A comparison of Quad-tree and Voronoi-based spatial partitioning for dynamic load balancing

M van Greunen
MIH Media Lab
Stellenbosch University
Stellenbosch, South Africa
Email: manrich@ml.sun.ac.za

H.A. Engelbrecht
MIH Media Lab
Stellenbosch University
Stellenbosch, South Africa
Email: hebrecht@ml.sun.ac.za

***Abstract*—Massively multi-user virtual environments (MMVEs) face scalability challenges, one being the large number of concurrent users interacting in the virtual environment (VE). Spatial partitioning addresses this problem by distributing partitions of the VE, and their associated users, to separate servers. Users dynamically migrate between partitions as they move within the VE and server load imbalances occur when users flock to popular locations (such as cities or boss arenas). Dynamic Load Balancing can be achieved by dynamically scaling the VE partitions and migrating users to underloaded servers. In this paper, we assume an MMVE has load balancing and focus on comparing two spatial partitioning methods, namely Quad-trees and Voronoi diagrams, using OverSim, an extension of the OMNeT++ simulation package. We evaluate each approach using the number of messages sent between servers, the distribution of users across servers and the number of servers in use as performance metrics. We conclude that a Voronoi based system is better in distributing the load across multiple servers, but has a greater computational cost than a Quad-tree based system.**

I. Introduction

Interest in Massively multi-user virtual environments (MMVEs) has increased dramatically over the past few years, particularly in multimedia and online gaming. MMVEs enables multiple users to assume virtual representations called *avatars* to interact and socialise inside a virtual world. Client/Server architectures are predominantly used for providing the resources required to host an MMVE from a central location. However, as the number of concurrent users of MMVEs increase, a single server is not capable of reliably providing the MMVE functionality. One solution to address this problem, is to distribute the load of hosting the MMVE to server clusters. Companies that develop and host MMVEs (such as *World of Warcraft* and *Diablo 3*) invest in large server infrastructures to provision for dynamic loads. Cloud Computing (or On-Demand Computing) now enables independent game developers, especially South African game developers to develop and host MMVEs on a distributed server cluster in the cloud, without the need for large upfront investments. The cloud could dynamically scale according to the number of users, ie. the load of the MMVE and be able to operate at the scale of other AAA game development companies' titles.

Using distributed server clusters introduces a number of challenges, such as how to distribute the virtual environment (VE) and keep the state of the MMVE consistent amongst the server nodes. Prominent approaches that address these challenges include *sharding*, *zoning* and *spatial partitioning* of the virtual environment. Sharding of the VE entails that the VE be duplicated and hosted on individual servers, whereby zoning entails that separate regions of the VE be hosted on separate servers and these regions be connected with so called 'portals' inside the virtual world. Both of these methods has a disjointed VE, with inter-server latencies causing users, connected to different server nodes, unable to interact and modify the game state in a *consistent* way.

Spatial partitioning divides the virtual environment (VE) into separate regions. Each region is hosted by a single server and that server manages the state of all entities within its region. Entities can move within the VE and are assigned to a new server node when crossing the border between regions, effectively migrating the state of the entity. An avatar can directly modify the state of an entity or indirectly cause the state of an entity to be modified. Both direct and indirect interactions consume resources on the server node, in terms of bandwidth and computational power. In this paper we do not make a distinction between direct and indirect interactions, but as a simplification, regard the number of avatars in a region as indicative of the load of that server node. By varying the size of regions assigned to each server node, the total load of the server cluster can be distributed. We investigate two spatial partitioning algorithms, namely Quad-trees [1, p. 307] and Voronoi diagrams [1, p. 147], for partitioning the VE in a Dynamic Load Balancing (DLB) system. The particular contributions of this paper is in the evaluation and comparison of the load balancing efficiency of both algorithms in terms of network bandwidth, server utilisation, user count and computational cost.

