2010

# PVW: Designing Virtual World Server Infrastructure

Francis Chang
*Portland State University*, chang.francis@gmail.com

C. Mic Bowman
*Intel*

Wu-chi Feng
*Portland State University*

# PVW: Designing Virtual World Server Infrastructure

Francis Chang*, C. Mic Bowman†, Wu-chi Feng*

*Department of Computer Science, Portland State University, Portland, OR
wuchi@cs.pdx.edu

†Intel Research, Intel Corporation, Hillsboro, OR
francis@francischang.com, mic.bowman@intel.com

## ABSTRACT

*This paper presents a high level overview of PVW (Partitioned Virtual Worlds), a distributed system architecture for the management of virtual worlds. PVW is designed to support arbitrarily large and complex virtual worlds while accommodating dynamic and highly variable user population and content distribution density. The PVW approach enables the task of simulating and managing the virtual world to be distributed over many servers by spatially partitioning the environment into a hierarchical structure. This structure is useful both for balancing the simulation load across many nodes, as well as features such as geometric simplification and distribution of dynamic content.*

**Keywords**: Metaverse architecture, 3D Virtual Worlds, Partitioned Virtual Worlds, Massively Multiplayer

## 1. INTRODUCTION

Virtual reality systems [aw,croquet,sl,os] have risen in popularity with readily available high-speed networking and affordable consumer computer graphics processing hardware.

This paper focuses on metaverses – a shared 3D virtual space in which people can interact and communicate through virtual avatars. Unlike massively multiplayer online games (MMOGs) which strive to simplify their universe to optimize their implementation for a specific game environment, metaverses are characterized by a generalized approach to the problem of 3D worlds. These designs seek to promote unconstrained user-generated content for services such as social networking and collaboration, scientific experimentation, e-commerce, marketing and gaming.

The unconstrained nature of metaverses requires a different style of architecture to manage computing and networking resources than online gaming.

This paper introduces PVW (Partitioned Virtual Worlds), an architecture designed with the goals of managing 3D virtual space and content in a client-server situation.

In the following section, we describe some of the goals of our architecture. Related work in is outlined in Section 3. Section 4 describes our architecture and its algorithms. Section 5 discusses implementation notes and properties of PVW while Section 6 introduces extensions to this architecture to support different spatial topologies and administrative requirements.

## 2. DESIGN GOALS

The following are design considerations for our architecture:

- The design must be a client/server architecture. In this way, the service provider can guarantee security, availability and adequate resource provisioning.
- Storage and computing power is large, but no single computer can handle the computing load.
- Clients have relatively small computing and network resources. Servers must simplify the world state for each connected client.
- The virtual environment is a free form universe, and cannot make strong assumptions about the type of content.
- The world is dynamic and constantly changing and expanding. User generated content is a fundamental component of metaverse development, and we cannot rely on pre-downloaded content.
- There will be many metaverses, both persistent and temporary. Even though these metaverses will be part of different administrative domains, there must be a way to link and embed them in a logical manner.
- The population is large, and unpredictable. The architecture must accommodate flash crowds as well as vast unused or unpopulated spaces.

It is the goal of PVW to be an architecture for metaverse-like entities and to be a foundation for all types of MMO virtual simulations including online gaming and 3D social networks.

### 2.1 Elements Not Part of PVW

In designing any large multi-user system there are many architecture constructs that are only weakly tied to the problem of managing 3D virtual space. Components such as asset storage, user profiles, authentication, exploit detection, domain administration and instant messaging are not discussed in this paper. These problems can be addressed in by more general system solutions that are not encumbered by the constraints of managing a metaverse-style universe.

The PVW architecture only addresses the problems of managing, streaming and connecting 3D virtual space and the objects contained within.

# 3. RELATED WORK

There are many examples of massively multiplayer virtual spaces that each have distinct solutions to the problem of managing vast virtual spaces that need to service a high number of simultaneous clients.

