like to thank my former and current bosses, LudoCraft CEO Dr. Tony Manninen and Adminotech CEO Tommi Hollström, for making it possible to dedicate some of my work time to working on my thesis. Also, my former LudoCraft co-workers Jukka Jylänki and Lasse Öörni deserve a big thank you for their valuable tips regarding the subject matter. I would also like the thank Dr. Jari Hannuksela for acting as the second reviewer and Lee and Graham Walton for the proof-reading.
Also my student counselors Varpu Pitkänen and Maritta Juvani deserve special thanks for the counseling and help that I have received throughout these many, many years.

Helsinki, May 27, 2014

Ali Kämäräinen

LIST OF ABBREVIATIONS

2-D, 2D	Two-Dimensional
2.5-D, 2.5D	Two-and-a-half-Dimensional
3-D, 3D	Three-Dimensional
API	Application Programming Interface
CAVE	Cave Automatic Virtual Environment
CMS	Content Management System
COLLADA	Collaborative Design Activity
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CVE	Collaborative Virtual Environment
DIA	Distributed Interactive Application
DVE	Distributed Virtual Environment
DWTP	Distributed Worlds Transfer Protocol
FPS	Frames Per Second
GPU	Graphical Processing Unit
HTML	Hypertext Markup Language,
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
LOD	Level Of Detail
MMORPG	Massively Multiplayer Online Role Playing Game
MMO(G)	Massively Multiplayer Online Game
MUD	Multi-User Dungeon
Net-VE	Networked Virtual Environment
NTP	Network Time Protocol
RAM	Random Access Memory
RIA	Rich Internet Application
RPC	Remote Procedure Call
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
VE	Virtual Environment
VLE	Variable-Length Encoding
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
VRTP	Virtual Reality Transfer Protocol
VW	Virtual World
WAN	Wide Area Network
X3D	Extensible 3D Graphics
XML	Extensible Markup Language

FIGURES AND THEIR LICENSES

Figure 1. Dungeons & Dragons board game. URL: http://en.wikipedia.org/wiki/File:Dungeons_and_ Dragons_game.jpg. Used under the GNU Free Documentation License. Accessed 27.04.2014.

Figure 3. Rogue on a color PC. URL: http://en.wikipedia.org/wiki/File: Rogue_ Screen_Shot_CAR.PNG. Used under the GNU Free Documentation License. Accessed 27.04.2014.

Figure 4. Lucasfilm's Habitat. URL: http://scara.com/~ole/literatur/LessonsOf Habitat.html. © 1986 LucasArts Entertainment Company. Used under fair use. Accessed 27.04.2014.

Figure 5. A typical graphic MUD (screenshot of Wyvern). URL: http://en .wikipedia.org/wiki/File:Wyvern_screenshot.png. © Steven Yegge, Cabochon, Inc. Used under fair use. Accessed 27.05.2014.

Figure 7. EverQuest. URL: http://en.wikipedia.org/wiki/File:Sand_Giant_-_EverQuest_-_1999.jpg. © 1999 Sony Online Entertainment LLC. Used under fair use. Accessed 27.05.2014.

Figure 8. An ongoing meeting at Second Life. URL: http://en.wikipedia.org/wiki/File:Academia_Electronica-Instytut_Filozofii_UJ_(2013).png. Used under the Creative Commons Attribution-Share Alike 3.0 Unported license. Accessed 21.05.2014.

Figure 9. World of Warcraft. URL: http://en.wikipedia.org/wiki/File: Modified_ WoW_User_Interface.jpg. © Blizzard Entertainment, Inc. Used under fair use. Accessed 27.05.2014.

Figure 10. CAVE room with stereoscopic rendering. URL: http://en.wikipedia .org/wiki/ File:CAVE_Crayoland.jpg. Released to public domain. Accessed 27.04.2014.

1. INTRODUCTION

When dissecting the term “virtual world” (VW), the word “world” implies that there exists some kind of space in which one can exist and interact. The world can be realistic, something more abstract, or something in-between, depending each time on the purpose and use cases of the world. The word “virtual” describes that the world is artificial, not real. In this case it refers to the fact that the world is implemented by a computer.

Many other terms for virtual world are constantly present in the academic research world, such as virtual environment (VE), networked virtual environment (net-VE), distributed virtual environment (DVE), distributed interactive application (DIA), and metaverse. However, all of these definitions and acronyms refer more or less to the same concept, only emphasizing some functional aspect of the virtual world, for example collaboration, or some design or software architectural aspect, for example distributed. Also, it seems the term “virtual world” is avoided to some extent in the scientific research world in order to emphasize the more serious nature of the work – virtual world is a term often associated with commercial and especially entertainment products.

The key aspect of a virtual world is immersion, a sensation of truly being part of and roaming in a different world. Murray defines immersion as follows: "the sensation of being surrounded by a completely other reality ... that takes over all of our attention, our whole perceptual apparatus … in a participatory medium" [1]. Immersion is a concept not only limited to virtual worlds, virtual reality or computer games in general, but can be found equally in literature, movies and other art forms, especially in the narrative mediums.

Immersion can be divided into physical (perceptual) and psychological immersion [2]. Perceptual immersion means "the degree to which a virtual environment submerges the perceptual system of the user" [2]. Psychological immersion means the depth of users’ involvement and engagement with the virtual medium.

Complete photo- and audio-realism is not necessarily required for achieving a sense of immersion [3]. Realism can be divided into social realism and perceptual realism, which is commonly referred to also as photo-realism, or just simply as realism [3]. Social realism means the accuracy of users’ social interaction in the virtual world compared to the social interactions in the real life [4].

Whereas in the real world human beings’ properties and capabilities are strongly restricted by things such as the laws of physics, in virtual worlds such restrictions do not exist: species and gender is no longer decided by the DNA, and for example flying is possible. Imagination, and to some extend technology, is the only limitation.

1.1. Virtual Worlds and Virtual Reality

Virtual worlds can be potentially anything, ranging from an astronomical simulation with complex scientific models to a massively multiplayer online role-playing game (MMORPG) in a fantasy setting with thousands of other users. Bartle describes the distinguishing features of virtual worlds compared to other related non-real spaces as follows [5]:

- the world has a set of rules,
- interaction takes place in real time,

- the world is shared,
- the world is persistent, at least to some degree, and
- the user is represented and strongly identified in the world by an individual character.

Singhal and Zyda use very similar characteristics when identifying the distinguishing features of net-VEs [6]:

- a shared sense of space,
- a shared sense of presence,
- a shared sense of time,
- a way to communicate, and
- a way to share.

The graphical presentation of a user's virtual persona in a virtual world is often referred to as an *avatar*. An avatar is commonly a humanlike entity, but it can take any size and shape: an animal, an imaginary monster, or something more abstract [6]. Even though most commercial virtual worlds are avatar-centric, the requirement for existence of a single, or even multiple for that matter, avatar can be seen unnecessary for various virtual world applications.

Today's virtual worlds can be divided into two broad categories: MMORPGs and general purpose virtual worlds, commonly also referred as metaverses [7]. A MMORPG is a role playing game played by multiple players in a perpetual world: users can log in and out at any given time and the world continues to exist and evolve regardless of the user's presence. Usually players roam in the world performing quests, fighting battles, and collecting different kinds of items which help their cause. Usually the worlds are inhabited by different races, such as humans and orcs. The worlds of MMORPGs, such as World of Warcraft [8] and EVE Online [9], are usually static, meaning that the users cannot create content and interact freely. In general purpose virtual worlds such as Second Life [10], the users can socialize, interact and build quite freely with other users.

The term "virtual world" should not be confused with virtual reality (VR) which primarily concerns mechanisms by which human beings interact with computer simulation [5]. However, the terms can be seen to somewhat overlapping, especially in a modern context, as the development of consumer electronics has made many virtual-reality-like controlling devices, such as cameras utilizing computer vision and motions sensors also available for home users at affordable prices.

1.2. Applications of Virtual Worlds

Modern virtual world engines are essentially networked 3-D game engines so it's no wonder that games, massively multiplayer online games (MMOs) and MMORPGs in particular, dominate today's virtual world market share. The market is led by commercial MMORPGs with titles such as World of Warcraft, Aion, and EVE Online, but also free MMORPG titles have gained large popularity with titles such as Lord of the Rings Online, RuneScape and Dungeons & Dragons Online, for which commercial version existed many years before the free version [11, 12, 13, 14]. During the last decade virtual worlds focusing on social networking have also gained great popularity. The most successful social virtual worlds include titles such as

Habbo Hotel, a cartoonish graphical chat room aimed for teenagers and like-minded, and Second Life, a 3-D virtual world for a bit more mature audience with vast world and socially realistic interactions [15].

Virtual worlds can be used to train people for expensive and dangerous real life situations in advance in a virtual environment, for example military combat and aviation training. Distributed military virtual environments such as SIMNET developed by The United States Department of Defense during 1983-1990 and its indirect successor NPSNET developed by The NPSNET Research Group developed during the late 1980s and throughout 1990s are two of the best known such applications [6]. Also educational virtual world environments and collaborative workspaces are gaining increasing interest. Virtual world environments are applicable also for modeling, building, and structural design for designers, artists, architects, engineers, and healthcare professionals [16, 17].

Virtual world - real life interaction and other augmented reality applications are becoming more popular. For example, integration of the X10 technology and sensor networks has been implemented [18]. Virtual worlds can be also used for visualization of data [19]. Also, existing online virtual worlds where users interact in somewhat realistic manner provide an intriguing platform for research in the areas of social, behavioral, economic, and human-centered computer sciences [20].

Real life meetings gathering people from all around the world are expensive and cumbersome to arrange. Also, high quality real-time multi-user video conferences set high demand for the network bandwidth and requires very small communication latency. More importantly, simply displaying faces of many people on the screen makes the meeting hard to follow, instead of having clear user presences (avatars) and interacting in more human-like ways (moving around, gathering people around different areas, and so on). Using virtual worlds as the meeting area, it's possible for the participants from all around the world to interact efficiently utilizing instant messaging, VoIP conversation, and sharing and editing documents concurrently [21].

1.3. Structure of the Thesis

A brief history and evolution of virtual worlds and their recent trends and possible future directions is presented in Chapter 2. Chapter 3 introduces the key principles of networked 3-D multi-user virtual world systems. The design choices made in these key areas lay the foundation for opportunities and challenges regarding the scalability of a virtual platform. In Chapter 4, virtual world systems' scalability challenges are presented alongside the commonly used solutions for these problems.

An existing virtual world platform is inspected and used as a testbed for scalability improvements in Chapter 5. The aim of the thesis is to demonstrate that taking a practical approach and optimizing parts of a system locally instead of jumping right into building complex platform is rewarding. Taking into consideration the current state of the inspected platform, it can be seen that when considering hundreds or thousands of concurrent users the potential server-side communication hotspot can be alleviated using an interest management technique.

The experiments and their results are presented in Chapter 6. Possible future work regarding the interest management implementation and experienced problems in the test system are discussed in Chapter 7. Chapter 8 contains a brief summary of this thesis.

2. A BRIEF HISTORY OF VIRTUAL WORLDS

The origins of virtual worlds can be tracked down to the early 1970s fantasy role-playing board games, most notably Dungeons & Dragons (Figure 1), of which the original installment was published in January 1974, by Tactical Studies Rules, Inc. In Dungeons & Dragons, one player acts as a referee, a Dungeon Master (also known as Game Master), who designs the world and the overall storyline, and describes the occurring events. [22]

Figure 1. Dungeons & Dragons board game.

Dungeons & Dragons established many of the essential aspects of virtual worlds to emerge: the player controlled a single character, a representative of the player in the world, which was chosen from the range of available character classes and races, whose properties developed throughout the game in the shape of experience points and other metrics. The board represented the world which consisted of a grid of square blocks – a feature that would be the standard for virtual worlds for many years to come.

The virtual worlds as we know them today are originated from multi-user dungeons (MUDs), multiplayer real-time virtual worlds described originally entirely in text. MUDs were heavily influenced by text adventure computer games, a very important genre of computer games at the time, and role-playing board games. MUDs and virtual worlds are close to tabletop role-playing games, but in their case the referee is the computer and creator and developer of the MUD.

2.1. Early Alphanumeric Multi-User Dungeons

The first MUD (named simply as "MUD") was developed by Roy Trubshaw on a DecSystem-10 mainframe at the Essex University, England, in 1978 [5]. The early MUDs were text-based, meaning that the environment and events occurring were described using text instead of images. Later on, along with the introduction of computer graphics, the MUDs were categorized as text MUDs and graphical MUDs. The first MUD engine could have a maximum of 36 players at once, as the

DecSystem-10 mainframe used a 36-bit word and Trubshaw used one bit per player for internal reference. [5]

In a text MUD (Figure 2) the user interface is implemented using solely alphanumeric characters. The players explore and interact in the world by using simple text commands, or by choosing from a predefined action set. Even with a very rudimentary interface, a sense of immersion can be achieved by compelling storytelling and the help of the player's vivid imagination. The worlds of MUDs were, and still are, strongly inspired by the fantasy literature, most notably Lord of the Rings by J.R.R. Tolkien, and fantasy role-playing game Dungeons & Dragons [5].

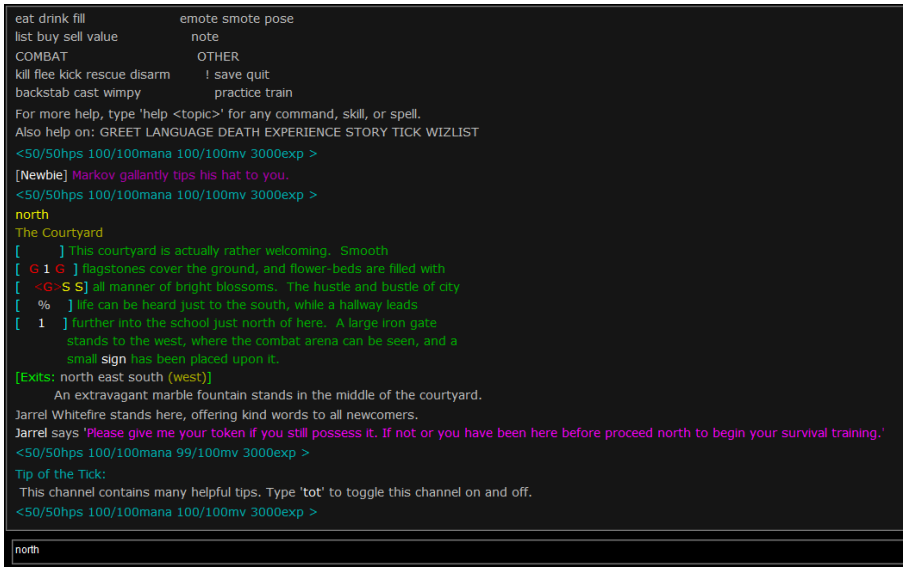

Figure 2. A Text MUD (screenshot of Adventures Unlimited).

