Workshops der
Wissenschaftlichen Konferenz
Kommunikation in Verteilten Systemen 2009
(WowKiVS 2009)

Load Characterization for Distributed Virtual Environments

H. Lally Singh and Denis Gračanin

12 pages

Load Characterization for Distributed Virtual Environments

H. Lally Singh[1] and Denis Gračanin[1]

Virginia Tech[1]
Blacksburg, VA USA

Abstract: Due to their sensitivity to user positions, actions, and visibility, Distributed Virtual Environments (DVEs) are well served through a focused load characterization effort. Quantified load can then be evaluated through models of the system behavior, resulting in expected behavior. A discussion of determining some characterized load from system requirements and instrumenting a DVE to provide validation metrics is also provided.

Keywords: DVE, Load Measurement, Performance Modeling

1   Introduction

DVE systems are often created with a specific range of user load in mind, often within the context of supporting specific user activities. In order to design, deploy, and maintain a DVE filling those requirements, the load must be transformed into more specific metrics that may then be used as performance objectives. Model-based transformation of that load may derive relationships between these and potentially other metrics to provide additional insight into system performance.
Load characterization efforts should support:

1. *Software Design* — During development, the performance objectives can assist in algorithm and data structure determination. Back of the envelope estimates can provide useful guidance for meeting the objectives.

2. *Engine Development* — Software Performance Engineering (SPE) techniques [SW02] can use the metrics and iteratively guide the development of a system that meets their criteria. Additionally, feedback from early prototypes provides useful insight on development progress.

3. *Asset Design* — Geometric models, terrain, and graphical assets can affect system performance significantly. Derived metrics covering these assets can provide design guidance to help ensure satisfaction of the objectives.

4. *Testing* — During development, validation, and continuous quality checks after deployment, the system will require testing to ensure that performance goals are met. Objectives may be derived from the originals to support measurement and testability.

5. *Operations Management* — After deployment, changes in user behavior can affect system performance. In response, changes to the deployed system configuration, or the system design may be necessary.

To support these activities we discuss methods to derive measurable metrics to characterize load, measure system performance, and identify components' response and behavior under load. All of the activities may be supported through the conversion of the DVE system from a "black box" into a set of componentized models. Each model represents a software component in the DVE. Each converts incoming load to resource requirements. Additionally, they are individually predictable from inputs, and are individually observable through known input, output, and intermediate variables. Finally, the outputs of the models may include resource requirements, such as memory, processor time, or network I/O.

Once the "black box" has been broken into models, load may be characterized within the context of one or more component models. Within this context, any incoming value to any component or pure resource can represent load. The definition of that value specifies the format for characterizing that part of the system load.

Within the SPE process, the performance characteristics of the system are maintained through design and development for both the software and assets. Initially, a model is constructed to estimate resource requirements, represented as a set of software components. A system design that satisfies the approximate scalability and performance requirements is selected. Once development has started, the model is updated to match the developed system, and then used for analysis of current performant progress. Evaluation of the model can provide insight into performance characteristics at each development iteration, allowing fast adaptation to discovered problems. Additionally, the model provides benchmark values for validation during the load testing of the system.

After deployment, the model provides a baseline for health monitoring of the system. Changes in user behavior or system configuration on the server, client, or network, can be interpreted in terms of their system ramifications, enabling adaptation to current or expected future changes.

This paper begins with a discussion of analytical tools for derivation, follows with a focus on spatial metrics common for DVEs, and finishes with a case study. The terms "level" and "terrain" are used interchangeably, to refer to the virtual space for users provided by a DVE.

2 Load Analysis

Load is characterized within the context of a model. Models of the software system often base themselves upon the software components, data structures, or algorithms used. This discussion will focus on component-centered models of the system.

The software system may be considered as an interconnected graph of software components, with workload applied from one end of the graph to the other. Figure 1 illustrates the concept. The bottom-most line indicates the flow of load from original inputs into the system, finally finishing as load applied to the operating system and hardware resource level.