In MMOGs, sharding is a popular approach to broadly partition the user base into disjoint copies of the world. In this model, replication is easy because users belonging to one shard cannot interact with users in other shards [uo,wow]. Load balancing is accomplished by restricting the number of simultaneous users in a shard. In these environments, only a minimal amount of functionality is placed at the server to allow them to scale up. For instance, generalized physics and dynamic content are usually omitted.

Croquet [croquet] is a decentralized approach to the problem of virtual spaces relying on a peer-to-peer synchronization protocol to distribute the contents of the virtual space. A single croquet instance can become congested with many simultaneous users since there is no mechanism to subdivide existing space.

Active Worlds [aw] is another metaverse-like virtual world that allows dynamic content creation, including a simplified scripting interface. The Active World universe hosts hundreds of worlds which can be traversed by users, where each world is hosted on a single server.

Second Life [sl,kumar] and its open-source counterpart OpenSimulator [os] are metaverse-like worlds that allow users to explore and create a dynamic 3 dimensional space. This space is partitioned into square 256x256m regions, each managed by a separate sim process. Each sim is tied to a specific region of land, and cannot be repartitioned to react to a changing workload. This is the primary reason that scaling up is such a difficult problem in this architecture. Larger spaces are created by placing sims adjacent to one another. Shards or instancing is not supported.

Different topologies of fixed grid spatial subdivision have been explored, such as triangular, square, hexagonal and brickworks [presetya]. These systems are not as scalable as spatial subdivision approaches using hierarchical grids. Either dynamic resource allocation is not present, or it involves moving server processes around so that unloaded servers can time-share a single CPU.

The Project Darkstar (Sun Gamer Server Technology framework) approach to accommodating massive world state avoids spatial subdivision in favour of storing object and world state in a massive database. Actions on objects are performed through the database.

ALVIC approaches metaverse design by using quad-tree subdivision for partitioning logic servers and employing many proxy servers to hide the network topology from clients [quax].

## 3.1 Algorithms from Computer Graphics

PVW borrows fundamental tree data structures from computer graphics. All modern ray-tracers rely on acceleration structures to manage scene and world data to minimize computationally expensive collision and lighting calculations.

One classic approach to this problem is to divide space into hierarchical bounding volumes (HBV) [rubin]. In this approach, the 3D space is divided into rectangular prism hierarchies and arranged in a tree structure. Child nodes represent space encompassed by the parent, with leaves being atomic renderable objects such as triangles and spheres.

kd-trees are a more restrictive type of spatial partitioning, only allowing partitioning planes to subdivide space, perpendicular the canonical 3-space axis, resulting in a binary space partitioning (BSP) tree. This data structure is successfully used in modern ray-tracing algorithms [reshetov].

Extending these ideas, a recent contribution to the area is the idea of bounded interval hierarchies (BIH) which modifies kd-trees allowing two split planes per descendent [wachter]. This added flexibility allows us to explicitly encode overlapping regions into our BSP tree.

# 4. The PVW Tree

The core design motivation of PVW is the assumption that no single computer has enough resources to manage the entire metaverse simulation. PVW provides a convenient load splitting and management mechanism to distribute computation over a set of servers.

At the core of the PVW architecture is the PVW tree. The PVW tree is very similar to a BIH tree discussed in Section 3.1. The most significant difference between the PVW and the BIH tree is that leaves in a PVW tree represent virtual 3D spaces instead of objects. Each node in the tree, including interior nodes and leaves, is managed by a separate server process. Just as in all HBVs, parent nodes must completely encompass the space occupied by child nodes.

The root node in the PVW tree represents a simulation process (sim) managing an entire PVW universe. To distribute the workload of managing the PVW universe, each node can divide its managed space in two, and pass off the processing to two child nodes.

Just as in BIHs, the space managed by the child nodes is expressed by two partition planes, aligned perpendicularly to either the *x*, *y* or *z* axis. The left child is responsible for managing all objects on one side of the first partition plane while the right child is responsible for managing all objects to the opposite side of the second partition plane.