II. Related Work

Yu and Vuong [2], developed a Mobile Peer-to-Peer Overlay Architecture (MOPAR) for interest management of MMOGs that use *zoning*. By dividing the virtual environment into same-sized hexagonal zones and assigning each to a server node, the virtual world is distributed and scalability is achieved. By utilising an Area of Interest (AoI) neighbouring scheme together with virtual locations, each node only handles updates from its neighbours. They focused on guaranteeing all clients have consistency across zones within the virtual environment [2]. Yu and Vuong do not show experimental or simulated results, but their analysis only states the following: "MOPAR uses less nodes, uses resources more effectively

 ISBN: 978-0-7972-1533-7

and is more fault-tolerant than other Peer-to-Peer interest management schemes." Backhaus and Krause [3], proposes QuON: a Quad-tree based Overlay Protocol for distributed virtual worlds. QuON sends game event messages to a client's neighbours, defined by an AoI. Because update-messages are only sent to nodes which will be affected, the load is distributed among all peers. State consistency and security is achieved by organising neighbours into Quad-trees. QuON is shown to be a practical and effective solution over C/S networks based on OverSim simulation results [3]. Hu and Chen [4] presents VSO: a Voronoi-based Self-organising Overlay (VSO), which dynamic balances the load using spatial partitioning. Neighbouring nodes are organised into Voronoi diagrams and using Spatial Publish/Subscribe (SPS) clients are able to subscribe to a Voronoi region that is the responsibility of a server node. VSO measures the load of a specific node, based on the number of subscribers to that node's region. When a specific node is overloaded it requests a neighbouring node to move its virtual location to resize the enclosed area. Clients are unsubscribed from the overloaded node and ownership is transferred to neighbouring nodes. VSO evaluations shows an improvement in bandwidth usage of up to 95% when comparing static and dynamic partitioning for up to 500 nodes. In terms of scalability VSO shows that for 1000 nodes, 90% of servers are still functioning under an 'acceptable load' as specified by Hu and Chen [4].

III. System Model

The system model of an MMVE consists of multiple users that dynamically connect to a VE and generate events that include the movement of their avatar, the manipulation of VE objects and interaction with server-controlled entities (called non-player entities or NPEs) [5]. These events are sent to the server which updates the global game state using server-side game logic, generating state updates, that are then distributed to all subscribed clients. According to Carlini et al. [6], the management and handling of events created by user avatars and passive entities, constitutes the typical computational and bandwidth load of a distributed VE.

As previously mentioned, spatial partitioning divides the VE into regions and distributes these regions to server nodes that are responsible for the game state of that region. A client/server architecture that supports spatial partitioning is shown in Fig. 1. It consists of four parts: (1) distributed servers that are connected in a server cluster, (2) a partitioned VE (3) clients and (4) a directory server.

The **Client** is a representation of a user within the VE. It has a virtual location within a particular region and the server hosting that region is responsible for handling its state updates. A client can change its virtual location incrementally and is *migrated* when it moves across a border between regions. *Migration* is defined as the transfer of the responsibility of a client from one server to another. Migration requires communication between the servers and the migrating client so that state can be transferred and the state of the VE kept consistent.

A **Server** is responsible for hosting a region within the VE. It has a virtual location, a list of clients it owns and a list of server neighbours within the VE. A server can measure its load and determine its *ownership* of a specific client, defined as that client being within the server's region.

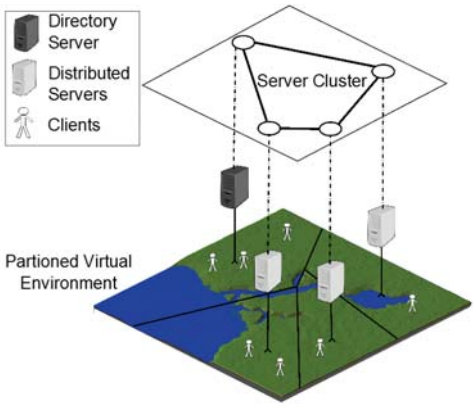

Fig. 1. The spatial partitioning client/server architecture: Distributed servers are connected in a Server Cluster and each is responsible for a region in the Partitioned Virtual Environment and all Clients in that region.