At the far left of the diagram, top-level performance requirements are listed in original form. They are transformed into observable metrics once they are considered within the context of the software components that implement the functionality in question.

Before a modeling effort begins, some analysis may be necessary to reduce the scope of work. The minimal set may be determined from the top-level requirements; sufficient modeling must be done to determine the set of validation metrics required.
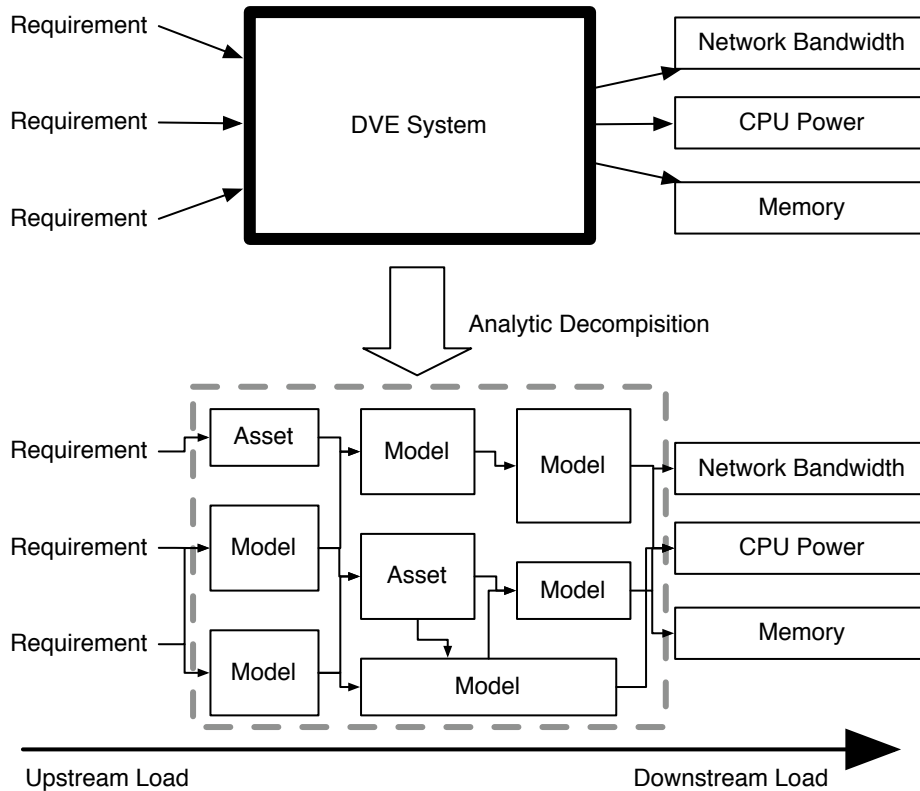
Figure 1: Refining the System Performance Model

Additional concerns for different phases of the system lifecycle: design evaluation, engine development (for Software Performance Engineering [SW02] activities), asset design, testing, or operations management will also drive the creation of additional models.

Models may also require further data, driving the creation of additional dependent models to provide the required parametric data. Alternatively, placeholder versions of the inputs could be created from simplified models of the values. For example, a constant, random, or linear interpolation of measured values could be used instead of creating the component model(s) that would otherwise be required.

2.1 Types of Load

Examples of *upstream* load within DVE systems include avatar movements, inter-user communications, and various interactions. Any characteristic of these upstream elements is usable as load for components. In the avatar movement case, the frequency of movement data coming in, divergence from prior directions or speeds, or number of other objects in the new position may be the relevant load characterizers for specific software components.

Downstream load includes the resulting load or resource requirements caused by software components, due to load upstream from them. Avatar movements may cause load upon the sim-

ulation or networking systems, which consequently provide load upon the processor, memory, and networking resources of the host computer system.

2.2 Validation of Load Response

The downstream load created by software components is measurable as a performance metric. During the various lifecycle stages of a DVE system, the measured performance of a software component may be used to make design, deployment, or operational decisions for that subsystem or the DVE as a whole.