Partition planes must be chosen to balance the load and ensure that all objects are fully enclosed within a child sub-volume.

Figure 1 illustrates the recursive construction of a PVW graph in a 2 dimensional Cartesian space.

This dual-partition structure has several benefits:

1) It allows us to divide space without duplicating or partitioning objects. In Figure 1b, objects k and i cross the first left child boundary, but are fully enclosed by the right child.

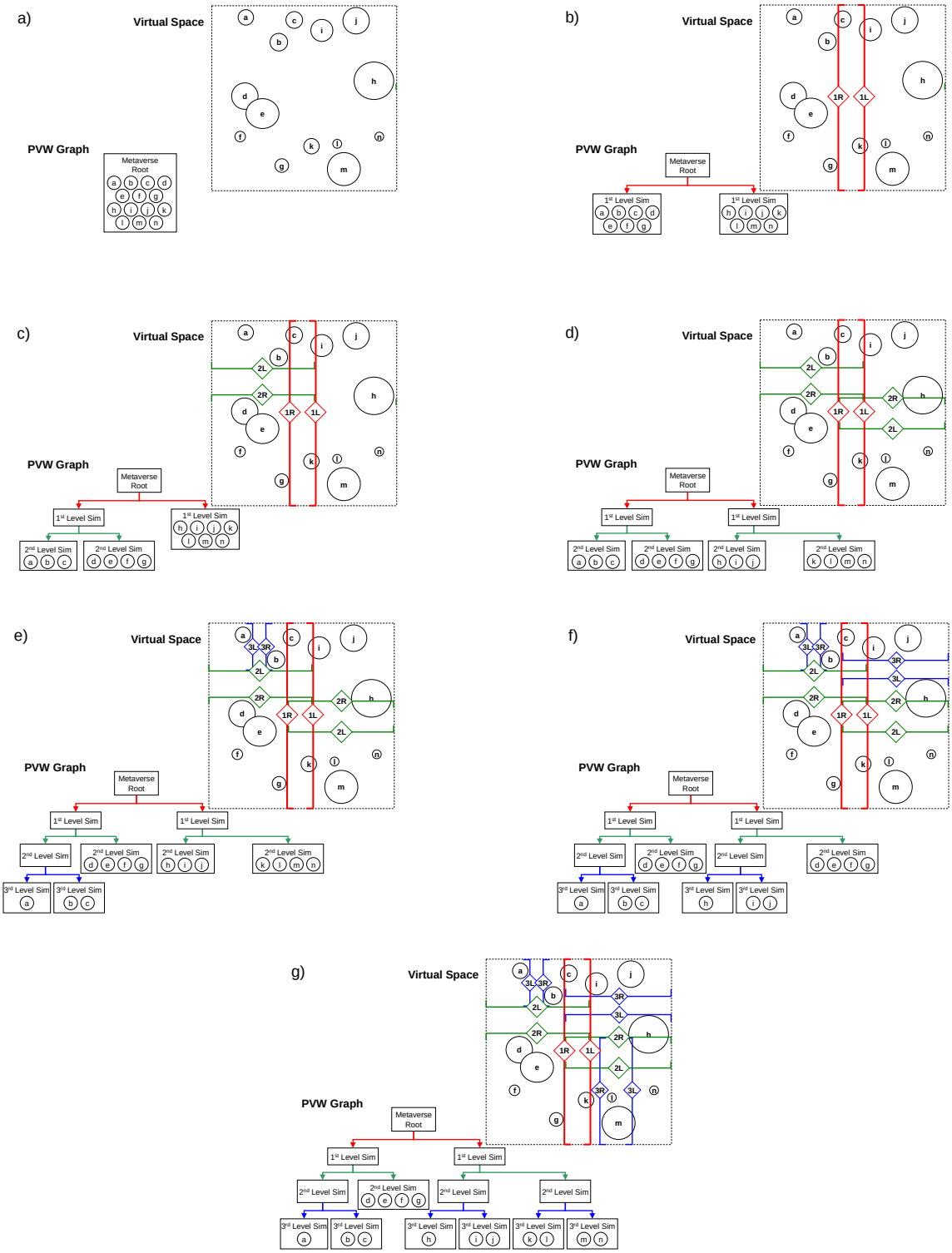


Figure 1: Recursive PVW partitioning of the virtual space, as well as a graph representing the partitioning. Each circle represents an object in virtual space, while each box represents a separate server process managing the space. Note in Figure c) that empty space does not necessarily need to be managed by a specific process, while in figure g) object l could be managed by either of two different sim processes.

2) It allows us to express empty and unmanaged regions. In Figure 1c, we've partitioned the space in a way that there is empty space between two sibling PVW nodes.

3) It allows us to express transitional objects. In Figure 1g, object "l" is fully enclosed by the left and right PVW nodes. This object could be in the process of being moved between the two peers sims.

4) It allows us to dynamically resize and manage the space without modifying the topology of the tree, which is an expensive operation. If an object inside a volume moves closer to the boundary of a PVW node, it may be more convenient to simply move the node boundary rather than to transition the object to another node.

# 5. PVW PROPERTIES

A property of PVW volumes that is inherited from the BIH-derived structure is that all objects will be fully enclosed by a bounding sub-volume. This is an important property, because it allows us to assign processing of objects to a hierarchy of sims. Leaf nodes are responsible for the direct processing of objects in their enclosing volume, while parents are responsible for shadowing the state managed by its direct children.