The **Directory** server is used as the entry point for clients connecting to the VE and for new servers joining the distributed server-cluster. Initially the *Directory* server is the only server and it owns the entire VE. The typical role of a directory server is to verify users accounts and redirect clients to servers hosting the VE, but for the purpose of this paper we assume that this does not have a significant contribution to the load.

IV. Dynamic Load Balancing

Dynamic Load Balancing require that the regions can be dynamically resized and that clients can be migrated to underutilised servers. We define two states of server load, namely **overloaded** and **under-loaded**. For the purpose of this paper, a server is deemed to be overloaded when the number of clients it owns exceeds a predetermined overload threshold. A server is deemed to be under-loaded when the number of clients it owns is lower than the under-load threshold.

When an active server enters the overloaded state, it migrates its load by setting up a new server. The new server is provided with a virtual location and region to host. All neighbouring servers are also notified of the new server so they the servers an adjust their regions to accommodate the region of the new server. Clients that are now within the region of the new server is migrated to that server. When an active server enters the under-loaded state it returns the region it is hosting to its neighbouring servers. Clients are migrated to the neighbouring server hosting the region where its avatar resides.

There are two issues that still need to be addressed for dynamic load balancing: (1) distributed computation of the VE partitioning and (2) implementation of dynamically load balancing by resizing and migrating VE regions between servers. Both of theses issues will be discussed when we discuss the detail of Quad-trees and Voronoi diagrams in the rest of this section.

A. Distributed computation of Quad-trees partitioning

A Quad-tree is a rooted data structure starting with the first (parent) node and is characterised by each parent node

 ISBN: 978-0-7972-1533-7

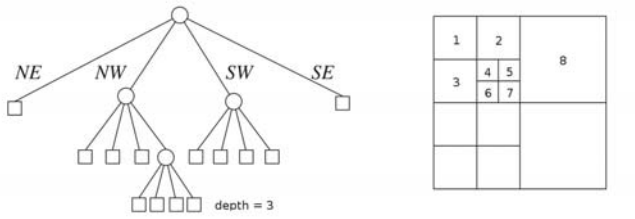

Fig. 2. The Quad-tree structure [7].

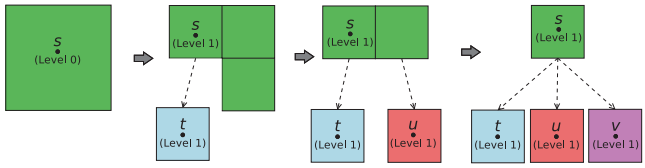

Fig. 3. Quad-tree partitioning of the VE. Partitioning level limited to level 2.

having four child nodes. In the case of two-dimensional VE partitioning, each server node owns a square in the VE, with each child node of that server responsible for a quadrant of its square (see Fig. 2). Using the recursive property of Quad-trees one can construct an algorithm for computing a Quad-tree as follow. For any server node n, divide its square into four squares, calculate each square with side length half of n's original side length and find all neighbours of n.

This same property can be used to distributively subdivide each server node's own square. In order for the global Quad-tree structure to be kept intact, a *Parent-child relation* is defined by: each distributed child node keeping a reference to its parent node (i.e. the server node that created it).

Quad-tree neighbours are typically defined as server nodes that share a edge [1, p. 313], but in this paper we include neighbours that share a vertex. In order for the child node to determine its neighbours, it evaluates its parent's neighbour- and child nodes. Fig. 2 illustrate node 4's edge neighbours as, $\{2, 3, 5, 6\}$. But including node 4's vertex neighbours it becomes, $\{1, 2, 3, 5, 6, 7\}$. Server node 1 should include server node 4 as a neighbour, because the squares regions represent the VE and if a user's avatar move diagonally (from 1 to 4) across the border, it should be migrated.