Latency requirements serve as an example of load response. The time between an incoming update on an object state and the new resulting simulated state is measurable. Additionally, the times of the update and the state change are measurable at the component interconnect level, enabling analysis of latency contributions from individual software components.

2.3 Load States

Once it is understood which load values need characterization, there may be a requirement to characterize the entire system load state. Specifically, sets of load characteristic values are often related together as common values during a specific state of the system.

The simplest example is a steady-state run of a DVE with a specific number of users logged in. These are referred to as $SteadyState(N)$, with N being the number of logged-in users. If the number of users is expected to stay nearly constant, then this description may be sufficient for use.

Some systems may have peaks and valleys in load, requiring consideration for the outlier points. Similarly, the transitory states may also require consideration.

For example, some 24-hour continuous operations may have large shifts of load during the day as the user base logs in during the morning and out during the night. Additionally, certain spikes in activity may be expected during special events. In any of these cases, both the steady state and the transitions are relevant for study. Figure 2 is one such transition graph. Displaying a base steady load of N_1 users, it rises to N_2 during some part of the day. At some part in that day, a peak load of N_3 arises. It assumes that if $N_i < N_j$, then load to the system at N_i, $Load(N_i)$ is less than the load to the system at N_j, $Load(N_j)$. Specifically, that $Load(N)$ is a nondecreasing monotonic function over N.

Examples of spikes in load include the release of new content for the DVE, a planned special event, or the entry of a second group of users who stay for only part of the day.

Load state model requirements may be determined from expected variations in the load characteristics. System sensitivity to load variations is also a factor in determining if a load state model is necessary.

3 Spatial Load Metrics

DVE systems have many different elements, typically including some spatial sensitivities in performance. Area of Interest (AoI) mechanisms will load the network and computational load of the system, depending on aspects including inter-avatar visibility and distance.
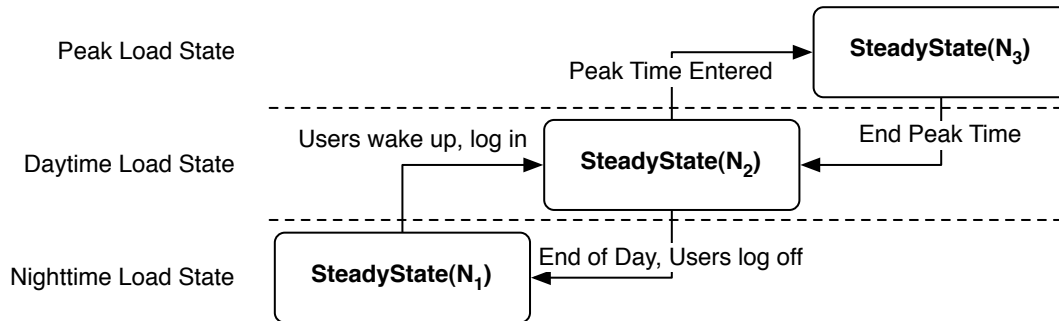
Figure 2: State Diagram of System Load

Physical simulation can depend on how many other objects are nearby. Collision detection is similarly sensitive to the number of objects in the vicinity.

For the spatial data described in this paper, a cellular structure is used. Additionally, a finite occupiable space is assumed. For infinitely occupiable spaces, the format will need reconsideration, based on the ways the DVE handles the space. For example, a hash function that remaps blocks of the occupiable space back into a finite set may also serve as the basis for storing spatial data.

In finite spaces, the space may be broken into a set of cells, which are each assigned data. The size of each cell should be determined from both the data collection mechanism and the minimal relevant granularity needed for analysis. For example, a cell size set to the bounding box size of an avatar can serve as a useful basis for both visibility and collision analysis, as in both cases, no more than one avatar can occupy the same cell at the same time.

In either of the data formats specified below, the analyst has the option of making multiple versions. The first analyzes inter-cell visibility within the level. The latter focuses on the frequency of object locations.

3.1 Visibility Analysis

A *visibility map* provides a representation of which other cells have a direct line-of-sight with the current cell. Depending on the type of objects in the system, the maximal length of that line-of-sight may be restricted to the largest AoI used in the system.