In 1981, a MUD titled Island of Kesmai introduced a graphical aspect – but not computer graphics per se - to the MUD genre: Island of Kesmai used different ASCII characters to symbolize the user's character, monsters, walls and other entities on the screen from a bird's-eye-view. This technique was used for the first time in computer game Rogue (Figure 3) in 1980. In this kind of MUD, the level of interaction wasn't typically as high as in the purely text-oriented MUDs, as the graphical presentation took the majority of the screen space, and hence limited the user interface. The worlds of MUDs consisted of square-shaped rooms. Users and objects in one room, or another logically partitioned space, could not interact directly with users in other rooms – a characteristic that would prevail for a surprisingly long time. [5]

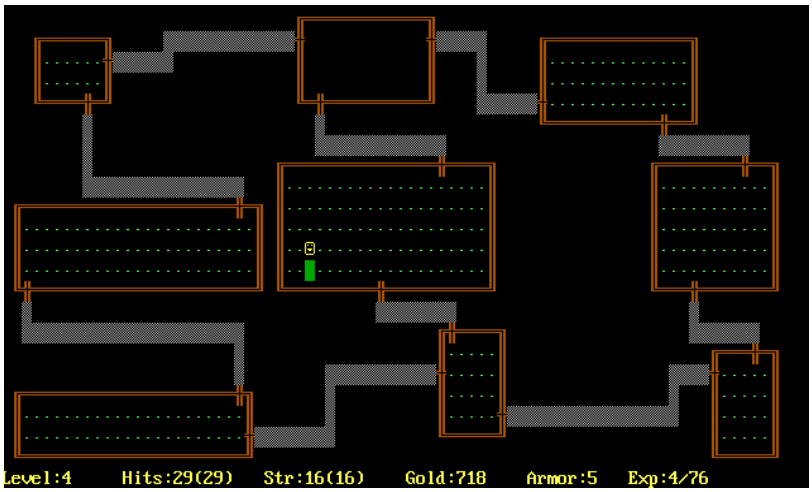

Figure 3. Rogue on a color PC.

2.2. Graphical 2-D Multi-User Dungeons

The first graphical MUD and virtual world by definition (persistent and shared world), Avatar, was released in late 1979. Avatar employed a first-person point of view and it was developed on the Programmed Logic for Automatic Teaching Operations (PLATO) system. Despite the advanced networking and graphic capabilities provided by the PLATO system, the virtual worlds developed on it had very little effect on virtual worlds in general, as the strengths of the system didn't become available for home users until 15 years later. [5]

Lucasfilm's (LucasArts since 1990) Habitat (Figure 4) for the Commodore 64 released in 1986 is considered to be an influential virtual world of the early days. Habitat used a side-view, a style that LucasArts utilized in the future for many of its puzzle point-and-click games that are regarded nowadays as classics. Habitat on-line services were provided by Quantum Link, which later on became America Online (known as AOL Inc. today). In Habitat users could move around, interact with objects, gesture and chat. [5]

Figure 4. Lucasfilm's Habitat.

The early two-dimensional (2-D) virtual worlds represented the world, built from blocks, players and other entities as flat sprites (two-dimensional images) rendered

using a bird's eye view. Separate areas in the screen were used for showing other information, such as chat and inventory (Figure 5). The block structure is an efficient, but not that flexible, way to implement a virtual world: world creation is simple and its server bandwidth requirement is somewhat light with a satisfactory level of interaction between players. [5]

Figure 5. A typical graphic MUD (screenshot of Wyvern).

TinyMUD released in 1989 was one of the first primarily social virtual worlds allowing user-created content - locations and objects - but contained very little game-like functionality. LPMUD, developed by Lars Pensjö of the University of Gothenburg, Sweden in 1989, introduced an in-game programming language LPC, which made it possible for users to add objects and functionality to the game at run-time. At around the same time, DikuMUD, which was released in 1990, took the completely different route, and instead hard-coded as much functionality as possible. However, the DikuMUD internals were designed well and a reasonable programmer could use its source code to create one's own world. The DikuMUD codebase was used as a basis for creating many other MUDs. [5]

The transformation from alphanumeric representation to more graphically appealing form did not change the fundamentals of the MUDs: they continued to be faithful adaptations of the early board game principles with the square-shaped rooms.

2.3. 2.5-D Virtual Worlds

By utilizing an isometric view, fixing a camera at an angle other than directly overhead, a sense of a 3-D world can be achieve despite the world and objects being in fact mere 2-D sprites. This technique is commonly referred as two-and-a-half-dimensional (2.5-D). [5]

Ultima Online, developed by Origin Systems, Inc. and launched in 1997, is widely regarded as one of the most important milestones in the history of virtual worlds. It is said that Ultima Online was the first virtual world to actually realize the potential of virtual worlds. Ultima Online gained huge popularity fast, over 100 000 subscribers within one year. Ultima Online still continues to exist, and it has a faithful fan base. [5]

One of the first large scale virtual worlds was The Kingdom of the Winds, a 2.5-D MMORPG developed by Nexon Co. Ltd., which managed to have an impressive number of 12 263 concurrent users in 1999 using distributed server technology [5].

Another notable 2.5-D virtual world is Habbo Hotel (Figure 6), founded in 2000, that is a web browser based virtual world implemented by using Adobe Flash (formerly called Macromedia Flash), a rich Internet application (RIA) technique. Habbo Hotel has over 200 million registered users in 32 countries and 8 million active users a day. The revenue is gathered by trading real money for Habbo Credits, which can be used to buy virtual furniture and other content [23].

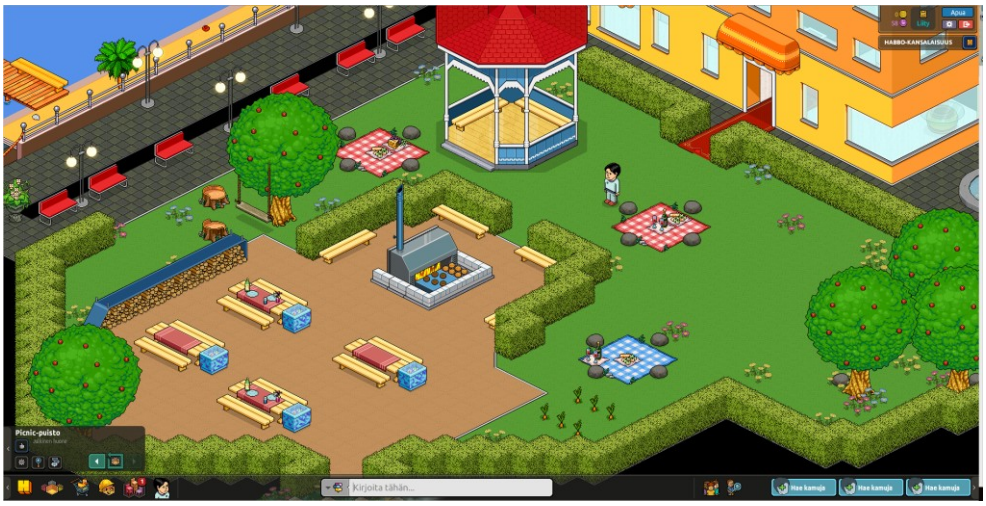

Figure 6. Habblo Hotel (screenshot).

2.4. Introduction of 3-D Virtual Worlds

Meridian 59, released 1996, can be considered as the pioneer of 3-D virtual worlds, and it had a great impact on MMORPGs to come. It utilized the first-person view and it can be considered being the first real 3-D MUD. However, Meridian 59 was brought down by its soon-to-follow competitors Ultima Online and EverQuest and ended up being a commercial failure. [5]

EverQuest (Figure 7), released in early 1999, was essentially a DikuMUD with a 3-D graphical client. It used similar a first-person 3-D view as Meridian 59, but allowed a freely movable camera. EverQuest's world was divided into multiple zones each running on its own server. Notable delay when moving from one zone to another was noticed. [5]

In the vein of EverQuest a virtual world boom was seen. Amongst most notable followers was Asheron's Call, launched in late 1999, which introduced both inventive ideas earlier seen only in text-based MUDs and technical innovations: a single seamless world running on multiple servers but using dynamic load balancing. The first virtual world seen on console platforms was Phantasy Star Online in 2000 on the Sega Dreamcast. [5]

Figure 7. EverQuest.

Second Life (Figure 8) was launched in the summer of 2003 and it is one of the most widely known 3-D virtual world phenomenon of the past decade. Second Life provides realistic social interaction with other users – even virtual weddings of the users are no rarity. The users can trade real money for Linden Dollars which can be used for uploading content (textures, sounds and animations allowed), purchasing land, and buying and trading content with other users. In-world content can be created freely and without charge only in dedicated sandbox areas. Users can program in-world functionality using the Linden Scripting Language (LSL) in a sandboxed (secure and controlled) environment.


Figure 8. An ongoing meeting at Second Life.

World of Warcraft (Figure 9) is the world's most-subscribed MMORPG with over 12 million active subscribers in October 2010 [24]. The fantasy world setting of World of Warcraft consists of two sides, Alliance and Horde, battling against each

other. Both sides have their own races. Players gain experience points and levels by collecting items, performing quests and battling with their opponents.

Figure 9. World of Warcraft.

2.5. Immersive Virtual Environments

In 1956 Morton Heilig developed a simulator called Sensorama which presented passive simulation of an imaginary motorcycle ride at Manhattan. A sense of immersion was achieved by using smells of exhaust gas and the city, wind in the hair, stereo sound and vibrations of the seat. [25]

The first Cave Automatic Virtual Environment (CAVE) was introduced in 1992 [26]. In Figure 10, a user is experiencing a virtual world using CAVE room and stereoscopic rendering. Whereas in the early days of virtual reality head-mounted displays were used to close out the surrounding environment, with CAVE technology the whole room functions as an immersive virtual environment where the user can literally stand inside. As traditional input devices, such as keyboard and mouse, aren't really sufficient for an immersive controlling experience in CAVE environments other devices have been researched for the use.

Figure 10. A CAVE room with stereoscopic rendering.

During the last couple of years all of the major console companies have each introduced their own peripheral devices to achieve a more engaging, involving and immersive gaming experience, for example PlayStation Move and Kinect for Xbox 360 [27, 28]. Microsoft released a non-commercial Kinect for Windows software development kit (SDK) for researchers and enthusiasts to use in the spring of 2011 [29]. The Kinect device has already been used for many experiments by both researchers and hobbyists. Also new companies focusing on virtual reality are surfacing, for example Oculus VR, which develops the Oculus Rift virtual reality headset [30].

2.6. Standardization

The most notable standard made within the virtual world domain is the Virtual Reality Modeling Language (VRML) (ISO/IEC 14772-1:1997) standardized in 1995 and readjusted in 1997 in the form of VRML97. VRML is a file format for representing interactive 3-D vector graphics.

By now VRML is already superseded by its successor Extensible 3D Graphics (X3D) (ISO/IEC 19775/19776/19777), standardized in 2004. X3D adds extensions to VRML and makes it possible to encode the data using various syntaxes, such as Extensible Markup Language (XML).

X3D seems, however, not to be that commonly used compared to for example the similar and more widely adopted Collaborative Design Activity (COLLADA), managed by the Khronos Group. Many common web standards and protocols, such as HTTP, XML, HTML, CSS, and ECMAScript (JavaScript) are used as is in many different virtual worlds.

2.7. Current State and Future Directions

The fundamental features of modern virtual worlds are still greatly the same as in the early days: the focus is on a vivid and immersive world with rich content and multi-

user experience with social interaction between the users. The scope of modern-day virtual worlds can vary greatly in both implementation technology and complexity of the content from simple 2-D or 2.5-D chat rooms running within a web browser to full-blown standalone applications utilizing the most cutting edge 3-D rendering techniques.

The constantly increasing amount of computing resources helps to develop larger, richer and more impressive 3-D virtual worlds with an increasing amount of concurrent users. However, the underlying resources themselves don't automatically guarantee a smooth and seamless virtual world experience and effective usage of the resources. Thus scalable networking and software architecture design are more essential than ever, especially when considering a large scale and general use 3-D multi-user virtual world system.

Immersive environments, virtual reality, and augmented reality applications are gaining more popularity. Home users are able to avail themselves of immersive virtual reality enabling technologies, such as machine vision, gyroscopes and inertial sensors, at affordable prices. Also the recent renaissance of stereoscopic and 3-D display techniques has for its own part increased the general interest in virtual worlds and virtual reality.

With ubiquitous and increasing variety of client hardware and software, different users are able to access the same virtual worlds using different client platforms. These include the popular desktop operating systems Windows, OS X, and Linux, and also mobile systems, such as Android and iOS. Even pure web browser based clients have been emerging, for example the Meshmoon WebRocket, aided by the latest web technology standards such as HTML5, WebGL, and WebSocket [31].

The increasing amount of popular operating systems has steered the virtual world client software architecture development towards cross-platform solutions. Also, in addition to the traditional closed and commercial solutions, open source virtual world platforms are gaining more momentum and interest.

Another notable trend is the convergence of traditional 2-D document-centric web technologies and real-time 3-D content. Integration of virtual worlds and social media and other web services is relatively simple via the services' application programming interfaces (APIs). Also the term 3-D internet has been used to describe this convergence [32].

However, interoperability between different virtual world platforms in general is currently largely lacking, and the interoperability consists mostly of asset conversion tools and other utilities. Typically each virtual world platform implements its own content formats and wire-protocol, and uses some traditional web technologies on top of that. Some new attempts at standardization have been happening, for example the FI-WARE platform started by the European Commission [33].

3. 3-D MULTI-USER VIRTUAL WORLD SYSTEM PRINCIPLES

A typical 3-D multi-user virtual world system consists of at least a seemingly unified environment, to which the users have views, and in which the users can control their presences and have opportunities to interact with the environment and other users, and manipulate the shared data within the rules of the virtual world system (Figure 11). The rules consist of application logic, physics simulation, user rights, and so on. Usually a single avatar, or other controllable entity, is assigned for the user in order to provide a means of presence in the world. The degree and nature of manipulation and interaction possibilities are dependent on the type and purpose of the world.

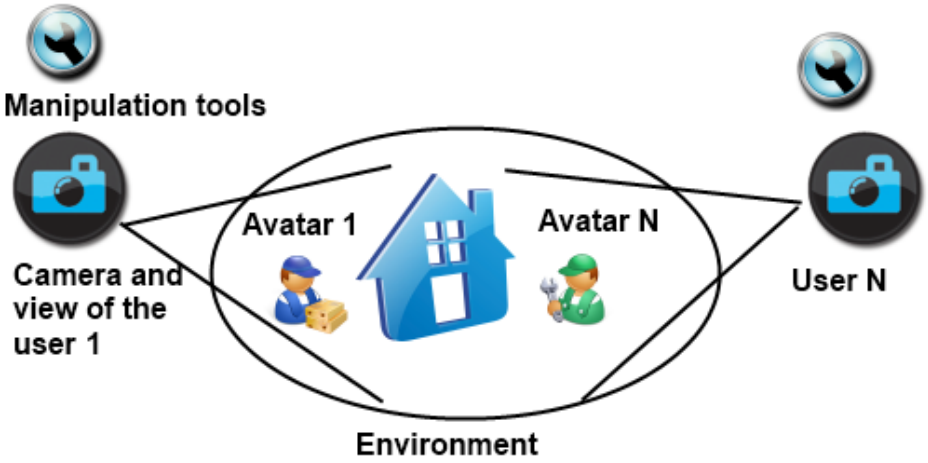

Figure 11. Conceptual virtual world.