B. Quad-tree DLB implementation

To implement dynamic load balancing servers need to be able to migrate VE regions, which is achieved by using the previously mentioned *Parent-child relation* as shown in Fig. 3. Each *Child* server keeps a reference to the *Parent* server that created it. The *Directory* server s is the root node and first Parent node of the Quad-tree and is initialised with the entire VE at level 0. When the *Directory* server s determines that it is overloaded, it partitions its region into four squares and sets up a new child server t, increasing its own level and assigns one of its four squares to t. If s becomes overloaded again, it sets up a new child server u and assigns it one of its squares regions. This procedure continues until s owns just one smaller square region. This Quad-tree partitioning scheme is one of many possible methods. Some of these methods allows for regions to be made-up of a combination of squares (including different level squares). This creates complex regions and intricate neighbouring schemes. Our approach is similar to the one used in Backhaus's QuON [3] and makes for a simpler neighbouring scheme.

In our Quad-tree DLB system we limit the partitioning level to 2, preventing the number of neighbours to increase indefinitely, but be limited to a maximum of eight neighbours. The Quad-tree in itself has an unlimited number of levels. A *Server* discovers its neighbours by determining if it shares an edge or a vertex with any of its parent's neighbours or other child servers.

C. Distributed computation of Voronoi diagrams

A Voronoi diagram in two-dimensions, is a tessellation of a plane into a set of polygons $V_1, ..., V_n$, associated with the nodes $v_1, v_2, ..., v_n$ respectively. These polygons are termed Voronoi cells [8] and are calculated using the following (taken from [1, p. 149]):
1. For two nodes v_p and v_q and their associated locations p and q in the plane we calculate the *bisector of p* and q as the perpendicular bisector of the line segment $\overline{pq}$. This gives two half-planes, denoted by *h(p,q)* and *h(q,p)* containing the points p and q respectively.
2. Calculate n-1 half-planes where n is the number of nodes.
3. The half-plane intersections define a region that is bounded by a convex polygon, with at most $n-1$ vertices and at most $n-1$ edges.

Algorithms for calculating Voronoi diagrams that use the coordinate information about *all* nodes in the plane and are termed full information algorithms. The Voronoi diagram can be calculated in a distributed manner by each server node v_i using a full information algorithm such as Steve Fortune's sweep-line algorithm [9], but limiting the calculation to only using the server's own VE location and the VE locations of its *relevant* neighbours. A *relevant* neighbour for server node v_i is defined as a node that will change v_i's Voronoi cell if included in server v_i's Voronoi diagram calculation.

D. Voronoi DLB implementation

The Voronoi DLB implementation is best explained using a typical case as shown in Fig. 4. If s is overloaded, it sets up a new server t and assigns it a location within the VE. For the purpose of this paper, we chose the virtual location as the geometric centroid of the locations of s's clients. Server s should transfer some of its clients to t and reduce its own load, thus we assume that choosing the client-locations centroid will suffice in satisfying this requirement. If s becomes overloaded again, a new server u is set up and both s and t determines if u is a *relevant* neighbour and recalculates their Voronoi cells. To determine the *relevant* neighbours we first need to determine the set of possible neighbours from which the *relevant* neighbours is a subset. Fig. 5 illustrates our neighbouring scheme: Server s owns a Voronoi cell $V(s)$, calculated using s's location and its current neighbour locations $\{t, u, v, w\}$. We define a circle centred at s with radius r, where r is the maximum distance to any vertices p in $V(s)$.

$$r = \max_{p \in V(s)} [dist(p, s_{loc})] \quad (1)$$

 ISBN: 978-0-7972-1533-7

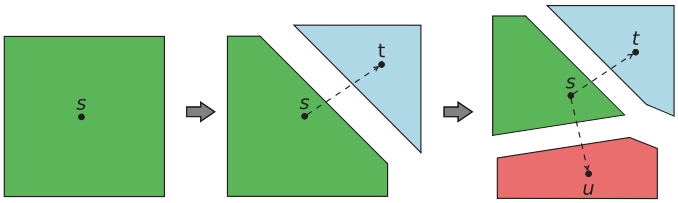

Fig. 4. Voronoi partitioning of the VE.