If different types of avatars have different visibility properties, then separate maps may be needed, particularly in the combinatoric case. However, they may be generated from per-type raw data.

Each cell is considered independent in a visibility map. Consequently, the number of dimensions used in the map is not considered for format of the map itself. Analogous to the way that multidimensional arrays are mapped to sequential cells in memory, each cell is assigned a single number, and they are treated as a one-dimensional sequence.

Several formats are available for different analytic needs. First is the raw avatar line-of-sight visibility map. Several of them may be required for different types of avatars. Finally, there are synthetic visibility maps for limited Area of Interest systems.

For a single avatar type, a half-matrix of boolean values can represent the full raw visibility map. Multiple types of avatars will require $T \cdot (T - 1)$ maps for T avatar types in order to capture all combinations of visibility. The boolean values can then be summed against a column or row to find the total number of other cells a given cell may see. AoI rules relating to object types and cell distances may be applied to alter these totals.

Alone, visibility maps provide a measure of how many simultaneous visibility relationships may need maintenance through the DVE synchronization mechanism. Peak and mean cell values provide feedback on the terrain design for the DVE, potentially providing indications of necessary changes. Combined with density maps, visibility data can provide specific expectations of how many relationships need to be maintained.

3.2   Density Analysis

The second spatial format discussed is the spatial distribution of objects. Called a *density map*, this data indicates which parts of a terrain have the least and greatest number of objects at a given time. Combined with the visibility map, it provides useful data for analysis.

When the system is in a stable load state, such as $SteadyState(N)$, the positions of all objects may be polled for an interval. Each position is mapped into a cell, and a counter in that cell is incremented. When complete, each cell value is divided by the sum of all cells. The result is a normalized density map.

For different load states, multiple density maps may be needed. It is likely that users will behave differently depending on different user counts within an area. For that reason, density maps for different values of N are recommended.

Differing types of objects should be collected into separate density maps. For example, different types of avatars can cause different movement behavior for multiple reasons: user identity, different avatar abilities, and social groups. Non-avatar objects, such as vehicles or projectiles, should be collected and maintained separately as well.

Alternatively, a set of line segments may better represent the paths of projectiles. Normalizing all the coordinates of segments into $(start, end)$ cell pairs reduces the storage cost into one similar for a visibility map, with a floating-point value replacing the boolean.

Raw data from multiple object density maps may be combined and the aggregate normalized for analysis. In situations where several types of objects need consideration as a single type (e.g. as a parent type in an ontology), such combination will be necessary.

For many-valued load states, the data collection requirements can become excessive. Instead, interpolation and extrapolation may be necessary. For example, density data may be collected at $N = 10$ and $N = 20$ and interpolated for $N = 11 \ldots 19$.

In either case, the analyst must look at the collected data to determine the viability of these transformations: the spatial behavior of users differs between different DVE systems and specific terrains. Specifically, users may choose to avoid areas when they become excessively populated, or avoid regions when nobody else is nearby.

In the interpolation case, the raw density maps for the closest two values of N are each weighted to their distance from the target value. Each cell value is a linear interpolation of the cell values from each map, weighted by their respective values of N.

In the simple extrapolation case, the largest normalized collected density map is assumed to

be valid for the larger value of N. Larger extrapolations will require more sophisticated methods.

4 Data Collection

Visibility and density maps will typically need collection as a separate process from software development. While the former may be done mechanically, the latter will require user studies.

Visibility data may be collected mechanically once the terrain data is complete. A ray is cast from the center of a cell to the center of another, for each pair of cells in the level. During level design, feedback from visibility analysis may drive design choices to minimize system load components driven by inter-user awareness.

Density map collection is more complex, requiring user studies and manipulation to properly determine expectations for specific user counts. Once the desired user count is known, a process for determining density data may be determined.