3.1. Virtual World System Architecture

Virtual worlds and networked virtual environments in general can be seen as multiple traditional software types combined into one: virtual worlds contain elements from distributed systems, graphical applications, and interactive applications [6]. The design is often complicated by the fact that the system has to usually work and interact with existing application services, such as database systems, user authentication and other transactions systems, and storage systems [6]. Of course the virtual world system can be designed as a single application, but a system consisting of several interchangeable subsystems can be seen as more extensible and less vulnerable. A general overview of the virtual world system architecture is presented in Figure 12.

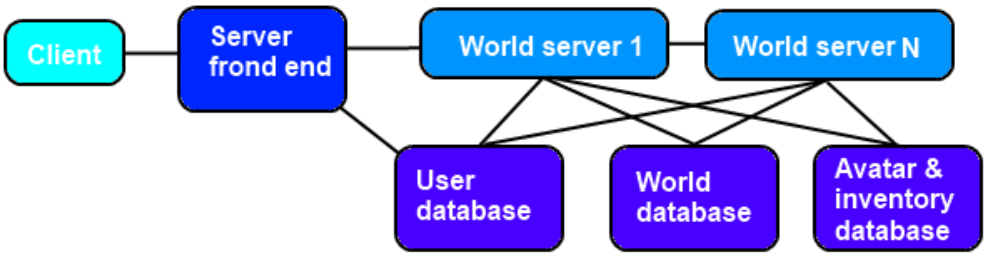

Figure 12. Overall architecture of a virtual world system.

Virtual worlds are networked applications. Usually the environment is hosted on one or more server instances, depending on the chosen network and communication architecture. The users connect to the virtual world server via the underlying communication path and network protocols by using their client software.

The events and states of the world are replicated, i.e. communicated, between the participants. The server might have different front ends for different types of client software: one for native PC clients, one for lighter mobile devices and one for web browsers and so on. The purpose of the front end is to provide access to the same shared data regardless of the client's operating system and hardware capabilities.

Virtual world's content and object information are stored often, but not always, on separate database servers. Different database servers can be used for different types of data. For example, separate avatar and inventory databases can be used in order to make the transition between servers fluent while maintaining the virtual identity and its belongings.

Essentially, most of the modern virtual world engines are networked 3-D game engines, but on the other hand, especially general purpose virtual worlds can be seen similar to content management systems (CMS). When designing a pure game, or any large piece of software, the developers usually have a known specification of the required functionality. However, reimplementing a game engine or any other substantial large scale software over and over again from scratch is an expensive task. Hence, having a modular, dynamic, and extensible system provides a foundation for building long-living solutions for various applications and uses. This modularity and dynamic extensibility of the software platform adds its own challenges to virtual world engine design and implementation.

3.1.1. Implementation Issues

The functionality, computational tasks, data storage, and so on of virtual worlds' can be implemented in multiple machines and in multiple different configurations. Fully distributed and fully centralized systems can be seen as the two extreme approaches to the virtual world system implementation, but most commonly the solution is something in-between.

The responsibilities and tasks between the client and the server applications can be broken down into a set of high-level components. The division of components varies between different network architectures. The typical high-level components and their division in traditional client-server architecture are presented in Figure 13. In the peer-to-peer architecture, every participant is responsible more or less for all of the components

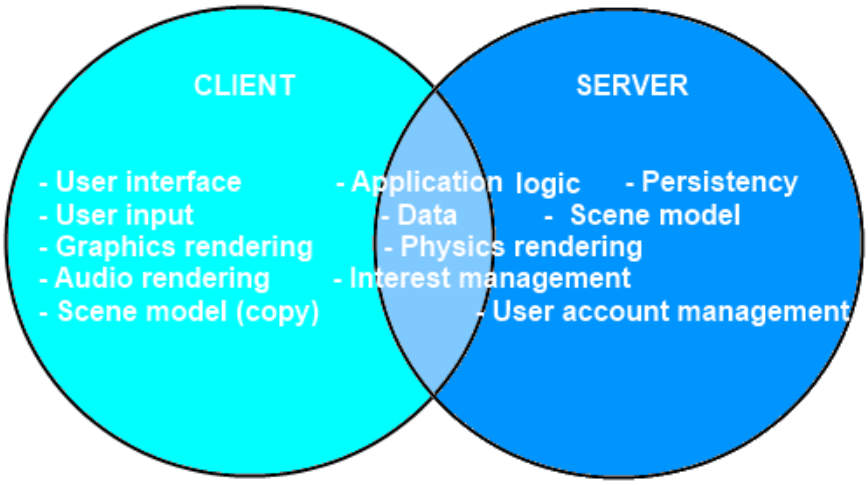

Figure 13. Typical high-level components of a virtual world system.

The server is authoritative for the scene model, and the client stores a full or a partial copy of it in order to be able to represent the world. Data in the figure refers to the source of the data that is needed for the virtual world, including types such as 3-D meshes, audio, scripts, and so on.

Naturally the client needs a copy of most of the data in order to present the scene for the user. The data can be retrieved from different local or networked sources. The persistency of the scene is handled by the server, meaning the world state is typically stored in a database. User account management and authentication are also typical server-only tasks.

Traditionally the input refers to the actions that a user enters via some kind of user interface: the interface can be an alphanumeric command line interface or a graphical user interface using the typical input devices, mouse and keyboard. Input can also be from a wider range of devices including touch-sensitive devices, audio through the microphone, video feed from web camera, and so on. Typically the retrieved input is translated into events (“W key pressed”) or actions (“move forward”) for the use of the application logic locally, remotely, or both.

Application-specific logic and behavior is typically achieved by using some kind of dynamic scripting language. The execution is typically both a client and server-side task. The graphics and audio rendering tasks are virtually always performed at the client, whereas the server usually takes care of most of the physics rendering (simulation). The graphics rendering is a substantial workload so there's no point in performing it at the server as it has very little use for the information. Audio rendering means real time mixing of both non-spatial and spatial audio sources according to the listener's (typically avatar or camera) position.

Interest management is an optimization mechanism for the bandwidth usage (server-side) and graphics rendering (client-side). Interest management is optional, but a necessity in worlds of many thousands of objects, if desiring adequate performance and real-time experience.

3.2. Scene Model

The virtual world environment typically needs to persist. Typically this is achieved by using a database which holds a copy of the current state of the world. This *world state* is described using a *scene model*, a high-level description of the contents of the virtual world environment, which is usually held in is some kind of tree structure. The details of different scene model implementations can vary greatly, but two commonly used architectures for describing the world object structure can be identified: object inheritance and component aggregation.

3.2.1. Object Inheritance

The traditional design of a world object structure is the object-oriented architecture in which new object types, `Camera`, `Avatar` or `Terrain` for example, are created by inheriting from appropriate parent object types, for example `Renderable` and `Controllable`. This creates a deep hierarchy where specializations of generic objects reside deeper in the hierarchy (Figure 14).

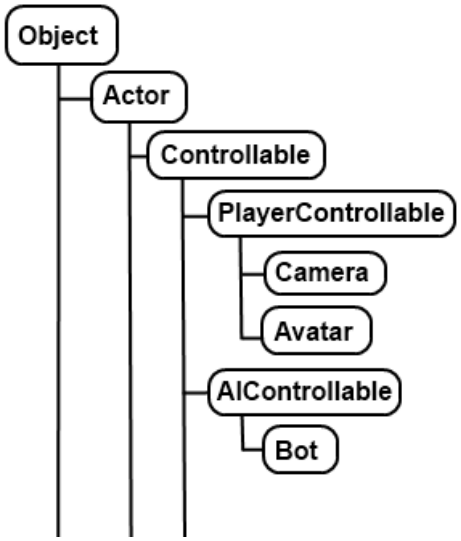

Figure 14. An example of deep object hierarchy.

The world object model based on object inheritance is something that is nowadays considered somewhat old-fashioned. The complexity of the system grows as the dependencies between different levels of the hierarchy become heavier. Without careful design of object interfaces, the specialized objects' interfaces can be bloated with unwanted functionality leaking from the base objects. As deeper and more complex hierarchies and dependencies are formed, the model becomes heavy and difficult to both maintain and handle.

The aforementioned matters can also make modifying existing behavior rather difficult which makes extensibility, especially at run-time, difficult and problematic. As the functionality and properties of an object are usually fixed and specialized, it makes designing the set of network protocol messages straightforward. However, changes to the object model usually cause protocol and binary incompatibilities between different host versions and break the interoperability.

3.2.2. Component Aggregation

The other common scene model architecture chooses aggregation instead of inheritance. This method is considered more modern, generic, and extensible. This model is used in many MMOs based on the observations of the author. In a component-based scene model, an object, often referred to as an *entity*, may be a collection of components. This model is usually called an *entity-component* model or system (Figure 15). The model can also be implemented as pure aggregation where the concrete concept of an object does not necessarily even exist.

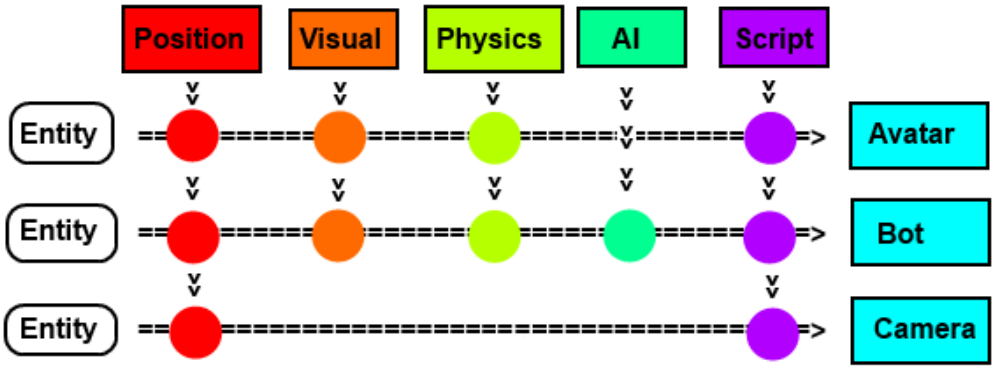

Figure 15. Component aggregation example.

In component-based systems, the components can contain data, or both data and behavior. For example, a `Position` component can simply have position, orientation and scale attributes and nothing else, or the component can contain also behavior: the component itself reacts to the updates of the aforementioned attributes and updates them accordingly to the scene graph. If the component is pure data, some subsystem of the application implements the behavior related to the specific component.

Individual components can be examined separately and desired object behavior can be achieved by combining the needed components and nothing else. Similar behavior can easily be added to markedly different kinds of objects and different types of behaviors are clearly decoupled from each other. This decoupling leads to reduced complexity and to a more easily comprehensible overall system. Component-based systems are also easily expandable and adding a new component type usually requires no changes at all to other components.

On the other hand, a component-based system may introduce some undesired overhead due to possible inter-component dependencies as a specific component type might need to check for the presence of other component types. For example for a `Render` component, *what* to render, is usually tightly tied to the `Position` component, *where* to render. On the plus side, typically in an entity-component model the data updates are done per component, not per entity, which generally leads to a simpler network message set and lighter object state update packets meaning better scalability.

3.2.3. Scene Graph

The scene model is typically accompanied by a scene graph which is a representation of the spatial 3-D information for the use of the graphics renderer. A scene graph consists of a node hierarchy forming parent-child relationships between the nodes. Typically a node contains the necessary spatial information – position, orientation and scale – and a reference to the actual visible, renderable, data. The node's properties relative to the parent node are referred to as the *local* information, whereas the absolute values measured from the origin of the world, i.e. the root node, are referred to as the *world* information.

Alternatively the scene graph can be integrated into the scene model. Other important application subsystems, such as physics and audio rendering, have their own internal data structures. Visible, physical and audible content and behavior are rendered accordingly to the information provided by the scene model, scene graph and the actual data.

3.2.4. Data distribution

Choosing the location of the data is a critical decision as it affects the scale, communication requirements and reliability of the virtual environment data. Macedonia et al. have identified three data distribution models: the shared centralized world, the homogeneous replicated world and the partially replicated (or distributed) world. [34]

In a **shared centralized world** one database is shared by all users and all of the data resides on a central server. When the user wants to modify an object state, the client sends a modification request to the server. The server processes the request and depending on the credentials of the user, or some other criteria, the server either accepts or denies the request. After this, the server performs the necessary processing and sends the new object state to all users.

This model ensures consistency and thus avoids data replication, communicating state changes or events between peers or hosts. Consistency might come with the price of possible interaction latency: with a high number of users the server might become bottlenecked. [35]

In a **homogeneous replicated world**, the world state of all users is initialized with a common database that contains all the necessary information for the virtual environment. The data may be present already on the user, as is done in many games, or if not, it is replicated from the server or from another location to the new users. Data modifications are executed locally, but additional synchronization mechanism can be used. [35]

The advantage of the model is that the number and size of sent messages in the network is small as only update messages and some special events, such as physics collisions, are sent. Also the model has low interaction latency as modifications are performed locally before being sent to other participants. [35]

On the negative side, state inconsistencies are possible as a result of packet delay or loss. As modifications are performed first locally, a concurrency management mechanism is needed for checking conflicts. With large amounts of data the database on each node is also large. Furthermore, this model is not flexible if users want to add new objects outside of the initial database and hence nor really suitable for general use virtual worlds. [35]

In order to overcome the drawbacks of the previous two data distribution modes, several hybrid solutions have been used. These solutions, referred to as partially replicated world, or distributed world, contain characteristics from both of the aforementioned modes and both data and its updates are distributed to various extent. [35]

3.3. Communication and Network Architecture

The communication and network architecture is one of the most crucial design issues when designing a new virtual world system as both offer opportunities and challenges to be faced with vary with each option. The chosen architecture affects the scalability, extensibility, and security of the overall virtual world system.

The two dominant and most commonly used network architectures in virtual worlds and suchlike environments are the **client-server** (or **client-multiple server**) and **client-multiple coordinated servers** architecture. The **peer-to-peer** (P2P) architecture has been researched a lot, but lacks commercial and successful implementations. Regardless of the chosen network architecture, the communication protocol must be chosen and implemented wisely as a naive and unoptimized communication protocol can hinder both performance and scalability of the system.

The client-server model and its variations are the most common techniques used in commercial products, whereas the P2P model has mostly been a research subject and only used by academic researchers. This makes the case for the reason that integrity and authorship of the world state and data required by commercial products is much easier to achieve in a scenario where only single point of authorship (server) exists, whereas in P2P architecture each host is authoritative for the state changes. Also, hybrid architectures that combine elements from both of these architectures exist, for example the peer-server model [36].

3.3.1. Client-Server Model

In the classic client-server model, represented in Figure 16, each client communicates with a single server instance which handles all of the network traffic. This model is especially common in popular multiplayer computer games as the server has total control and is authoritative for all simulation which prevents cheating and other malicious usage.