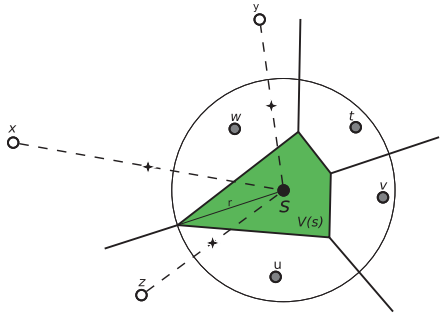

Fig. 5. Voronoi neighbouring scheme: Determining if x, y, or z are possible neighbours of s.

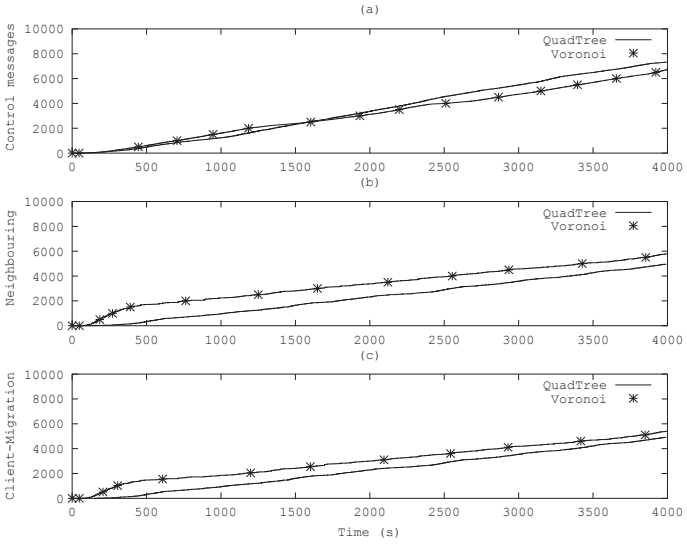

Fig. 6. Quad-tree vs. Voronoi: The number of (a) *Control*, (b) *Neighbouring* and (c) *Client-Migration* messages.

Next, s tries to eliminate x, y, or z as possible neighbours, by calculating the midpoint between it and the possible neighbour. If the midpoint is outside its circle with radius r, the server is discarded as a possible neighbour, e.g. in the case of x. If the midpoint is inside s's circle, the server is kept as a possible neighbour.

The same test is executed for y and z and both results in s including them as possible neighbours. When referring back to the definition of a *relevant* neighbour, a server node that will affect $V(s)$, y should be excluded from s's list of neighbour. Up to this point, s has no way to determine whether both y and z are *relevant* neighbours, but as Cao [8] stated, including y as an additional node does not affect the Voronoi calculation of $V(s)$ except by increasing computational costs. For the purpose of this paper we will accept the additional cost of including irrelevant neighbours.

V. Evaluation

In this section we present the evaluation of both DLB systems. The purpose of this evaluations is to determine which of the DLB systems is more efficient at balancing the load across a distributed VE. We use the following to evaluate the performance of the DLB systems i.e. *bandwidth*, *effectiveness*, *fairness* and *partitioning computational cost*.

A. Metrics

Bandwidth: Network bandwidth is measured as the number of inter-server and server-client messages sent and the scheme with less messages requires less bandwidth and thus increases the MMVE user experience. Inter-server messages are divided into *Control* and *Neighbouring* messages while *Client-Migration* messages are server-client messages, i.e.:
- Control messages are used to indicate a overloaded server, under-loaded server or to request a new server be started.
- Neighbouring messages are used to notify about neighbour changes.
- Client-Migration messages are used to migrate a client to a neighbouring server.