In a user study, the recording process will wait until the desired user count is available. Then, the process records the positions and types of every object at a specific interval, such as one second. When a representative sample is collected, a cellular grid (e.g. density map) representing the level is constructed, and each cell value is incremented for each object observed. Each cell may be represented with a single value, or as a sum of per-object type counters. The latter format enables per-type analysis.

5 Case Study: Network Synchronization

Presented is a simplified version of a First-Person Shooter networking system, based on the Torque engine. The requirements analysis and collection process are accurate, but the software component models have been simplified for brevity. The level used is a mountainous terrain with a small virtual village in the center. Only one type of avatar is available, using only a single type of projectile weapon. Avatars may walk, run, and jump: enabling a two-dimensional treatment of the level.

DVE systems such as MASSIVE-2 [Gre96, Gre99], MASSIVE-3 [GPS00], Quake [Fer, Dar], and the Torque [Gar, FG] system described here use some form client-server architecture, and transmit updates at 10 Hz. The server receives updates from all clients, executes the simulation, and sends updates back to all clients. Each update sends an orientation, position, and motion vector to the recipient for every object within its Area of Interest. The definition of Area of Interest for each node varies.

Back-of-the-envelope calculations quickly reveal a large growth rate for bandwidth requirements as the number of logged-in users grows, specifically $10Hz \cdot N^2$. depicting the notification of every client of every other client's state at every update. Combinations of dead reckoning [dis96] and AoI mechanisms reduce the frequency and cardinality of updates required, respectively.

5.1 Data Collection

First, density and visibility maps need construction. With only a single type of avatar, each is unidimensional. User studies collect 1 Hz samples of avatar density, keyed to the user count. Torque enables very large levels using tiled height maps specifying the terrain. However, a bounding box may be constructed using the maximal positions found within the sample data. The level is thusly converted to a $95x179$ cell two-dimensional grid with eight-meter square cells, and the density data mapped to it and normalized.
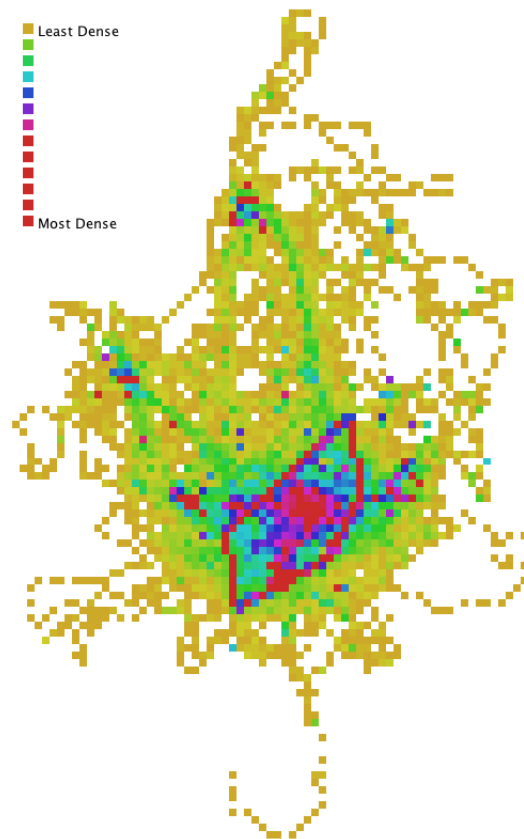

Figure 3: Density Data from a 10-user Study

Figure 3 shows the density map graphically. White represents no data points.

The visibility map was collected from an in-engine script running ray-cast tests between center-points in cells. The avatar's eye height was added to the terrain height and tested against a point similarly calculated for the destination cell. The final data was represented as a $17,005x17,005$ half-matrix of single-bit values. They were stored in a serialized Java Array, requiring less than 20 megabytes.

5.2 Requirements Analysis

Due to the variable density of objects over a DVE terrain, the network synchronization mechanism can have significant differences in performance for different terrains. First presented is a prediction mechanism for the number of visibility relationships to be maintained. Next, discussion on predicting the bandwidth requirements from the relationship count.

This case study will start with a system requirement that it must be able to support 10 simultaneous users, e.g. $N = 10$.