## 5.1 Load Balancing and Splitting

The most significant motivation to PVW design is the need to divide and distribute processing load of a metaverse over many servers. The two most significant operations in managing PVW systems are node splitting and joining.

When a simulation process is overwhelmed by an implementation-specific definition of load, it can choose to split its workload between two child sims (Figure 1). For this operation, the PVW system will need to assign two servers (from a pool of idle simulators) to the task, and give them each a portion of the simulation state to manage.

The converse operation is much simpler – when two sibling leaf simulators have a small workload, they can choose to simply synchronize state and revert processing to their parent. The now vacated child sims can rejoin the pool of idle simulators.

In PVW, the partitioning borders between sims are dynamic and reactive to the workload. In the case where two neighbouring nodes in a PVW tree have an unbalanced workload, one child can grow while the other shrinks to distribute the workload evenly between the two nodes. In this manner, the PVW tree is constantly rebalancing itself, and avoids the long-term unbalancing problems associated with KD trees.

## 5.2 Geometric Simplification

One of the challenges of designing a functional metaverse is creating a system that can manage a vast collection of objects while still being able to simplify the world in a way that can be streamed to a more bandwidth-restricted client.

The canonical example of this behaviour is streaming avatars – at speaking distances, we prefer describing avatars as individuals, but at a stadium level it is more appropriate to describe avatars in terms of crowds.

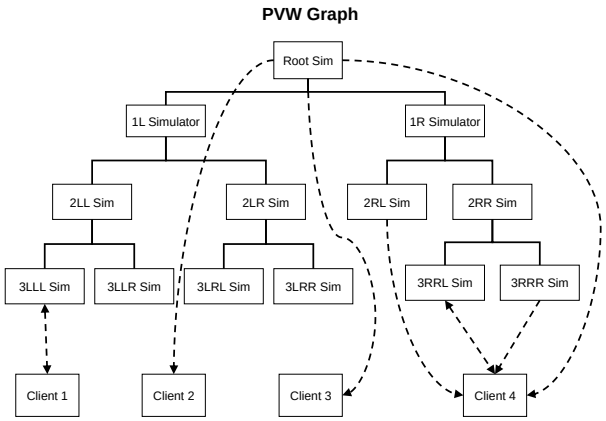


Figure 2: Network connections in a PVW instance. Dotted lines represent network connections, arrows represent information flow.