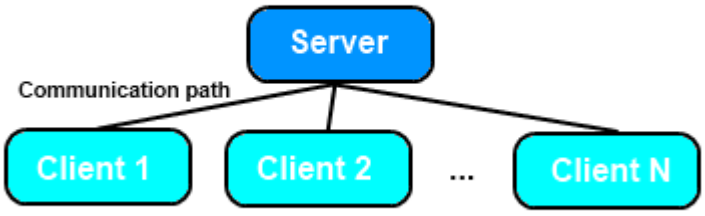

Figure 16. The client-server architecture.

Time synchronization, world state consistency and concurrency control are relatively easy to implement in a client-server system. Usually this architecture is not considered very scalable as the server's bandwidth or computational resources easily become the bottlenecks of the system with high numbers of concurrent users leading to interaction latency and jitter. [37]

The multiple server architecture (Figure 17) is very similar to the regular client-server architecture with the exception that several independent servers exist with no or very little interaction between each other.

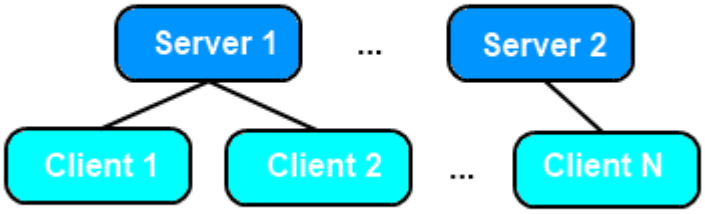

Figure 17. The multiple server architecture.

Multiple server architecture can be utilized in different ways. The most common case is to use the other servers as backups. Another way to utilize multiple servers is to divide the responsibilities: one acts as the world server, one as the login and user database server, and so forth. This architecture provides more scalability and reliability than the client-server model, but only in a way that does not allow the users on different servers interacting with each other. [37]

3.3.2. Coordinated Multiple Servers Model

The coordinated multiple servers architecture (Figure 18) is the dominant architecture amongst modern commercial virtual worlds, such as Second Life and World of Warcraft. Coordinated multiple servers architecture adds server-server communication to the client-server model. Usually one server instance represents a single geographical area or region where the user can interact. Users can move between regions as server-server communication takes care of the handshake procedures.

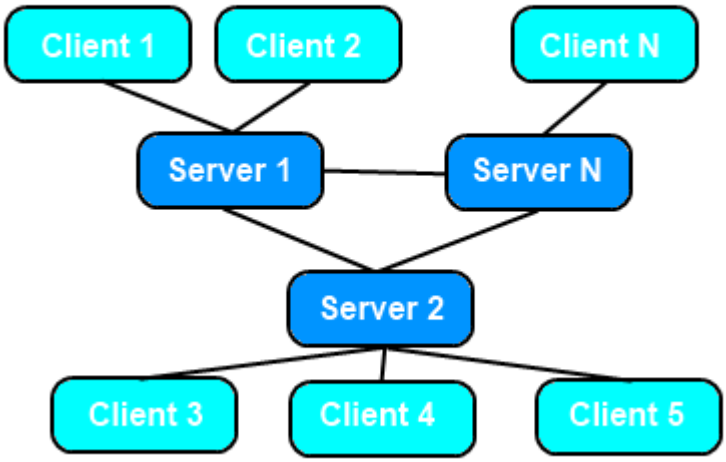

Figure 18. The multiple coordinated servers architecture.

This architecture increases scalability and reliability. More users can exist simultaneously in the virtual environment even though they cannot see or interact with each other in most cases. This architecture provides reliability in a sense that if the server is down users can always exists on other regions**.**

The interactibility level stays pretty much the same compared to the traditional client-server model, as usually the maximum number of concurrent users per server is restricted and if not, the performance of the server will become very poor after a threshold point. This diminishes the overall user experience and sense of large and continuous environment. Also the view from the current region to the neighboring regions is usually limited or nonexistent.

3.3.3. Peer-to-Peer Model

In the P2P model, represented in Figure 19, a server instance per se doesn't exist at all, and all hosts communicate directly with each other. Usually each host holds a full or partial copy of the current state and is authoritative for state changes. It's also possible to use a server instance as the source for the initial state when there are no hosts and the world must be set up. [37]

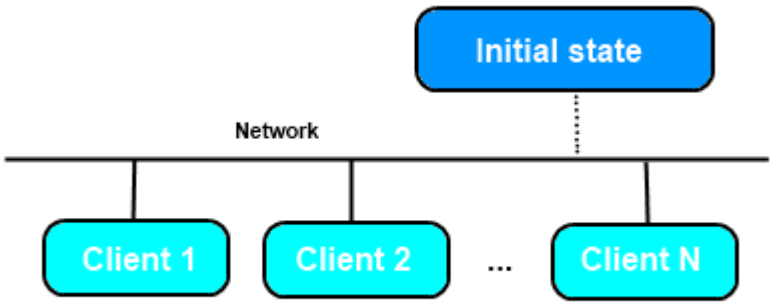

Figure 19. The peer-to-peer architecture.

The biggest drawback of the P2P model is that the hosts' bandwidth usage is heavy. Also, time synchronization and consistency is harder to achieve and maintain, as it's difficult to contact all users at the same time. For this reason, a global time keeping mechanism, f. ex. Network Time Protocol (NTP) or reliable time-stamp update messages at minimum, is required. P2P model could be more efficient when implemented using multicasting, but available support for multicast is poor, as covered later on. [37]

3.3.4. Comparison of Communication Architectures

Comparison of commonly used network architectures in terms of scalability, reliability, interactibility, costs (software, hardware, maintenance), and security, is presented in Table 1.

Table 1. Comparison of commonly used network architectures.

	Client-server	Peer-to-peer	Client-multiple servers	Client-coordinated multiple servers
Scalability	-	--	-	++
Reliability	--	++	+	++
Interactibility	+	++	++	++
Costs	++	++	-	--
Security	++	--	++	++

The client-server and client-coordinated multiple servers are currently the most popular architectures in virtual world systems. These architectures provide satisfactory interactibility – consistency and concurrency of interaction with multiple users within the same world. In both architectures the servers are fully authoritative for the world's state which prevents cheating and makes malicious usage more difficult.

While the client-server architecture is relatively straightforward to implement, optimizing it for hundreds or thousands of concurrent users is a major obstacle. With the client-multiple coordinated servers architecture, the world is partitioned into smaller regions which keeps the computational burden of a single server instance lower. However, this approach often reduces the immersion and sensation of a continuous world as inter-region communication is very limited or not possible at all in most cases.

In both architectures, the server quickly becomes the bottleneck, as computational complexity and bandwidth usage grow leading to greater interaction latency and restricted amount of concurrent users. Regardless of the architecture, the protocol must be efficient on both application and transportation level. Efficient and intelligent interest management and other optimizations are needed for increasing the user and content count while retaining sufficient interactibility level.

Reliability is an issue with the client-server architecture as only one server exists. Client-multiple servers and client-coordinated multiple servers provide the solution for this problem. In the P2P architecture security and consistency are significant challenges and a reliable time synchronization mechanism is needed. P2P is problematic as a virtual world communication model as the virtual world host typically wants to be in control and authoritative for the content and its changes.

3.4. Communication Protocols

Choosing and implementing a virtual world communication protocol is about finding a balance between the need for efficiency and good transmission semantics. A network protocol describes the set of rules that applications use to communicate with each other.

A protocol can be broken into three components: packet format, packet semantics, and error behavior. Packet format defines what information and data a packet contains and packet semantics describes the meaning of each packet. Error behavior defines rules that must be applied when an improperly formatted packet is received or no packet is received at all. [6]

Each of these components can be implemented very differently between different virtual world systems, but naturally common traits can be identified. In addition to the communication protocol, the actual data transfer protocol must be chosen.

Virtual world systems define their communication protocol at the *transport* and *application* layer. The Internet Protocol (IP) is used virtually always as the underlying *Internet* layer protocol for providing the end-to-end connectivity between hosts. The message routing schemes typically used in virtual worlds are unicast (one-to-one), broadcast, (one-to-many), and multicast (one-to-unique). Comparison of common protocols used in virtual worlds is represented in Table 2 in terms of scalability, interactibility, header size (the size of inevitable payload when using the particular protocol) and interoperability.

Table 2. Comparison of network protocols.

	TCP/WebScoket	UDP	Reliable UDP(*)	IP Multicast	IP Broadcast
Scalability	--	++	++	++	++
Interactibility	++	--	++	--	--
Header size	--	+++	++	+++	+++
Interoperability	+++	++	++	---	---

*application-specific, details vary

The most commonly used transmission protocols are Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and reliable UDP. Platforms aiming for web browser usage also utilize the WebSocket protocol, which is essentially just a layer on top of TCP sharing the same fundamental characteristics [38].

TCP provides reliable and in-order point-to-point connection using acknowledgment and retransmission and stream-based semantics [6]. TCP supports only unicast. TCP is transmission and connection-oriented protocol and has congestion and flow controls. Because of the added functionality TCP must transmit more information in its header and it has a higher bandwidth overhead. If operating on a bad or slow network connection causing lots of packet loss, the fact that everything is reliable means that packet drops can cause heavy congestion which makes TCP less than optimal for real-time streaming – there's no point to resend data that is already outdated. The networking hardware is usually optimized for TCP.

UDP provides unreliable lightweight data transmission and packet-based data semantics. UDP supports IP broadcast, multicast and unicast. UDP is a connectionless protocol at its core. Because of missing features compared to TCP, UDP has a very light header. UDP is almost ubiquitous but handled as a "second class citizen" by firewalls, routers, and other networking hardware. UDP has immediate delivery but no delivery order guarantee, and corruption of data is possible as no checksum is used. [6]

Due to the shortcomings of the default UDP, most virtual world systems implement their own reliability and ordering mechanisms on top of UDP on the *application* layer. This extended protocol is typically called **reliable UDP**[1]. In reliable UDP the plain connectionless UDP is extended to be connection-oriented: an end-to-end connection is established before any data is sent. Also, reliable UDP typically provides optional reliability and order guarantees for the packet delivery. Reliable UDP introduces some overhead to the packet size due to the increased information carried in the packets.

Also **IP multicast** and **IP broadcast**, that allow single transmission to be delivered to all (broadcast) or multiple (multicast) hosts using UDP, are quite often mentioned in the virtual world research, but they are problematic and even unusable for commercial purposes: IP broadcast cannot be used outside of LAN and IP multicast requires an arcane multicast backbone (MBone) for wide area network (WAN) or Internet usage [6].

Also, in the past a few virtual reality dedicated protocols, such as Virtual Reality Transfer Protocol (VRTP) and Distributed Worlds Transfer Protocol (DWTP), have been proposed [39, 40]. However both of these protocols seem to be missing from any recent implementations or utilizing applications. The Stream Control Transmission Protocol (SCTP) is a relatively new protocol defined in 2000. SCTP combines features from TCP and UDP and sounds promising for virtual world usage, but so far its support by networking hardware has been poor.

The choice of network architecture affects the network protocol choice. In the client-server architecture and its variations, the protocol can be chosen freely. However, in the P2P architectures TCP is not really usable as there’s no point in resending potentially already old and obsolete data for which constant resending can congest the bandwidth. UDP and reliable UDP are suitable for all communication architectures.

Also, the nature of the network traffic affects the choice of the network protocol. After the initial scene model and asset replication, the majority of virtual world communication traffic consists of real-time streaming of state updates, movement for example, and TCP is not generally considered suitable for streaming purposes; however, World of Warcraft for example is known to utilize TCP with good results.

[1] Not be confused with the informal standard *transport* layer protocol of the same name.

4. IMPLEMENTATION TECHNIQUES FOR SCALABILITY

When the first MUDs were designed, no-one could anticipate the future scale of their successors. In the early days, the computational resources were very limited, not to mention the available network resources. As the early MUDs had very low-end graphics, the client-side performance was hardly an issue at the time. The emergence of 3-D virtual worlds brought its own challenges both in the areas of the client-side performance and networking. The modern games set the bar high for visual quality continuously and virtual worlds must be able to meet the requirements of the evermore demanding users, while battling with the issues of already high computational complexity.

The creators of MUDs and other early virtual worlds built the systems for one specific use in mind, the game. The current virtual world system architectures are designed to be more generic and extensible for more long-lived and general-purpose application development. When building software architecture for a certain type of application, certain assumptions regarding the software design can be made in order to optimize the system's performance but building a generic system requires making compromises between efficiency and generality.

The design of a modern virtual world system reflects strongly on the scalability and overall performance of the system. Lee et al. have identified five key design issues related to scalable DVEs:

- communication architecture
- interest management,
- concurrency control,
- data replication, and
- load distribution. [36]

Communication architecture, data replication and their general high-level implementations have been discussed earlier. These two are fundamental features of DVE architecture, and cannot be omitted, whereas interest management, concurrency control and load distribution can be seen as optional, but necessary at certain points of development as building blocks when building large-scale virtual worlds.

Many existing systems have proposed solutions to the scalability issues of large DVEs [36]. However, the approaches taken are commonly too application-specific and do not address all of the five key issues. Of the five key design issues, communication architecture, interest management, data replication and synchronization can be seen as important always, whereas concurrency and consistency control is very important in collaborative virtual environments (CVE). Concurrency and consistency control is out of the scope of this thesis.

Scalability problems are not tied only to the number of concurrent users, but also to the computational complexity of the virtual world. At the extreme we have general purpose virtual worlds and collaborative software applications. In general purpose virtual worlds (e.g. Second Life) the world content can be created, edited, and removed somewhat freely. Additionally in such worlds, the amount of user interaction, for example user communication and trading, is typically high. These result in high computational complexity, and hence lower scalability. Typically collaborative software applications, for example videoconferencing, have low computational complexity (no complex world content and user presence) and hence high scalability. Games (e.g. World of Warcraft) can be seen as the middle-ground

with many concurrent players, but within a somewhat static and immutable environment. [41]

Interestingly, the amount of concurrent users that can interact with each other has not been drastically increased during the last couple of decades, with a couple exceptions to the rule. It would appear that the ever-growing complexity of graphical 3-D content, physics simulation, and the amount of data transferred over the wire has grown significantly over the years, all of which take their toll. Whereas 36 concurrent users occupied a single server in the first MUD ever, currently on average less than one hundred users interact with each other in the current popular 3-D virtual worlds or MMORPGs. [5, 42]

4.1. Scalability Challenges

Liu et al. categorize scalability of virtual worlds into three different dimensions:

1. scaling the number of concurrent users interacting with each other,
2. scaling the scene complexity, and
3. the fidelity of user interaction. [7]

In order to resolve growing numbers of concurrent users, many virtual worlds restrict the amount of concurrent users interacting with each other using different techniques. Scaling the scene complexity is especially important in general purpose virtual worlds. Keeping the fidelity of user interaction sustainable requires the system to have sustainable responsiveness, which is achieved using concurrency control mechanisms.