Using a singly-typed normalized density map, of a single type, or an aggregation of different object types, some functions are defined:

1. G — The set of cells in the normalized density map.

2. $P(i, N)$ — The value of a single cell i in the density map for N users. The value corresponds to the likelihood of a single object being within the cell. Note that $\sum_{i \in G} P(i, N) = 1$.

3. $V(i, j)$ — A visibility query function, taken from the visibility map. Its value is one if objects at cells i and j can see each other, and zero otherwise.

4. $A(i, j)$ — Indicates if cell j is within the AoI of cell i. One if so, zero otherwise.

With the parts defined, a predictor for the number of visibility relationships is presented as $Rel(N)$:

$$Rel(N) = N^2 \sum_{i \in G} \sum_{j \in G, j \neq i} [P(i, N) \cdot P(j, N) \cdot V(i, j) \cdot A(i, j)]$$

Given the number of logged-in users N, $Rel(N)$ provides the number of visibility relationships that must be maintained. The two invocations of P determine the joint probability of objects being in both cells. V and A provide logical tests to determine if the combination of cells is valid. Finally, the N^2 term converts the probability to an expected value.

Note that $i \neq j$. If server-side canonicalization requires that the client is sent a copy of its own state, an additional N term should be added to $Rel(N)$ before the N^2.

In combination with an estimate of the frequency of dead-reckoning changes, one may estimate how many updates require transmission per second.

Using a collected density map and generated visibility map from a user study of ten members in a Torque-based system, $Rel(10) = 15.4884$. The Torque engine prioritizes updates by distance, but does not define a discrete AoI set of criteria for the type of terrain used in the study. At the required update rate of 10 Hz, sending each side of each pairwise relationship an update, the expected number of of updates that the DVE would have to send is 309.768 per second. Using a custom Java-based analysis tool to run the equation, the computations took less than three seconds to run on a desktop computer.

5.3 Model Analysis

The full Torque implementation of the components described below are more heavily optimized, and thusly more complex, than those described here. Simplifications will be noted inline.

The first component is already a simplification: the contents of a single update. The model used here will be a thirty-two byte element, consisting of eight values: an identifier for the object that the update refers to, a "health" attribute for the avatar, the three-dimensional position (x, y, z), and the orientation (h, p, r). The first two elements are integers, the remainder are single-precision floating point values.

Serialization   First considered is the computation time required for the data transmission. Two components exist: the serialization time and packet transmission time. Assuming each update requires 24 clock ticks (read, write, and update-write-pointer), the current expectation of 309.768 updates per second, rounded to 310, requires $7,440$ clock ticks per second. Additionally, a bandwidth-relative packet-management cost is involved for destination buffer creation, user-space to kernel copying, and the transfer to the network. For this exercise an arbitrary value is chosen at 100 clock ticks per byte, requiring $31,000$ ticks per second. The total is $38,440$ ticks. While several billion are available in modern processors, note that $Rel(N)$ grows at $O(N^2)$. Systems with much higher user counts may encounter competition for processor time between the networking and simulation systems.

Transmission   Second is the network costs of transmission. Torque is similar to other systems in using the UDP [Pos80] protocol. As each set of updates transmitted to each client has a different destination address, they must be considered differently. From the understanding that $2 \cdot Rel(N)$ updates must be transmitted at 10 Hz, and that they will be distributed over $N = 10$ users, one may estimate the number of packets to transmit at $\frac{2}{N} Rel(N)$.

A new function, $Pack(b)$, can determine the resulting network costs of transmitting b bytes of data:

$$Pack(b) = \begin{cases} 1500 + Pack(b - 1460) & b \geq 1460 \\ b + 40 & otherwise \end{cases}$$

The number 1500 comes from the Maximum Transmission Unit (MTU) of Ethernet, assuming Ethernet is used here. 40 bytes are used for the combined IP [Pos81] and UDP protocol headers.