Effectiveness: The effectiveness of the DLB scheme is determined by the distribution of the number of clients owned by each server.
Fairness: The number of servers in use, overloaded or available (i.e. under-loaded) indicates the *fairness* of each DLB system. A *fair* system is one where the servers in use have a similar load.
Computational cost: The per server computational cost of each partitioning algorithm gives an indication of the additional computational burden of the DLB scheme.

B. Simulation parameters

Simulations are performed in OMNeT++ with OverSim [10], a discrete event simulator for peer-to-peer networks which simulates real-world network conditions and enables simulations to be executed on physical server clusters. We set the VE dimension to $10,000 \times 10,000$ units and a client's movement speed is set to 10 units/second, which is comparable with typical MMVE environments and movement (i.e. *World of Warcraft*). Clients are added to the VE at the *Directory* server's location, every 5 seconds up to a maximum of 40 clients. We define overload threshold as 5 clients or more. The number of available servers are limited to 10 instances. The length of simulation is set to 4,000 seconds. We assume this to be a sufficient time for the system to become fully loaded and reach a steady state. We use Pastry as a peer-to-peer communication overlay to simplify network communication, but for the purpose of the simulation the specific communication scheme is not important.

C. Simulation Results

The network bandwidth results for both DLB schemes are shown in Fig. 6. Fig. 6(a) shows that both DLB schemes send a similar number of *Control* at the beginning of the simulation.

 ISBN: 978-0-7972-1533-7

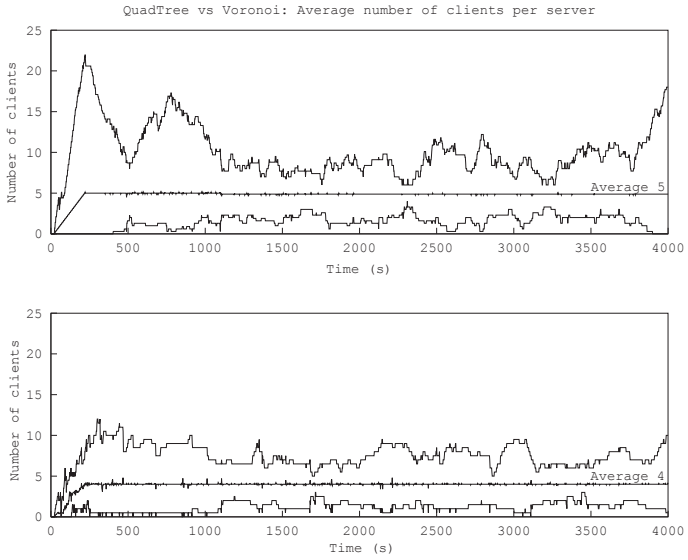

Fig. 7. Quad-tree(top) vs. Voronoi(bottom): The average number of clients per server.

At 2500 seconds the Quad-tree scheme's *Control* messages increase, possibly indicating the the Quad-tree servers are overloaded. The reason it's more than the Voronoi server's, could because of the limitations on our Quad-tree structure, only being divisible to level 2 and thus causing a server to remain overloaded when reaching this level. Fig. 6(b) shows significantly more *Neighbouring* messages for Voronoi DLB up to 400 seconds as compared to Quad-tree DLB. This can be contributed to Voronoi's dynamic behaviour to resize regions in response to the changing load. The increase in *Client-Migration* messages seen in Fig. 6(c) can also be attributed to this rapid response to load increase. It can be seen that Voronoi DLB results in less *Control* messages being sent, but that more *Neighbouring* and *Client-Migration* messages are needed that for Quad-tree DLB.