In addition to the aforementioned dimensions, two additional dimensions can be identified:

4. scaling augmented functionality, such as integrated VoIP and web camera feed , and
5. scaling increasing detail (more complex meshes, more and larger textures, etc.)

In this thesis we're interested in (1) and (2). In (4), the issues usually are heavily dependent on the implementations of the third-party libraries that are integrated into the system. Ideally, the streaming of the additional data can be joined to the same data stream that is used for the data replication if wanted. This allows finer control over the overall bandwidth usage per client. Problem (5) is an interesting issue, but out of the scope of this thesis. Efficient handling of (1) is largely dependent on the network protocol efficiency and the server's capabilities.

Although network bandwidth is becoming less of an issue, the server's bandwidth is still somewhat scarce. When considering traffic of hundreds or thousands of client connections the physical limitations of the network hardware layer still yields a notable part of the scalability issues. Server-side computational footprint per user is another important optimization area.

The majority of the virtual world network traffic consists of movement updates: already over 50 % for MUD2, 90 % or more for graphical virtual worlds [5]. In this light, it seems obvious that at the very minimum, the format of the movement packets must be carefully designed. Moving objects can be divided into two categories: non-physical and physical objects. Physical objects in virtual worlds are typically so-called *rigid body* objects. *Soft body* objects are another type of physical

object, but due to the complex nature they are not typically replicated. Dead reckoning and client-side extrapolation are used for optimizing the bandwidth usage of physical objects at the cost of accuracy [6].

When maintaining the distinguishing features of virtual worlds mentioned before Liu et al. have identified three unique simulation requirements specific for scaling virtual worlds [7]:

- large-scale, real time and perpetual simulations with distributed interaction,
- simultaneous visualization for many endpoints with unique perspectives, and
- multiple simulation engines with different operation characteristics.

4.2. Scalability Constraints

As the virtual worlds grow in size and in amount of content and concurrent users, problems arise most notably in the areas of the communication channel's bandwidth usage, script execution performance, and physics simulation. These problems can be present at either the client's or server's end, or at both ends. On the hardware level, the problem areas are typically bound to the central processing unit (CPU), graphical processing unit (GPU), amount of memory, i.e. random access memory (RAM), of both the aforementioned computational units, and to the latency and bandwidth of the network connection.

4.2.1. The Infinite World Assumption

When considering a theoretical situation of running an infinite virtual world with client and server with infinite resources, the scalability problem areas of a typical virtual world platform can be reasoned and identified. By beginning from the client's end, and solving each problem one at a time by assuming infinite resources as the solution, the problem areas in typical client-server architecture can be identified as follows:

- client's 3-D rendering performance (CPU and GPU),
- client's memory usage (assets, CPU and GPU RAM) ,
- server's application simulation (CPU usage),
- server's memory usage (scripts, assets),
- server's bandwidth usage,
- computing precision (both), and
- power consumption (both).

When rendering an infinite 3-D world with no view of distance limitations, the client's rendering performance will die out eventually. Even if the client could have an infinite 3-D rendering performance, the memory limitations of both the GPU and the CPU will become problems, as all of the renderable data will not fit into the memory at once.

If assuming infinite memory resources, and thus overall client-side performance, the server CPU performance required by the application simulation (scripts and physics) will become a problem. By assuming infinite server CPU performance, the server memory resources will become a problem, which, when solved, will lead to problems in the server-client bandwidth usage when transferring the infinite content.

Going even further and assuming an infinite, or even simply a very large, world where the values of the world coordinates get larger than (or around) 16 777 216 (2^{24}) units using single precision floating-point numbers, or 9 007 199 254 740 992 (2^{53}) when using double precision floating-point numbers, the computing precision will be a problem due to the IEEE 754 floating-point presentation that is typically used, as there are no more bits left to present any more precision. Finally, theoretical infinite computing and memory resources would of course require an infinite power supply, but that subject will be ignored in this context.

4.2.2. Data Flow

When the server has loaded and initialized the parts required by the server e.g. meshes for rigid bodies for authoritative physics simulation, the world content from the data storage is loaded into its memory. The server is then ready for accepting client connections. In this case we assume that a hard disk drive (HDD) located physically in the same computer is used for the data storage for simplicity.

When a client joins the server, the data from the server's HDD is transferred to the client via the network. When the client receives the data, it stores the data typically to its HDD in an asset cache, so that duplicate asset transfers are not needed upon possible reconnection, and loads the content to its memory. If the loaded data is GPU-related (mesh, texture, etc.), the data will be loaded to the GPU RAM. Commonly a copy of the data is also preserved in the CPU RAM for various purposes. This typical data flow of a networked 3-D application is presented in Figure 20.

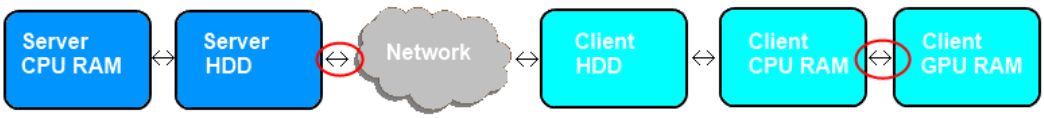

Figure 20. A typical data flow of a networked 3-D application.

Assuming that all of the contents of an infinite or just a relatively large world is transferred to the client always, the amount of data transferred during the initial connection is huge. This will mostly likely result in poor user experience due to the fact that there are no guarantees at which point the relevant assets from the user's point of view - the user's initial camera position in the world - arrive for the client.

Also, the client can't have all of the renderable data in its memory (CPU or GPU RAM) at once, so the data must be stored in the computer's mass storage device, and loaded into GPU and CPU RAM on an on-demand basis. This causes a data flow hotspot between CPU and GPU RAM. Relentless GPU data loading and unloading will result in poor performance for the end user. By utilizing an interest management mechanism the hotspot of data flow is moved to between the server's HDD and the network.

4.3. Scalability Solutions

Current virtual world architectures can be divided into two broad main categories: centralized and distributed. In this context, however, centralized doesn't necessarily mean a single centralized server, but typically a multi-server architecture. Distributed architectures typically refer to some kind of P2P architectures. The scalability solutions can be divided roughly into three categories. From a client's perspective, typical solutions include **client-side optimizations** such as enforced limitations, run-time level of detail (LOD) techniques, and asset load-on-demand techniques. From a single server's perspective, **interest management** is the most important technique. In the multi-server architectures it's possible to utilize **partitioning and distribution**.

4.3.1. Client-Side Optimizations

By using a limited draw distance, reducing the detail of the content (geometry, textures, etc.) using LOD techniques, and by performing other simplifications, it's possible to help the client to cope with rendering a large 3-D world. However, it's of course possible for a small set of very large and complex 3-D data to consume all of the CPU and GPU resources, even if only visible content is shown, so the content must be optimized regardless. In games and virtual worlds where the scene is mostly static, predefined and pre-calculated optimizations can be used to optimize rendering [42].

Loading assets on-demand means that assets are loaded and unloaded to CPU and GPU memories based on similar criteria as utilized by the interest management techniques: assets used by the visible (or audible, or otherwise relevant) objects for the user. Asset load-on-demand techniques focus more on solving the client-side rendering performance issues than the server's bandwidth usage, but naturally can also be combined with interest management techniques.

4.3.2. Interest Management

Interest management means utilizing the available geometric information, or other type of suitable attribute, in the client-server communication and data replication, to synchronize data between the participants on a need-to-know basis. This allows the server to handle more concurrent users as less bandwidth is used per client. For the client, using interest management means the ability to observe larger worlds in a slightly restricted fashion while maintaining a satisfactory performance. When interest management is used, all of the world's state and data is not typically replicated to clients at all times, meaning that the system uses a partially replicated world data distribution model.

Typical interest management approaches in game engines include *priority*- and *budget*-based approaches. In a priority-based approach some kind of formula is used to determine priorities for the objects in the world. A simple example of a priority formula could be:

$$priority = object's\ size\ /\ object's\ distance\ from\ the\ camera. \quad (1)$$

This priority is used for filtering network traffic and the amount of objects whose state is replicated to the client. In the budget-based approach, for example, a client-side polygon budget determines the size of the interest set ("don't send me more data that I can handle").

Various approaches exist for defining the client's area of interest (AoI). Traditionally interest management techniques are divided into two categories: region-based and aura-based. In addition to these, Lee et al. have identified three additional techniques: class-based filtering, hybrid approaches, and aggregation mechanisms [36].

In **region-based filtering** (Figure 21) the virtual world is divided into static regions or zones, either regular shapes, such as hexagonal, square, and triangle, or into *locales,* arbitrary sizes and shapes. [36]

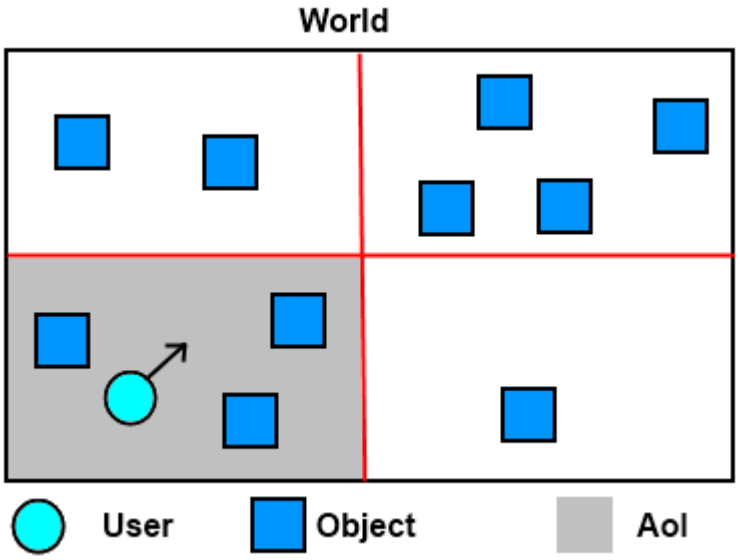

Figure 21. The basic principle of region-based filtering.

The messages are sent to all users within the region, and additionally also to adjacent and neighboring regions. Region-based filtering is simple, providing coarse-grained message filtering, which does not take users' actions into consideration (view, visibility, direction, etc.). Handovers between regions add some complexity to the system, so a key issue in region-based filtering is finding a balance between region size and computational overhead from management of different numbers of regions. Example systems using region-based filtering include SPLINE (locales) and NPSNET [43, 44]. [36]

In **aura-based filtering** (Figure 22), distance, visibility, audibility, or similar aura of interest defines the user's interest set. This type of filtering is based on the aura-nimbus model [45].

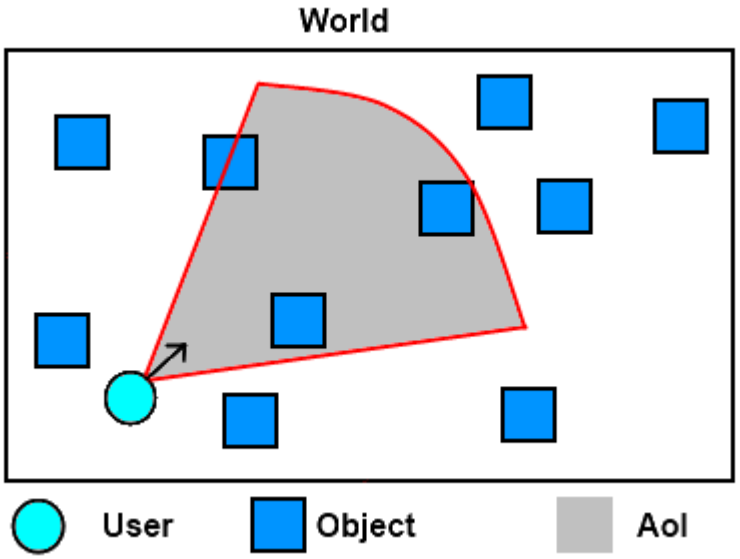

Figure 22. The basic principle of aura-based filtering.

In an aura-based approach the filtering is done on a per user basis, which makes it more dynamic than the region-based alternative. In this approach, the computational overhead is larger due to the need for distance calculation, raycasts, view frustum culling, or similar queries and computations. In some systems when users' auras overlap, a connection is established between the users and messages are exchanged through that connection. Example systems include DIVE, MASSIVE-3, and Sirikata [46, 47, 42]. [36]

In **class-based filtering**, the delivered messages are filtered based on the object types. For example in a military simulation, a soldier, a general, and a fighter plane, are interested in different things. Class-based filtering allows more fine-grained filtering compared to the proximity-based techniques. The scene model affects the possibilities and granularity of the filtering. Example systems using class-based filtering include HLA and Ding's scheme [48, 49]. [36]

Also **hybrid filtering techniques** exist combining elements from the aforementioned filtering techniques. Hybrid techniques try to find a balance between computational overheads and fine-grained data partitioning. Hybrid implementations have been developed at least for three-tiered architecture and VELVET [50, 51]. [36]

Aggregation filtering means merging update messages on aggregation hosts (server or local) from multiple clients and forwarding this aggregated update to the other clients. By using this technique only an abstracted (partial) view of target information is provided. However, many implementations of this type of filtering seem to use IP multicasting which makes it uninteresting. Example systems include Paradise and MASSIVE-2 [52, 53]. [36]

4.3.3. Partitioning and Distribution

Gupta et al. categorize the most common techniques of centralized architectures as follows: zoning, sharding, and instancing. In zoning, the virtual environment is geographically partitioned into areas of size that a single server machine is capable of handling. The zoning technique is also known as static (or fixed) load balancing. In

dynamic load balancing, the responsibility for parts of the world is moved between servers [5, 41]

In sharding, multiple, completely separate instances of the same environment, shards, are instantiated, each serving a different set of users. Instancing is a technique where a group of users are put into private zones, which are separate from the rest of the world. Instancing is mostly employed for game design reasons, not to resolve scalability issues. [41]

In order to keep the responsiveness somewhat sensible, many virtual worlds, Second Life and World of Warcraft for instance, enforce limitations on the amount of concurrent users and interactions within a single area. In World of Warcraft, a combination of instancing and sharding is used to separate millions of users into smaller private groups. Second Life has an enforced visibility and interaction distance of around 100 meters, and typically a Second Life region (256 m x 256 m) scales up to around 40 concurrent users. [42].

However, these limitations typically degrade the multi-user experience. In an ideal situation these kind of restrictions would not exist and the scalability problems for a large number of concurrent users would be handled with interest management and distribution techniques. However, for example in EVE Online the simulation is slowed down when the object density gets very high and this doesn't bother the players because it leaves them more time to contemplate their next move [54].

Distributing different types of computational tasks between multiple hosts is problematic, as the nature of computation typically is not task-based, but real-time streaming. In distributed systems, consistency and concurrency control is a very critical issue, which requires additional mechanisms. Distributed architecture typically means utilizing some kind of P2P or hybrid network architectures. Most of the techniques used in centralized architectures apply also for distributed architectures.

4.4. Discussion

Software architecture, scene model, data replication model, network architecture and communication protocol lay the foundations for opportunities and challenges regarding communication and computational scalability. Enforced limitations can be considered only as a temporary or additional solution, and not as permanent solutions for scalability problems.