In PVW, each simulator is responsible for generating a simplified understanding of the volume they represent (e.g. a 64KB or smaller representation). This representation can take the form of a textured mesh, a skybox, a voxel cloud, a 3D texture, a collection of pictures or some combination thereof.

To maximize the utility of available bandwidth, the simplified understanding of the volume can also be viewer dependent. For example, in an ocean simulation, the view of the world to an airborne viewer would be radically different than to an underwater viewer.

The intrinsic hierarchy of BIHs and PVW lend themselves naturally to this form of geometric simplification and is one of the motivational factors for the choice of a hierarchical metaverse architecture.

## 5.3 Client Connections

In Figure 2, Client 1 represents the canonical client connection – in this example, the client interacts only with a single simulator. Client 2 and Client 3 represent observers – users who are interested only in viewing (but not interacting with) this PVW instance in the broadest sense.

For a client to determine which simulators to connect to, given a location in space and desired level of detail, it will be necessary to query the PVW tree, searching for areas of interest. The query must begin at the root of the PVW tree, and traverse the graph until the desired sims has been located. For efficiency, sims may track nearby child simulators and create skip lists, so that clients can skip querying the immediate child sims of a parent before reaching their desired nodes.

Since the act of locating a desired sim is a non state-modifying operation, these types of requests can be directed to shadow sims (discussed in Section 6.1), to avoid burdening the main sim with servicing these requests.

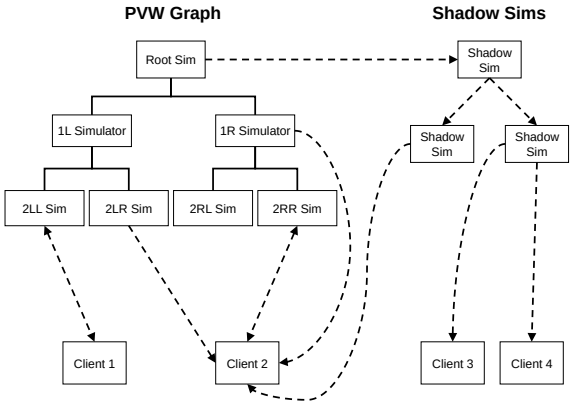


Figure 3: Network connections in a PVW instance, with shadow root sims. Dotted lines represent network connections, arrows represent information flow.

In the PVW architecture, clients may desire to extend their viewing range past their local simulator. By querying different simulators, a client can effectively extend their viewing distance to see more distant regions. Also, by querying up the PVW tree, clients can choose to only receive the level of visual detail (as discussed in Section 5.2) they desire.

In the PVW design, a client may have many read-only connections streams, but only requires one interactive connection (e.g. Client 4 in Figure 2).

## 5.4 Terrain and Global Objects

Certain types of objects that can potentially span an entire metaverse can pose problems to the PVW architecture if not specifically adapted to the system. A large entity, such as terrain or clouds & sky could easily cover the entire virtual space represented by a PVW tree.

These types of objects need to be specially constructed to allow PVW to split and share the load among many simulators.

# 6. PVW Extensions

The described PVW structure allows the expression of virtual spaces in a distributed load-splitting environment. For performance, stability and administrative reasons, this structure can be extended to accommodate differing demands of real-world implementations.

## 6.1 Shadow Sims and Reduction Engines

In many server-based architectures designed for streaming large amounts of dynamic 3D data (including PVW) the workload of managing the data-stream can be significant. This task involves non-trivial operations such as visibility calculation, data prioritization, progressive streaming and keeping track of client state.

To compound this problem, we anticipate there will be sims with a disproportionate amount of non-interactive streaming read load. Examples such as performers in a stadium simulation or being the root node of the PVW tree will attract a disproportionate number of client viewers who will not modify the state of the sim, but are interested in viewing the simulation.