For this case study, the resulting transmission cost is:

$$N \cdot Pack(32 \cdot \frac{2}{N} Rel(N))$$

At $N = 10$, 100 bytes are sent to each client, with 40 bytes of overhead, at 10 Hz costing a total of 14 KB/sec.

5.4   Discussion

Through a user study, density data was collected on the only avatar type. Combined with a visibility analysis of the same level, an estimator $Rel(N)$ was created to determine the number of visibility relationships the DVE will have to maintain for a user count N. The user count is used

to determine the computational and network costs required to transmit updates at 10 Hz to all N users. An example of $N = 10$ is used.

The models estimate a total transmission rate of 14 KB/sec over the network, and 38,440 processor cycles per second. It is noted that both values depend on $Rel(N)$, which is $O(N^2)$.

6 Conclusions and Future Work

Presented is a methodology for determining load characteristics of a DVE system, focusing on spatial metrics for visibility and object density. Through analysis of required input load, models of system components can derive requirements for performance and capacity.

Using models of the software components within a DVE, it is possible to convert customer-level load requirements into measurable performance specifications for those components. Through analysis of the models, the performance specifications can be transformed into performance requirements down through the interconnection graph. The interconnections between the components provide additional measurement points within the system.

Through complete analysis of the components and requirements, system resource requirements may be derived.

Future work focuses on the relationships between the three elements of Figure 1: performance requirements, software components, and system resources. Specifically, model refinement for the software components within a DVE will enable further analysis. Combined with new requirements elicitation and analysis techniques for DVE system performance, we expect further capabilities to relate load requirements to the resource requirements of an analyzed DVE.

Additionally, DVEs could adapt to variations in load through changes in the service given. For example, a DVE could reduce the rate it sends updates for less important objects when the network is heavily loaded. The model could act as a basis for such an environment monitoring and adaptation system. Future research in this area may prove useful.

Finally, the cell size used in the spatial metrics is currently set to only the size of a single avatar. For small terrains, the spatial data are straightforward to compute and analyze. However, larger terrains, especially in three dimensions (such as underwater, aerial, or space-based DVEs), could prove expensive to analyze with that cell size. The relationship between model accuracy and cell size, and whether the cell size should stay constant in different parts of the level, are areas for future research.

Bibliography

[Dar]   Darren. Quake 3 Network Protocol. http://www.tilion.org.uk/Games/Quake_3/Network_Protocol.

[dis96]   IEEE standard for distributed interactive simulation - application protocols (IEEE 1278.1-1995). 26 Mar 1996.
          http://ieeexplore.ieee.org/iel1/3700/10849/00499701.pdf

[Fer]   T. Ferguson. Quake II Network Protocol Specs. http://www.csse.monash.edu.au/~timf/bottim/q2net/q2network-0.03.html.

[FG] M. Frohnmayer, T. Gift. The TRIBES Engine Networking Model. http://www.garagegames.com/articles/networking1/.

[Gar] GarageGames. Torque Game Engine. http://www.garagegames.com/products/browse/tge/.

[GPS00] C. Greenhalgh, J. Purbrick, D. Snowdon. Inside MASSIVE-3: flexible support for data consistency and world structuring. In *CVE '00: Proceedings of the third international conference on Collaborative virtual environments*. Pp. 119–127. ACM Press, New York, NY, USA, 2000.

[Gre96] C. Greenhalgh. Dynamic, embedded multicast groups in MASSIVE-2. Computer science technical report NOTTCS-TR-1996-8, School of Computer Science and IT, University of Nottingham, 1996.
http://www.cs.nott.ac.uk/TR/1996/1996-8.pdf

[Gre99] C. Greenhalgh. *Large Scale Collaborative Virtual Environments*. Springer-Verlag, 1999.

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (Standard), Aug. 1980.
http://www.ietf.org/rfc/rfc768.txt

[Pos81] J. Postel. Internet Protocol. RFC 791 (Standard), Sept. 1981. Updated by RFC 1349.
http://www.ietf.org/rfc/rfc791.txt

[SW02] C. U. Smith, L. G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison Wesley, 2002.