The level of required message rate can vary between different worlds and applications - consider for example a high paced first-person shooter type of game versus an intergalactic starship war game - so, ideally the virtual worlds system supports various levels and techniques. The key to interest management is finding a balance between bandwidth usage and server CPU load. As the majority of the traffic is produced by objects' movement, an efficient movement packet format and client-side prediction is essential before considering further communication scalability optimizations.

It seems obvious that a hybrid filtering mechanism which combines features of region-, aura-, and class-based filtering would be the most optimal one. The mechanism needs to be highly configurable and have run-time adaptivity when the number of users and object density grows, meaning a somewhat high (but ideally configurable) server-side computational footprint per user. Hence optimized data structures, threading, and other software design aspects must be carefully applied.

When considering a theoretical scenario with one user but infinite world, compared to the typical approach of immediately partitioning the world into a server grid, where each server runs part of the world, the world could be divided into logical partitions within one server instance. Running a single server and taking into consideration that the server has limited and realistic amount of resources, the aforementioned client-side optimizations and interest management will help, but some kind of multi-server architecture will become inevitable at some point. This adds some complexity to the overall system.

When considering one user, an infinite world, and unlimited resources on the server, it doesn't make sense to keep the whole scene in memory. Instead, scene server offload could be used: the non-pertinent part (non-visible or otherwise non-relevant for the users) of the scene is kept in suspended mode and only the pertinent part of the scene is kept in memory. When a user moves to new part of the world, a snapshot of previous part is saved to storage and unloaded from the memory and a new part of the scene is loaded from the storage. This would be possible of course assuming that the non-pertinent parts of the scene do not require constant real-time simulation, for example pertinent artificial intelligence simulation.

5. A 3-D VIRTUAL WORLD IMPLEMENTATION

The realXtend Tundra (simply referred as Tundra from now on) was chosen for the interest management implementation testbed due to its open source nature [55]. Also, the author is very familiar with the platform and its codebase, being worked on it both professionally and on free time for several years. Also, the fact that Tundra currently has no established interest management technique in place makes it ideal to see what kind of improvements incorporating an interest management technique – even a simple one - into its scene synchronization mechanism produces.

5.1. realXtend Tundra

Tundra is an open source and a cross-platform virtual world and 3-D internet SDK, and it currently runs on common desktop operating systems Windows, OS X, and Linux. Tundra supports targeting Android, but currently in a limited fashion. Tundra doesn't provide a full virtual world ecosystem out of the box, but is aimed primarily at application developers as a platform for creating networked 3-D worlds with customized content.

Tundra uses the typical client-server networking architecture with reliable UDP, TCP or WebSocket for the basic communication and world state synchronization. Also experiments have been done to support SCTP, but this is not officially available in Tundra [56]. The actual asset data can be transferred via multiple routes, but usually with HTTP, which although is somewhat suboptimal it is convenient, and often used for public networked worlds. A somewhat unique approach in Tundra compared to many other virtual world platforms is the fact that exactly the same codebase implements both the client- and server-side functionality.

The heart of Tundra design is the generic and extensible *scene-entity-component-attribute* scene model. Each scene consists of entities. An entity is an actor in the scene with a unique identifier number (within the parent scene). Each entity contains a set of components that define the data – a set of attributes – and usually possibly add new behavior to the parent entity. Components are identified by unique type name (and unique type identifier number) and an optional arbitrary identifier name.

A mechanism called *entity actions* is provided for object communication and remote procedure calls (RPCs) in a data-driven way, but other mechanisms, such as HTTP, can also be used. An overview of the Tundra platform's architecture is presented in Table 4.

Table 4. Tundra architecture overview.

Network architecture	Client-server
Networking protocol	Reliable UDP, TCP and WebSocket
Application	Modular and extensible plug-in architecture
Data distribution model	Hybrid
Interest management	None
Scene model	Generic and extensible entity-component model
Object communication	Entity actions + other (HTTP et al.)

Tundra is based on the Qt application framework and OGRE 3-D rendering engine [57, 58]. Other notable third-party libraries used for the core functionalities include MathGeoLib (math and geometry), kNet (reliable UDP and TCP networking), and

Bullet Physics Library (physics simulation) [59, 60, 61]. The Tundra application consists of a fixed set core of components and plug-ins, accompanied by a user-defined set of additional plug-ins. The application logic typically is implemented by scripts written in dynamic scripting language QtScript (JavaScript) provided by the Qt framework.

5.2. Current State

Tundra's main development goals have been simplicity, modularity, and easy usability for the application developers. Tundra is essentially a single-threaded application, meaning that the core functionalities, i.e. graphics, physics, audio, and scripts, are run in a single thread. This is very typical for game engines.

The development has been driven by the requirements of the prevailing applications built on top of Tundra and not scalability per se. So far there has been no such application that would have required enhancing the support for very large worlds. However, special attention has been paid to the wire protocol efficiency and optimization early on during the development.

For now, most of the public Tundra scenes online have been relatively small [62]. With the current Tundra network protocol, and most notably the rigid body replication optimizations, a single moving object averages at about 11 bytes an update, whereas the old rigid body streaming code was about 70 bytes per update [63]. With Tundra, an adequate performance with up to 64 concurrent users with an avatar can be achieved [63]. This is in line with the maximum number of concurrent users of other popular 3-D virtual worlds such as Second Life and World of Warcraft. The number per se doesn't mean much though as in practice the maximum number of concurrent users while maintaining an adequate performance is largely dependent on the other content of the world, which will become the bottleneck in Tundra as the amount of content in the world grows.

Currently Tundra has no established interest management technique in use, however the scene synchronization protocol of Tundra is greatly optimized already [64]. It should be noted that some common protocol features in many virtual world platforms, for example user permissions and time synchronization, are not part of Tundra's core protocol.

5.2.1. Performance Issues

The most notable known server-side performance issues in Tundra are related to inefficiencies in the scripting engine (QtScript) and its interoperability with the native code. The most notable server-side single issue is the physics scripting when the amount of moving rigid body entities in the scene grows. However, it's possible to work around this problem by using an existing `PhysicsMotor` component that drives a `RigidBody` component of the same entity by impulses on each physics update from the native code so there is no need to jump between script and native code for driving the physics. [64]

The biggest client-side issue is the graphics rendering and script performance. As the graphics rendering and script performance bottlenecks are significant and not simple to resolve, this thesis keeps the focus on optimizing the network traffic due to its general value for scalability. [64]

5.2.2. World State Replication

Tundra uses a hybrid data distribution model. Entities and components in the scene can be either replicated or local-only (exist only at the specific host). By default, an entity's state, i.e. the set of components and their attributes, is altered in a replicated fashion. Tundra also supports local-only (only the host performing the action is notified) and disconnected (no notifications whatsoever are done) changes can also be performed.

Due to the generic nature of the Tundra platform, the platform itself doesn't really enforce the data distribution model: without specific application scripts, any modification to the world state is done by the copy residing at the particular host, which is then replicated to other hosts. If a host decides to reject a change, it will not signal the other hosts of this by default and thus leaves the states of different hosts in an inconsistent state. In other words, it's simply up to the application developer to guarantee the state consistency.

Tundra's network protocol is relatively simple due to the generic scene model. The Tundra scene synchronization protocol consists of the following network messages:

- `CreateEntity`,
- `CreateEntityReply` (server-client only),
- `RemoveEntity`,
- `CreateComponents`,
- `CreateComponentsReply` (server-client only),
- `RemoveComponents`,
- `CreateAttributes`,
- `EditAttributes`,
- `RemoveAttributes`, and
- `RigidBodyUpdate` (server-client only).

The purposes of the different messages should be pretty self-explanatory: each of the messages alters the state of the scene-entity-component-attribute scene model in a generic fashion, with the exception of `RigidBodyUpdate` which is a custom message for streaming movement of a spatial entity. Despite its name, the `RigidBodyUpdate` message is also used for updating the position information of non-physical objects. The `EditAttributes` message is also used to update an object's position when doing a full transform (position, orientation, and scale) update, with no client-side extrapolation.

A Tundra application's `TundraLogicModule` has a `SyncManager` class instance which takes care of synchronizing the world (`Scene`) state changes. On the server, a `SceneSyncState` is created for each `UserConnection`, a class representing a client-server connection. Changes to the server's `SceneSyncState` are replicated to all connected clients. The client has a virtual `UserConnection` instance for replicating the client's `SceneSyncState` changes to the server and to the other clients.

A `SceneSyncState` consists of `EntitySyncState`s which consist of `ComponentSyncState`s which hold a list of possible *dirty* (value has been changed, but not yet replicated to other participants) attributes. When the value of an attribute changes in a `Scene`, signals (events), to which the `SyncManager` is connected (subscribed) to, are emitted (published). `SyncManager` keeps track of these changes and marks the appropriate sync states dirty to `SceneSyncState`'s dirty queue.

During each network update tick (configurable, 20 Hz by default), the dirty queue is processed fully and state change network messages are sent accordingly. Replication of moving entities, with or without the `RigidBody` component, is optimized and handled as a special case (`RigidBodyUpdate`). When new entity, component, or attribute is created or removed, the corresponding message is sent.

5.3. Improved Scalability Using Priority-Based Filtering

The work was based on the Tundra 2.4 codebase [65]. The interest management code and the Infinite World test scene presented in this thesis, along with a complete commit history, is available at GitHub [66]. The implementation was inspired by the general idea of interest management used in "Halo: Reach" [67]. The implemented interest management technique can be considered an aura-based filtering mechanism which contains some elements of class-based filtering techniques. Contrary to the typical aura-based interested management filtering techniques, a priority cut-off radius was not implemented meaning that the set of replicated entities was constant on the server and the client.

5.3.1. Implementation Rationale

As the 3-D rendering performance itself, even with a somewhat modestly complex content, will become a bottleneck in Tundra as the 3-D content in the world grows, a server-side approach is chosen: the goal of the interest management implementation is to lessen server's bandwidth usage when the amount of moving (physical) objects and concurrent users in the world is high. As interest management is mostly for optimizing an invisible aspect for the end-user (content far away), the perceptual performance must be retained, i.e. the end-user experience must not have any significantly notable difference in the objects' movement whether the interest management is enabled or not in a scenario where the server is not overburdened. The following additional goals are set for the implementation:

- simple and efficient (modest CPU overhead),
- must work in "headless" mode (i.e. when rendering disabled), and
- easy to enable and disable, and (due to simplicity) easy to modify or enhance.

Making the interest management work in headless mode is crucial as it makes no sense to run the server with rendering enabled ("headful") in production usage: this would take a considerable amount of both CPU and GPU resources for no reason. These additional goals make the interest management technique usable in real life productions.

Also, when taking the current scalability bottlenecks into consideration, the following non-goals are set for the implementation:

- client rendering performance can be considered non-pertinent as focus will be on the server's bandwidth usage,
- due to the physics and script performance issues mentioned in Chapter 5.2.1., maintaining an optimal, i.e. 60 FPS, frame rate during the tests is not important, and
- runtime and/or script access to the actual interest management algorithm details not important.

5.3.2. Performance Evaluation Environment

For profiling and measuring the interest management algorithm and functionality, a test world and application named "Infinite World" was created. In Infinite World, the virtual world consists of symmetrical *NxN* grid of symmetrical *WxW* (where *W* is the width of a side in world units) cells, *scene blocks*. Each scene block has *B* physical non-player characters (NPCs), also known as bots, walking in circles via configurable waypoints. Each scene block can also contain *C* non-physical objects, simple boxes that move up and down constantly. All of the attributes are configurable during startup ($N \geq 3$, odd numbers allowed; $W > 0$; $B \geq 0$; $C \geq 0$). This Infinite World functionality is presented in Figure 23.

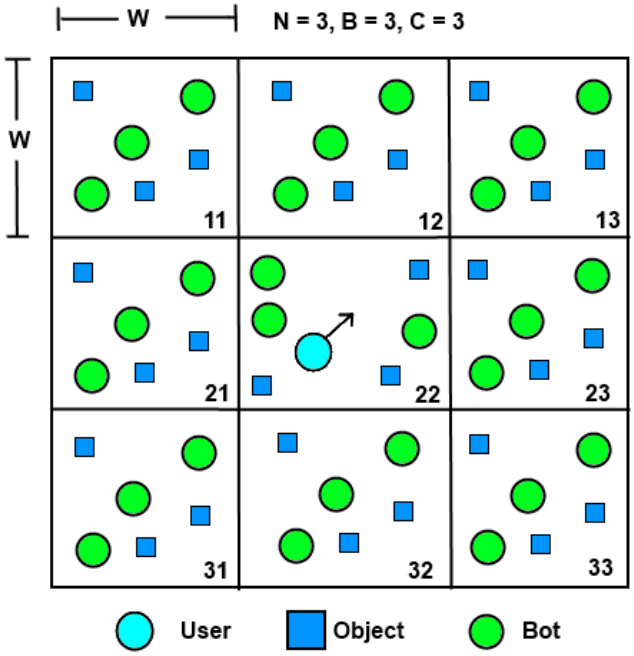

Figure 23. The configurable Infinite World scene block grid.

The server handles all of the Infinite World application logic, while the client is simply a dummy observer. An existing avatar application script was used to create an avatar for each client joining the server. When the user joins the world, and his avatar is created, the avatar is positioned to the middle of the grid. When a user enters a new scene block, a new row or column of scene blocks is created to the direction the user is travelling, and the most distant row or column of scene blocks is removed. A screenshot of the test application is presented in Figure 24.

Figure 24. Screenshot of the Infinite World.

5.3.3. Observer, Priority, and Relevancy

In the implementation, a concept of an *observer* was added to the `SyncManager`. Any entity with a `Placeable` component can act as the observer. When the observer is set (not done automatically for now), the position and orientation of the observer entity is sent to the server. Not forcing the observer to the active camera adds flexibility and debugging capabilities: it's possible to keep the actual observer camera entity constant while switching to another camera for observing the effect of the interest management technique.

Two variables were added to `EntitySyncState`: *priority* and *relevancy*. These two factors together are used to compute a prioritized synchronization frequency of an object. Priority of an entity is:

$$priority = size / distance, \tag{2}$$

where *size* is size of the object's bounding volume's surface area, and *distance* is the distance between the object and the observer. Relevancy is an arbitrary factor to which existence and attributes of different components can be made to affect, for example an object's velocity, if the `RigidBody` component exists. Larger numbers for both priority and relevancy means larger importance.

5.3.4. Implementation Details

Distance is applicable only for spatial entities, entities containing the `Placeable` component. For non-spatial entities, the highest possible priority (positive infinity) was used as non-spatial entities are typically some kind of application logic (script) or data entities, whose immediate replication is typically important. Size is only applicable for entities containing the `Mesh` component, 3-D geometry, but it could also easily be expanded to cover spatial audio sources, volume triggers, and other types of invisible but spatial objects. For spatial entities without the `Mesh` component, the default synchronization rate was used.