Fig. 7 shows the minimum, maximum and the average number of clients per server over time, for both Quad-tree (top) and Voronoi (bottom) DLB systems. We observe that Quad-tree DLB has a higher maximum users count per server and a larger spread compared to the Voronoi DLB system. The average number of five clients per Quad-tree server is also higher than the average of four clients per server for Voronoi DLB. This indicates that Voronoi DLB is more effective than Quad-tree DLB, since all Quad-tree servers are not being utilised. However, this could be as a result of the previously mentioned limitation imposed on the Quad-trees. We also note that Voronoi DLB is more fair than Quad-tree DLB, since the spread of the number of clients per server is smaller.

Table I shows the *Computational cost* of each partitioning scheme. We have count the number of computations, measure the calculate-time of a single partitioning and the total time spent calculating partitioning during the simulation. To ensure accuracy of the results, each algorithm is run 10,000 times and the average time used. We observe much higher computational cost for calculating Voronoi diagrams as compared to calculating Quad-tree partitioning. The Voronoi partitioning algorithm takes longer and is calculated more frequent than Quad-tree

Algorithm	Comp. time	No. of calc's	Total cost
Voronoi	*33 ms*	*4953*	*163.45 s*
Quad-tree	*0.96 ms*	*29*	*28 ms*

TABLE I. COMPUTATIONAL COST.

partitioning. This results in Voronoi DLB having a computation cost of 5837.5 times more than Quad-tree DLB. We deem the average computational time of 33ms to be acceptable, since Carlini et al. states that the computational cost of a partitioning algorithm should simply be acceptably low [6]. We assume that an acceptable range is less than 50ms, half of the typical acceptable MMVE of 100ms [11].

VI. Conclusions and Future Work

In this paper we have shown that, for dynamic load balancing of distributed MMVEs, using Voronoi diagrams for spatial partitioning is significantly more computationally expensive than Quad-trees, and also use more network bandwidth. However, Voronoi-based DLB system is more effective in utilising the available servers, because it can assign dynamic sized regions in comparison to the Quad-tree DLB system which has limitations on when a region can be divided as well as returned to a parent node. It has also been shown that Voronoi DLB is more fair, since it results in an average of less clients per region.

In future we plan on enabling Voronoi-based servers to move their virtual location to resize the region they own, as Hu and Chen [4] proposed as well as investigate the effect on Quad-tree effectiveness when allowing Quad-tree based servers to subdivide their regions into more levels. We also want to verify the simulation results by implementing both DLB schemes in a real MMVE such as Minecraft.

References

[1] M. de Berg, M. van Kreveld, O. Cheong, and M. Overmars, *Computational Geometry, Algorithms and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[2] A. Yu and S. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," *(NOSSDAV'05)*, pp. 99–104, 2005.

[3] H. Backhaus and S. Krause, "QuON: a quad-tree-based overlay protocol for distributed virtual worlds," *Int. Jo. of Advanced Media and Communication*, vol. 4, no. 2, p. 126, 2010.

[4] S.-Y. Hu and K.-T. Chen, "VSO: Self-Organizing Spatial Publish Subscribe," *2011 IEEE 5th Int. Conf. on Self-Adaptive and Self-Organizing Systems*, pp. 21–30, Oct. 2011.

[5] J. S. Gilmore, "A state management and persistency architecture for peer-to-peer massively multi-user virtual environments," Doctor of Philosophy, Stellenbosch University, 2013.

[6] E. Carlini, L. Ricci, and M. Coppola, "Flexible load distribution for hybrid distributed virtual environments," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1561–1572, Aug. 2013.

[7] K. Buchin, "Geometric Algorithms Lecture 9 : Quadtrees," 2013.

[8] M. Cao and C. Hadjicostis, "Distributed algorithms for Voronoi diagrams and application in ad-hoc networks," *Preprint, Oct*, pp. 1–12, 2002.

[9] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, pp. 153–174, 1987.

[10] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proc. of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007, pp. 79–84.

[11] A. Yahyavi and B. Kemme, "Peer-to-Peer Architectures for Massively Multiplayer Online Games : A Survey," *ACM Computing Surveys*, vol. 46, no. 1, 2013.