To address this predicament, we introduce the notion of ashadow sim – a read-only copy of a sim. A sim with a disproportionate number of clients can elect to create a shadow copy of itself, and direct clients to stream data from the shadow sim. Streaming reads will be directed from the shadow copies, while interaction that modifies the state of the simulation will be communicated directly to the main sim.

If streaming load on a sim is extremely large, multiple shadow sims can be created, potentially structured into a multicast tree (Figure 3). In larger multicast trees, the root shadow node could act as a load balancer – instead of servicing requests directly, it could choose to redirect requests to other the children shadow sims.

Shadow sims may also be able to act as reduction engines, approximating and simplifying the data to suit the needs of individual clients. For example, some shadow sims may only need to service a sub-volume governed by its parent sim, or may only service reads of the simplified representation of the world discussed in Section 5.2.

## 6.2 Robustness

There are two types of robustness that PVW must be specifically address: the unexpected failure of a node and resilience against DDOS style attacks.

To address failure of individual nodes, we require parent sims to keep track of the state of its children, and for aggregate child sims to contain the state of their parent.

In the event that a sim might crash, the parent can spawn a replacement sim, using sim split mechanics (Section 5.1). If the parent is busy or unavailable, it should also be possible to reconstruct its state from its child sims. If the lost sim had an associated shadow sim, it would be possible for the shadow to simply assume the responsibility of the original sim.

Shadow sims can also act as a convenient defence against DDOS style attacks. In the PVW architecture it is critically important that the root of the PVW tree not be exposed to attack. In this architecture, interior nodes of the PVW tree need only service state-modifying transactions from other PVW servers, but not clients – clients need only be directly connected to their local leaf simulators.

Since all external client requests to interior nodes of the PVW tree will be read-only in nature, these can be serviced by shadow sims (Figure 3). In this manner, it will not be necessary for clients to interact with, or even learn the address of the PVW root. This allows us to easily construct firewalls to protect the structure of the PVW tree.

## 6.3 Extending Beyond 3D Space

For some types of virtual environments, it may be necessary to extend our definition of space from a regular 3-dimensional representation to higher-dimensional spaces (for some types of physics simulations) or non-Cartesian spaces.

For higher-dimensional Cartesian spaces, BIHs can be naturally extended by using axis-aligned splitting hyperplanes in place of 3D BIH's splitting planes.

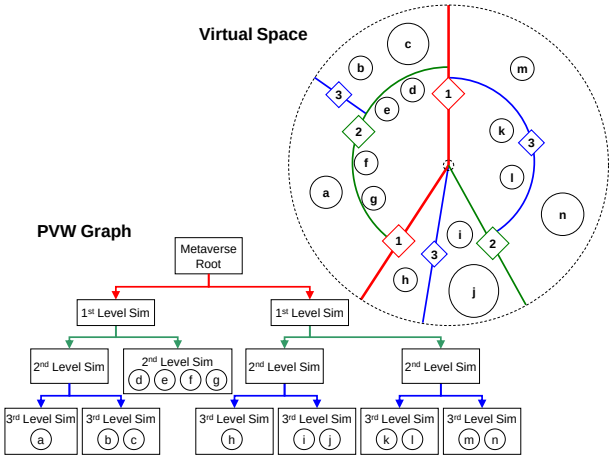


Figure 4: PVW partitioning on a non-rectangular space. A 3-dimensional extension of this 2 dimensional topology may be appropriate for some planet-shaped simulations.

In non-Cartesian spaces, it may still be possible to adapt BIHs in a meaningful way. Figure 4 shows a BIH partitioning scheme for polar coordinates, splitting on polar angles and distance from the pole. This concept can be extended to spherical/geographic coordinates, splitting on distance from the origin, and latitude and longitude.

Spherical coordinates may be appropriate for planet simulations for encoding the relative meaning of up and down in a simulated gravity environment.

## 6.4 Embedding and Linking Metaverses

PVW metaverses in different administrative domains should be allowed to interact without requiring the strong trust relationships that PVW implies.