Currently only the existence of `Placeable`, `Mesh`, and `RigidBody` components affect priority and relevancy. A prioritized synchronization interval (in seconds), f_{sync}, for a single entity is

$$f_{sync} = log2((100 * f_{max}) / (priority * relevancy)), f_{sync} \in [f_{max}, f_{min}], \quad (3)$$

where f_{max} is the maximum allowed synchronization interval, i.e. the smallest allowed value, and f_{min} is the minimum allowed synchronization interval, i.e. the largest allowed value. The equation was created by trying to achieve a formula that gives values close to f_{max} on objects that have high to medium priority, and then significantly steers towards f_{min} when the priority gets low. For the tests, f_{max} was 0.05 and f_{min} was 5.

First, the dirty entities are sorted before sending sync messages according to their priority in descending order (most important first). In `SyncManager`'s `ProcessSyncState` and `ReplicateRigidBodyChanges`, the time since the object's last update is checked and compared to a varying state update frequency for the object that is calculated as described in Equation 3. If enough time has not passed, the update is not issued.

A new `ObserverPosition` message was added to the protocol, telling the position and orientation of the observer entity. The message (client-server) consists of the following information:

1. scene ID,
2. position of the observer, and
3. orientation of the observer.

For now, the scene ID is just a placeholder, which has no use until Tundra has a proper multi-scene support. The scene ID was added for consistency, as the placeholder scene ID is present in all of the current protocol messages. The ID is a variable-length encoded (VLE) unsigned integer, 4 bytes at maximum, but typically (and during the tests) 1 byte. Both position and orientation (in Euler angles) consist of three 4-byte floating-point numbers each, yielding total of 24 bytes. The total content length of the message was 25 bytes for the tests.

The application script sets the observer entity to `SyncManager` on the client. As long as the observer entity exists and has a `Placeable` component, the observer's position information is sent to the server using the `ObserverPosition` message at the default scene synchronization rate (20 Hz).

When the server receives the `ObserverPosition` message, it saves the information for use. Priorities for all entities are recomputed at one second interval (configurable). The priority of a single entity is also computed upon its creation and upon addition or removal of a component. These priorities are used immediately during the next update tick. The interest management technique could be enabled and disabled at run-time by setting the `interestManagementEnabled` property of `SyncManager` in order to observe the effect of the technique easily.

6. EXPERIMENTS

For the final tests, *N* was 9, *W* was 128, *B* was 1-11, and *C* was 0, meaning a world with 1152 x 1152 dimensions with 81-891 bots running. During the development, constantly moving non-physical objects were added to the scene (i.e. $C > 0$) in order to see that the priority computation formula was applicable also for non-physical objects. For the final tests and measurements, however, the non-physical objects were omitted as it was observed that they hindered the performance significantly (the aforementioned scripting performance inefficiency).

For the tests, the client's `AvatarCamera` entity, a camera following the avatar, was used as the observer. The client was able to change to the `FreeLookCamera` entity, a camera that can move around freely, to observe more closely how the priority-based filtering logic affects the movement of the distant entities. For the final tests, the avatar did not move between blocks in order not to cause any unnecessary server overhead spikes which would occur when moving between scene blocks.

The Tundra client and server were run on a single same computer using the *localhost* network. It was observed that it was possible to run a single server and maximum of 30 clients at once without performance and reliability problems at the available computer. Due to this limitation, the moving objects in the world were chosen to be solely bots instead of real user connections in order to be able to have hundreds of simultaneously moving objects in the world.

6.1. Experimentation System

The tests were run on a computer consisting of Intel Core i7-3770K 3.5 GHz CPU, 8 GB RAM, NVIDIA GeForce GTX 660 GPU, and a 1 GB hybrid HDD running 64-bit Windows 8.1 Update 1. The Tundra application was built as a 32-bit Windows application using Microsoft Visual Studio 2008 Service Pack 1. For the code execution time measurement, the built-in Tundra profiling functionality and the Profiler window (Figure 25) were used.

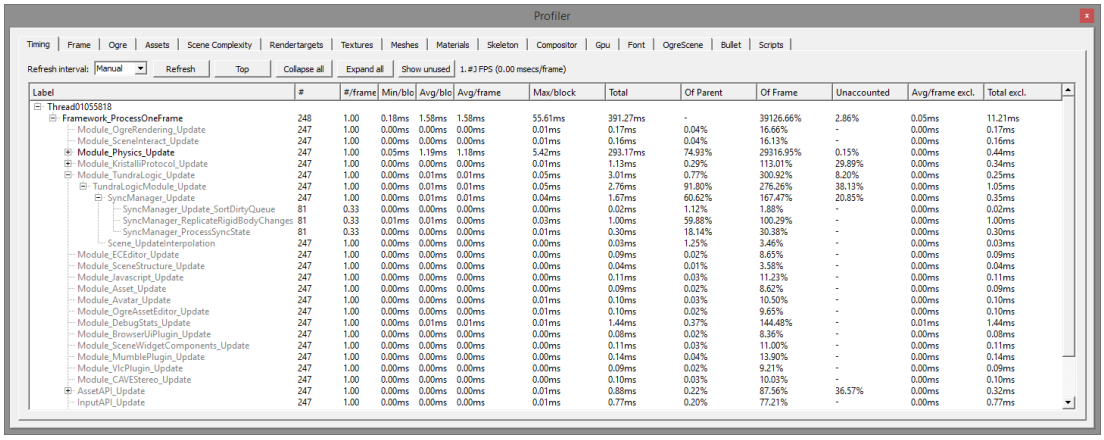

Figure 25. The code execution profiler window.

Bandwidth usage measurements were done by using the built-in functionality and statistics window (Figure 26) of the kNet networking library. It should be noted that

enabling and running the code execution and network profiling functionality causes some computational overhead in the application itself.

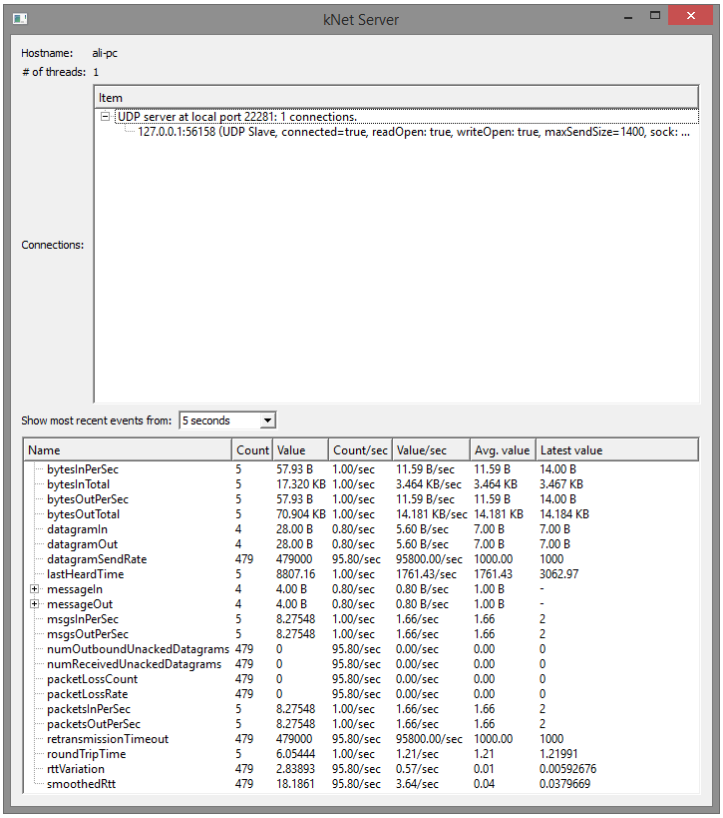

Figure 26. The network statistics window.

6.2. Measurements and Results

The measurements were done by configuring the desired number of bots to the scene, starting the server, joining the server with the client, and then observing the traffic from the server's network statistics dialog. The amount of network traffic fluctuated to some degree due to the constantly changing priorities of the bots, so the traffic was observed for couple tens of seconds in order to figure out some kind of approximate mean value.

The measurements are presented in Figure 27. The server had filtering enabled by default so the "IM enabled" figure with certain amount of bots was measured first. After that, the filtering was disabled at run-time and the "IM disabled" figure was measured. These steps were repeated eleven times. 972 (9 * 9 * 12) bots caused the server to run at 10 FPS which was too slow and caused clearly unresponsive user experience on the client. Thus 891 (9 * 9 * 11) bots was chosen as the upper limit for the tests. The server was able to perform over 20 FPS while running 891 bots, so, while not having an optimal performance, it provided somewhat responsive experience for the client.

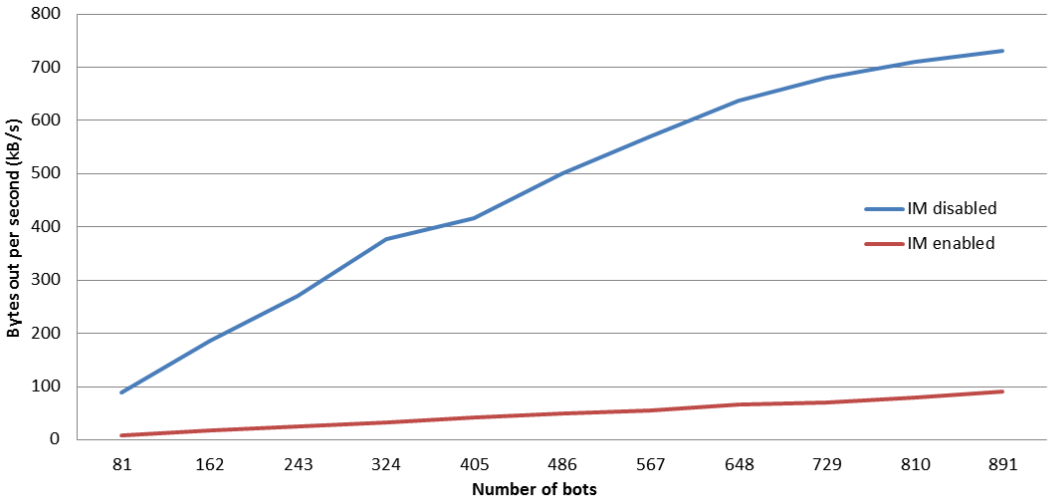

Figure 27. Server's bandwidth usage with and without interest management.

From the measurements it can be seen that the impact of the filtering was significant: the bandwidth usage is cut down roughly tenfold. It was quickly tested that when connecting to the server with multiple clients, the impact of interest management filtering continued to be significant: the server's bandwidth usage was cut down to roughly 15 % of the original (without interest management). With interest management filtering it was possible to run 891 bots in the scene with the same bandwidth usage that was used while running 81 bots without the filtering.

As a conclusion it can be seen that even though the scene state synchronization protocol was highly optimized already, a simple priority-based filtering gives nice results and allows increasing the number of moving physical objects in the scene significantly. The server-side computational footprint per user was observed to be somewhat high, so optimizations regarding this area are required. The possible improvements are discussed further in the next chapter.

7. DISCUSSION

In general, there's no point in focusing on the online scalability of a virtual world scene, if the scene doesn't even scale offline due to its complex nature. In practice this means the existence of excessive amounts of unnecessarily complex meshes, large and uncompressed textures, and other data. A good guideline is to design the world content around technical limitations of a wide range of client hardware capabilities. Additionally and preferably LOD techniques must be applied. The OGRE rendering engine supports LOD for meshes, but currently this is not utilized by Tundra.

In game development these matters are typically heavily optimized early on due to the static nature of the 3-D scenes, but in general use virtual worlds' content is often created naively on an ad hoc basis. Run-time compression techniques of 3-D models and textures can help, but this naturally causes computational overheads.

Instead of optimizing the current and usually obvious bottlenecks, partitioning and distributed computing are seen as a solution that automatically resolves all of the problems. Instead these techniques should be seen as an optimization strategy only after the performance hotspots and bottlenecks are optimized in the local domain.

7.1. Observed Problems

The focus of the test was to optimize a server's bandwidth usage, so, in order to overcome some notable performance issues of Tundra, certain simplifications were applied to the test case. The disabled features in the test preferably should be enabled in the future in order to get a more realistic picture of the client's capabilities.

The playback of the skeletal animations of the NPCs was disabled in order to increase the client-side rendering performance. Furthermore, in order to lessen the client-side rendering performance, flat terrain geometry was used using a simple box mesh that was flattened and stretched to cover a single scene block. The simpler terrain was chosen due to terrain's irrelevancy in the bandwidth usage context, but also due to the current `Terrain` component's inefficient LOD technique, which caused generally poor 3-D rendering performance. As a side note, the terrain rendering performance could be improved for example by using a *geometrical mipmapping* technique [68].

During the testing, discrepancies for some objects' bounding volume information were noticed when comparing the values between headful and headless modes. This issue needs to be researched and fixed. In general, support for bounding volume retrieval for billboards (3-D sprites that face the camera at all times), particle effects, sound sources, and other types of objects that can be considered to have a sense of proximity to the user, should be taken into account when calculating the priorities.

Due to the limitations of the underlying rendering engine, retrieval of bounding volume information in headless mode was difficult, and in some cases even impossible. In order to fully prioritize all types of visual objects, accurate bounding volume information would be needed also in the headless mode. In the current state, it could be possible to utilize the physical bounding volume information instead of graphical information for the server-side computations.

7.2. Future Work

Due to the goals and non-goals of the implementation, various possibilities for improving the interest management implementation exist. From more of a high-level perspective, a dynamic interest management registration mechanism could be useful: Tundra plug-ins could register different interest managers to the system without need to alter the core Tundra codebase. More importantly, a plug-in mechanism would allow the use of different types of interest management implementations depending on the requirements and needs of different types of scenes and applications. It also might be convenient to switch between different implementations at runtime.

7.2.1. Implementation

Ideally, the server should be able to define an output per user threshold, which the server could automatically adjust accordingly to the number of concurrent users. Some other data structure could be used for the dirty object list instead of the current doubly-linked list, for example a min-max heap or other type of priority queue. The min-max heap, for example, would remove the need to sort the list manually, but would make insertions and removals slower, so, profiling advantages and disadvantages of different data structures would be required when seeking the most suitable data structure for the task. Also, maintaining some kind of priority groups or ranges for objects, instead of every object having its own priority, could make sense as the priorities can be virtually the same for objects within the same area and small changes in the priority don't have significant effects on the synchronization rate. Also, factoring in an object's velocity vector (direction and speed) for the relevancy computation would help refine the priority calculation.

Currently the object priorities are calculated at fixed interval. Server CPU overhead could be observed when the server was computing priorities for many hundreds of objects, especially when number of concurrent users in the world started to grow. Threading the priority computation task could be used as a solution. An alternative approach would be not computing all priorities in a one go, but instead in batches, of say, 100 entities per frame.

A new `SetObserver` message, containing a scene ID and an entity reference (ID or name) of the desired observer entity could be added to the protocol. This message would allow the server to tell what entity the client must use as the observer when the client joins the server. The size of the `ObserverPosition` message could be reduced by optimizing the position and orientation in similar fashion as is done in the `RigidBodyUpdate` message. The `ObserverPosition` message could also be made server-client to tell the initial position, or to force the position, of the observer.

7.2.2. Filtering Technique

The next step for the filtering technique would be to define a cut off priority which would be used to remove the most distant objects from the client's view in order to ease the client's rendering workload on large scenes. Additionally, a view frustum query –based, or similar, technique for defining client's interest would be needed for more refined prioritization, meaning incorporating elements of aura-based filtering to the solution. For example, the solid angle query technique sounds promising [42].

However, this would increase the client's CPU overhead, so efficient implementation is required. The client could send the interest set to the server, or alternatively the server could construct the client's view frustum from the `ObserverPosition` information and use frustum-volume or ray-volume intersection tests.

7.2.3. Infinite World

For the Infinite World application, particle and sound effects, and other enrichments could be added in order to cover wider range of typical object types. Also the features that were disabled to overcome the performance bottlenecks should be enabled in order to get more realistic picture of the client's capabilities. The scene server offload functionality mentioned in Chapter 4.4 would be interesting to prototype using Infinite World. Due to time constraints, the Infinite World does not fully support multiple users moving into different scene blocks (multiple users within the same scene block work OK), so a proper multi-user support should be implemented for the application.

8. SUMMARY

In this thesis a working and production-ready interest manager technique was presented to lessen the server's bandwidth usage considerably when running a scene containing lots moving of objects. The implementation provides a clean and simple base for future development and improvements.

As computational scalability issues, such as graphics rendering and physics simulation, in 3-D environments are already well addressed with existing solutions, this thesis considered the performance problems of multi-user virtual world client-server communication when working on large virtual worlds containing arbitrary data and functionality.

Despite the differences in the existing virtual world applications and platforms, all of them share common needs: representing visual 3-D objects, moving and animating them, synchronizing all the necessary data among the participants, and storing it for continuing use and all of this in an efficient manner.

In order to succeed in this task, attention must be paid to the scene model that represents contents of the world, and the protocol that is used to disseminate the changes in the content. Also, by utilizing the available geometric information in the client- server communication and data replication, the data is synchronized between the participants on a need-to-know basis.

For Tundra, physics simulation and especially script performance must be optimized at some point in order to achieve good performance with larger and richer scenes. Some sort of multi-threading or multi-process optimization can be seen to be inevitable at some point. Also, having dedicated servers for different tasks (scripts, physics) by distributed computing could be interesting.

9. REFERENCES

[1] Murray J. (1997) Hamlet on the Holodeck: The Future of Narrative in Cyberspace. Cambridge, The MIT Press. 336 p.

[2] Biocca F. & Delaney B. (eds.) (1995) Communication in the age of virtual reality. Hillsdale, NJ, USA, Lawrence Erlbaum Associates. 416 p.

[3] Wolf M. & Perron B. (eds.) (2003) The Video Game Theory Reader. Routledge. 320 p.

[4] Lombard M. & Ditton T. (1997) At the Heart of it All: The Concept of Presence. Journal of Computer-Mediated Communication 3(2): 0. DOI: http://dx.doi.org/10.1111/j.1083-6101.1997.tb00072.x.

[5] Bartle R. (2003) Designing Virtual Worlds. New Riders Publishing, 768 p.

[6] Singhal S. & Zyda M. (1999) Networked Virtual Environments. Addison-Wesley Professional. 331 p.

[7] Liu H., Bowman M., Adams R., Hurliman J. & Lake D. (2010) Scaling Virtual Worlds: Simulation Requirements and Challenges. Proc. 2010 Winter Simulation Conference. Johansson B., Jain S., Montoya-Torres J., Hugan J. & Yücesan E. (eds.), 778-790. DOI: http://dx.doi.org/10.1109/WSC.2010.5679112.

[8] Anonymous (2014) World of Warcraft. URL: http://www.worldofwarcraft.com. Accessed 27.04.2014.

[9] Anonymous (2014) EVE Online is a Massive Multiplayer Online Roleplaying Space Game – EVE Online. URL: http://www.eveonline.com. Accessed 27.04.2014.

[10] Anonymous (2014) Second Life Official Site – Virtual Worlds, Avatars, Free 3D Chat. URL: http://www.secondlife.com. Accessed 27.04.2014.

[11] Anonymous (2014) AION Free-to-Play. URL: http://en.aion.gameforge.com/website/. Accessed 12.05.2014.

[12] Anonymous (2014) Play Lord of the Rings Online™ Free! URL: http://www.lotro.com/en. Accessed 12.05.2014.

[13] Anonymous (2014) Free MMORPG – RuneScape 3 – Online Fantasy Role Playing Game. URL: http://www.runescape.com/. Accessed 12.05.2014.

[14] Anonymous (2014) Dungeons and Dragons Online URL: https://www.ddo.com/en. Accessed 12.05.2014.

[15] Anonymous (2014) Habbo Hotel – Make friends, join the fun, get noticed! URL: https://www.habbo.com/. Accessed 12.05.2014.

[16] Sun Q. & Gramoll K. (2004) Internet-based Simulation and Virtual World for Engineering Education. Engineering Design Graphics Journal 68(1): 13-21.

[17] Chodos D., Stroulia E., Boechler P., King S, Kuras P., Carbonaro M. & de Jong E. (2010) Healthcare Education with Virtual-World Simulations. Proc. 2010 ICSE Workshop on Software Engineering in Health Care. Cape Town, South Africa, 89-99. DOI: http://dx.doi.org/10.1145/ 1809085 .1809097.

[18] Lifton J. & Paradiso J. (2010) Dual Reality - Merging the Real and Virtual. MIT Open Access Articles. URL: http://dspace.mit.edu/handle/ 1721.1/61991. Accessed 26.05.2014.

[19] Santaniemi M. (2010) Visualizing Open Source Project Information Using RealXtend Virtual World Platform. Master's Thesis. University of Oulu, Department of Computer Science and Engineering.

[20] Bainbridge W. (2007) The Scientific Research Potential of Virtual Worlds. Science 317(5837): 472-476. DOI: http://dx.doi.org/10.1126/science .1146930.

[21] Anonymous (2011) 3D-Multipolis virtuaalikokousmaailman esittely. URL: http://www .multipolis.net/index.php?1141. Accessed 12.05.2014.

[22] Cook M., Tweet J. & Williams S. (2003) Dungeon Master's Guide v3.5. Wizards of the Coast. 320 p.

[23] Kärkkäinen H. (2011) Tässä ovat Suomen suurimmat pelihitit. URL: http://www.taloussanomat.fi/harrastukset/2011/04/11/tassa-ovat-suomen-suurimmat-pelihitit/20114852/139. Accessed 27.04.2014.

[24] Anonymous (2010) World of Warcraft® Subscriber Base Reaches 12 Million Worldwide. URL: http://eu.blizzard.com/en-gb/company/press/ pressreleases.html?id= 10007508. Accessed 27.04.2014.

[25] Kalawsky R. (1993) The Science of Virtual Reality and Virtual Environments. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc. 405 p.

[26] Cruz-Neira C., Sandin D & DeFanti T. (1993) Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques. Anaheim, CA, USA, 135-142. DOI: http:// dx.doi.org/10.1145/166117.166134.

[27] Anonymous (2014) PlayStation® Global. URL: http://playstationmove .com/index.html. Accessed 14.05.2014.

[28] Anonymous (2014) Kinect for Xbox 360. URL: http://www.xbox.com/en-US/xbox360/ accessories/kinect/KinectForXbox360. Accessed 14.05.2014.

[29] Anonymous (2011) Academics, Enthusiasts to Get Kinect SDK URL: http://research .microsoft.com/en-us/news/features/kinectforwindowssdk-022111.aspx. Accessed 27.04.2014.

[30] Anonymous (2014) Oculus Rift –Virtual Reality Headset for 3D Gaming. URL: http://www.oculusvr.com/. Accessed 14.05.2014.

[31] Anonymous (2014) Meshmoon – Create 3D space. URL: http://www .meshmoon.com. Accessed 08.05.2014.

[32] Alpcan T., Bauckhage C. & Kotsovinos E. (2007) Towards 3D Internet: Why, What, and How? Proc. International Conference on Cyberworlds. Hannover, Germany, 95-99. DOI: http://dx.doi.org/10.1109/CW.2007.62.

[33] Anonymous (2014) FI-WARE - Open APIs for Open Minds. URL: http://www.fi-ware.org/. Accessed 27.04.2014.

[34] Macedonia M. & Zyda M. (1997) A Taxonomy for Networked Virtual Environments. Journal IEEE MultiMedia archive 4(1): 48-56. DOI: http:// dx.doi.org/10.1109/93.580395.

[35] Fleury C., Duval T., Gouranton V. & Arnaldi B. (2010) Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments. IEEE VR 2010 Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS).

[36] Seunghyun H., Mingyu L., Dongman L. & Hyun S. (2008) A scalable interest management scheme for distributed virtual environments. Computer Animation and Virtual Worlds 19(2): 129-149. DOI: http://dx.doi.org/10.1002/cav.218.

[37] Marshall D., Delaney D., McLoone S. & Ward T. (2004) Challenges in modern Distributed Interactive Application design. URL: http://eprints.nuim.ie/1445/. Accessed 27.05.2014.

[38] Fette I. & Melnikov A. (2011) RFC 6455 – The WebSocket Protocol. URL: https://tools.ietf.org/ html/rfc6455. Accessed 25.04.2014.

[39] Brutzman D. (1999) Virtual Reality Transfer Protocol. URL: http://faculty .nps.edu/brutzman/vrtp/. Accessed 27.04.2014.

[40] Broll W. (1998) DWTP - an Internet Protocol for Shared Virtual Environments. Proc. Third Symposium on Virtual Reality Modeling Language. Monterey, CA, USA, 49-56. DOI: http://dx.doi.org/10.1145/ 271897.274370.

[41] Gupta N., Demers A., Gehrke J., Unterbrunner P. & White W. (2009) Scalability for Virtual Worlds. Proc. IEEE 25th International Conference on Data Engineering. Shanghai, China, 1311-1314. DOI: http://dx.doi.org/10.1109/ICDE.2009.228.

[42] Cheslack-Postava E., Azim T., Mistree B. Horn D., Terrace J., Levis P. & Freedman M. (2012) A Scalable Server for 3D Metaverses. Proc. 2012 USENIX Annual Technical Conference. vol. 12. Boston, MA, USA.

[43] Barrus J., Waters R. & Anderson D. (1996) Locales: supporting large multiuser virtual environments. IEEE Computer Graphics and Applications 1996; 16(6): 50–57.

[44] Capps M., McGregor D., Brutzman D. & Zyda M. (2000) NPSNET-V: a new beginning for dynamically extensible virtual environments. IEEE Computer Graphics and Applications 20(5): 12–15

[45] Benford S. & Fahlen L. (1993) A Spatial Model of Interaction in Large Virtual Environments. Proc. Third European Conference on Computer Supported Cooperative Work, Milan, Italy, 109–124.

[46] Hagsnad O. (1996) Interactive Multiuser VEs in the DIVE System. IEEE Multimedia 3(1): 30–39. DOI: http://dx.doi.org/10.1109/93.486702.

[47] Greenhalgh C., Purbrick J. & Snowdon D. (2000) Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. Proc. ACM Collaborative Virtual Environments, San Francisco, CA, USA, 119–127. DOI: http://dx.doi.org/10.1145/351006.351027.

[48] Anonymous (2010) 1516.1-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. DOI: http://dx.doi.org/10.1109/IEEESTD.2010.5557728

[49] Ding D. & Zhu M. (2003) A Model of Dynamic Interest Management: Interaction Analysis in Collaborative Virtual Environment. Proc. ACM Symposium on Virtual Reality Software and Technology, Osaka, Japan, 223–230. DOI: http://dx.doi.org/10.1145/1008653.1008692.

[50] Abrams H., Watsen K. & Zyda M. (1998) Three-Tiered Interest Management for Large-Scale Virtual Environments. Proc. ACM symposium on Virtual reality software and technology, New York, NY, USA, 125-129. DOI: http://dx.doi.org/10.1145/293701.293717.

[51] Oliveira J. & Georganas N. (2003) VELVET: An Adaptive Hybrid Architecture for Very Large Virtual Environments. Presence: Teleoperators and Virtual Environments 12(6): 555–580. DOI: http://dx.doi.org/10.1162/105474603322955888.

[52] Singhal S. & Cheriton D. (1996) Using projection aggregations to support scalability in distributed simulation. Proc. 16th IEEE International Conference on Distributed Computing Systems, Hong Kong, 196–206.

[53] Benford S, Greenhalgh C. (1997) Introducing third party objects into the spatial model of interaction. Proc. Fifth European Conference on Computer Supported Cooperative Work, Lancaster University, UK, 189–204.

[54] Anonymous (2014) Time Dilation – EVElopeda. URL: https://wiki.eveonline.com/ en/wiki/Time_Dilation. Accessed 09.05.2014.

[55] Anonymous (2014) realXtend/Tundra. URL: https://github.com/ realXtend/ tundra. Accessed 24.04.2014.

[56] Anonymous (2014) karivatj / kNet SCTP – Bitbucket. URL: https://bitbucket.org/ karivatj/knet-sctp. Accessed 24.04.2014.

[57] Anonymous (2014) Qt – Cross-platform application and UI development framework. URL: http://qt.digia.com/. Accessed 24.04.2014.

[58] Anonymous (2014) OGRE – Open Source 3D Graphics Engine. URL: http://www. ogre3d.org/. Accessed 24.04.2014.

[59] Anonymous (2014) MathGeoLib. URL: https://github.com/juj/ MathGeoLib. Accessed 24.04.2014.

[60] Anonymous (2014) kNet. URL: https://github.com/juj/kNet. Accessed 24.04.2014.

[61] Anonymous (2014) Bullet. URL: http://bulletphysics.org/wordpress/. Accessed 24.04.2014.

[62] Jylänki J. (2012) Tundra Optimization Backlog Items. URL: https://groups.google.com/d/msg/realxtend-dev/wNTmx3keCfw/ 4gNz12Z6ypkJ. Accessed 27.04.2014.

[63] Jylänki J. (2012) RealXtend Tundra 2.3.0 is released! URL: https://groups.google.com/forum/#!topic/realxtend-dev/c3s8tQnKn_s. Accessed 27.04.2014.

[64] Jylänki J. (2012) Scalability study for Tundra (2012). URL: https://groups.google.com/d/msg/realxtend-dev/Lzzx_hZu38I/ XIXsl87GdZUJ. Accessed 27.04.2014.

[65] Anonymous (2014) Tundra 2.4. URL: https://github.com/realXtend/ naali/tree/tundra2.4. Accessed 24.04.2014.

[66] Anonymous (2014) LudoCraft/Tundra at InterestManagement. URL: https://github.com/LudoCraft/Tundra/tree/InterestManagement. Accessed 24.04.2014.

[67] Aldridge D. (2011) I Shot You First! The Gameplay Networking of Halo: Reach. Game Developers Conference.

[68] Anonymous (2014) Terrain LOD Published Papers. URL: http://vterrain .org/LOD/ Papers/. Accessed 24.04.2014.