The simplest solution to joining metaverses is to create portals - links between PVW virtual spaces. In the PVW tree, these portals can be represented by objects pairs, having one at each endpoint of the portal link.

This would facilitate linking both spaces within the same virtual world and between worlds in potentially different administrative domains.

In all MMO universes, there is frequent demand to create disjoint and potentially temporary spaces. In social networks, these can take the form of private chat rooms or sandboxes. In massively multiplayer online role-playing games (MMORPGs), we frequently find instance dungeons, where copies of game levels are instantiated so that different groups can play simultaneously without interfering with each other.

If we allow a PVW universe to be represented by disjoint PVW trees, it would be possible to support these types of private rooms/instance dungeons.

Another mechanism that we can use to join virtual worlds is to allow one PVW instance to be embedded in another. An object in one PVW space could represent an entire PVW tree, potentially run by an entirely different administrative domain with different spatial topologies.

It is easy to imagine the desire to have a space simulated by one PVW instance (e.g. Planet Intel), embedded in another (e.g. the ICANN galaxy), both administered by separate domains. In this manner, users would have a cohesive way to explore different worlds.

If we allow a transformation between embedded worlds, such as relative avatar/world scaling, it would be possible to support Alice-In-Wonderland-style adventures – visiting miniature worlds inside worlds. This mechanism could even allow a PVW universe to be embedded within itself in a recursive fashion.

## 6.5 Multiple and Extended Partitioning

One characteristic of PVW trees is that they only allow a single global partitioning topology. While this is a convenient mechanism to divide the processing and streaming workload, it will not provide an optimal partitioning for all aspects of the simulation, such as physics or sound processing which favour interactions that do not cross boundaries.

These secondary processing operations can be implemented sing a second set of servers, ideally using hardware optimized for the task. (E.g. Physics servers can be enhanced with physics coprocessors or faster CPUs.)

These secondary processing engines could interact with the main PVW tree as privileged clients, updating the state of objects managed by the primary simulators.

For some tasks, such as physics processing, it may be useful to construct a secondary PVW tree to divide the workload of processing this aspect of metaverse.

## 6.6 Proxy Servers Mitigation

In a real-world implementation of a PVW world, network latency between client and server may become an issue, especially during interactive sessions.

Introducing a proxy server inside the datacenter where the PVW servers are located can have multiple advantages. First, connection setup and teardown costs can be drastically reduced, since client will only need to establish a session between itself and the PVW proxy. Secondly, this may simplify the construction of the client. The proxy can be constructed in a way that obscures the complexity of the PVW world topology – the client will only have to communicate with a single server. This is similar to the approach used in ALVIC-NG [quax]. Thirdly, this can be used as a security measure. If all communication between a client and a sim must use trusted proxy server, the PVW core itself does not need to be globally routable. This defensive measure is similar, but more powerful than the approach discussed in Section 6.2.

If a PVW proxy server is introduced on a high-speed network link near the client, it would help minimize client latency. The proxies could be constructed with an internal reduction engine (similar to that discussed in Section 6.1). Since priority streaming algorithms are sensitive to network latency, this would allow more efficient and reactive data streaming.

### 6.7 Upper Level Object Allocation

One problem with dynamic, reactive, spatial partitioning is that it will lead to a high number of small, concentrated simulators to handle highly concentrated object loads (crowds). High-speed moving objects in the world, such as a rocket in a battlefield simulation, will need to quickly traverse many simulators. This can be a problem because migration of objects between simulators must be marshalled through the network, which is much slower than intra-simulator travel.

One approach to address this shortcoming is to allow high-speed objects to be handled by parent simulators. In this manner, high-speed objects can still travel quickly through virtual space but avoid crossing simulator boundaries.

Since parent nodes in a PVW tree already contain a representation of the contents of child sims, they can anticipate potential object interactions, and communicate necessary information to its child sims.

## 7. FUTURE WORK

The most significant area in need of development in PVW is designing an efficient partitioning strategy.

One operation that we should seek to minimize is region splits and merges. During these operations, the entire region state must be distributed to new simulators over the network.

The most obvious spatial partitioning algorithm is to simply choose a partition that evenly divides objects in the simulation. While this approach will guarantee an even workload among world simulators, it has the unfortunate characteristic that it often chooses partitions that divides groups of interacting objects. The goal of a good partitioning algorithm should be to partition the world in such a way as to allocate interacting groups of objects on a single server. If this guideline is ignored, nearby objects in different simulators will incur an additional network cost when interacting, which creates more work for the system.

Additionally, the portioning strategy should seek to minimize object simulator crossings. Any object that travels between regions hosted on different servers will need to be synchronized and marshalled across the network, which has a high cost compared to travelling to a new location in the same simulator.

This research is ongoing.

Another component of PVW that has not been explored in detail is the communication protocol necessary to support the types of interaction we anticipate. It will be necessary to develop synchronization and transaction primitives, as well as combining all the components of metaverse design (such as user authentication and asset management) which are not part of PVW.

## 8. CONCLUSION

The growing domain of metaverse applications use a variety of "scale-out" mechanisms to make ever larger virtual worlds. While these approaches provide a means to support increasingly large numbers of simultaneous users, they do not accommodate the demand for additional richer, simultaneous interactions. To drive new usages, what we want is to remove the limitations of current approaches so that the simulation architecture is driven by the content, rather than having the content limited by the architecture.

In this paper we described PVW, a hierarchical space partitioning architecture used to distribute a simulation workload in infinitely scaling chunks so that any simulation requirements can be met. PVW borrows acceleration structures from modern ray tracing algorithms to maintain a tree that successively divides the virtual space into manageable collections of objects and avatars. The unique benefit of the PVW hierarchy is that the simulation scales to accommodate both the limitations of the simulation and the requirements of the application. That is, the PVW architecture enables metaverse interactions to scale arbitrarily to accommodate the requirements of simulation by distributing the simulation across all available compute and communication resources.

## 9. Acknowledgements

We would like to thank the Jim Snow, Ed Kaiser, and Rob Knauerhase for their suggestions and advice.

## 10. References

[croquet] Croquet Project http://www.opencroquet.org/

[darkstar] Sun, Game Server Ttechnology White Paper, http://www.sun.com/solutions/documents/white-papers/me_sungameserver.pdf

[kumar] Sanjeev Kuma r, Jatin Chhugani, Changkyu Kim, Daehyun Kim, Anthony Nguyen, Christian Biania, Youngmin Kim, Pradeem Dubey. Chracterization and Analysis of Second Life Virtual World. *IEEE Computer Graphics & Applications*, (March 2008)

[os] OpenSimulator, http://opensimulator.org

[presetya] Kusno Prasetya and Zheng da Wu. Performance Analysis of Game World Partitioning Methods for Multiplayer Mobile Gaming. In *Proceedings of the Workshop on Network and Systems Support for Games (NetGames)*, (October 2008).

[reshetov] Alexander Reshetov, Alexei Suoupikov and Jim Hurley. *ACM Transactions on Graphics* 24,3, pp. 1176-1185 (2005)

[rubin] Steven M. Rubin and Turner Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. In *Computer Graphics (Proceedings of SIGGRAPH 80)* vol. 14, pp.110-116 (1980)

[sl] Second Life, http://secondlife.com

[uo] Ultima Online, http://www.uo.com

[wachter] Carsten Wächter and Alexander Keller. Instant ray tracing: The bounding interval hierarchy. In *Proceedings of the Eurographics Symposium on Rendering*, pages 139--149, 2006.

[wow] World of Warcraft, http://www.worldofwarcraft.com

[quax] Peter Quax, Jeroen Dierckx, Bart Cornelissen, Gert Vansichem and Wim Lamotte. Dynamic server allocation in a real-life deployable communications architecture for networked games. In *Proceedings of the Workshop on Network and Systems Support for Games (NetGames)*, (October 2008).