

THE “3D WIKI”: BLENDING VIRTUAL WORLDS AND WEB ARCHITECTURE FOR REMOTE COLLABORATION

Michael Roberts¹, Nicolas Ducheneaut¹, Trevor F. Smith²

¹Palo Alto Research Center and ²2038 Solutions, Inc.
mroberts@parc.com, nicolas@parc.com, trevorfsmith@gmail.com

ABSTRACT

While a lot of technical data is available on the Web, conveying information about detailed procedures for the assembly and repair of complex machinery has so far been limited mostly to 2D drawings and textual content. In this paper we describe the technology behind our 3D Wiki, a system meant to address this problem. By blending the functionality of virtual worlds (3D visualization and navigation) with the benefits of social software (editability, traceability, and hyperlinking), the 3D Wiki can help users tasked with the remote maintenance and repair of technical equipment. Our environment provides a kind of “living technical manual”, seamlessly blending spatial and textual content.

Keywords— Collaborative Virtual Environments, Social Software, Remote Collaboration

1. INTRODUCTION

Explaining a complex technical procedure over a distance is a long-standing problem. Paper-based technical manuals were the first attempt at solving this problem, but they suffer from their lack of interactivity and can easily become outdated. The advent of the World Wide Web helped address some of these issues with sites like Instructables (<http://www.instructables.com>) providing multimedia descriptions. Such sites most often offer digital pictures and movies, updating content through user submissions. Even so, complex procedures remain hard to convey online; the disassembly of an engine, for instance, can involve hundreds of steps resulting in an overwhelming amount of drawings and textual descriptions, or a very long movie that users are forced to experience linearly.

By contrast, users fortunate enough to have an expert on site can observe the problem and its solution directly: the machinery can be looked at from any angle and any level of detail, and the procedure can be experienced non-linearly by interacting with the expert. There is clearly a value in being able to experience the spatial aspects of the procedure and navigating the steps involved in whatever order and depth each user might find appropriate, and indeed recent work

exploring the use of immersive virtual reality on physical task learning confirms this impression [19].

In this paper we describe a prototype system, which we call the “3D Wiki”, aimed at blending the benefits of direct, physical interactions around complex machines with those of the Web. To do so, we combine a virtual world platform designed specifically to support Web applications with an interface inspired from common social software like Wikis. Other authors exploring Wiki-like 3D approaches to virtual worlds have recently applied them to city modeling [25].

We begin below with a high-level description of the user interface and actions available to users. We then focus the majority of this paper on the technical challenges we encountered in designing this system, and how we came to architect it to address them. We conclude with a brief performance evaluation demonstrating that the trade-offs we had to make still allow our system to support the kinds of load our application generates.

2. 3D WIKI: USER FEATURES

As a starting point, the 3D Wiki lets users create any number of “spaces” to store content and procedures related to a given machine. Users can upload content from a wide range of CAD programs, and this content is rendered in three dimensions in the rightmost part of the application’s screen (the viewer, see Fig.1). Users can navigate through the space using an avatar, much like they already do in commonly used virtual worlds or collaborative virtual environments, including early the early work exemplified in [20] or, more recently, Second Life. Thus, it is possible to use the system with multiple users simultaneously present in the same space, even though this is not a requirement and each space can be experienced individually and asynchronously. CAD models are parsed such that each individual component of a machine (e.g. a screw, a plate) can be selected and manipulated individually, using simple mouse clicks and drag-and-drop.

The leftmost part of the screen is dedicated to the Wiki functionality. Each component of a CAD model has a HTML document associated with it, which is automatically retrieved and displayed whenever the component is clicked on (clicking empty space retrieves the equivalent of a “home page” for the entire machine). Using the left pane, users can

edit these document pages much as they would with conventional Wiki editors on the Web including leaving comments, linking to external content, etc. For instance, a component's web page can be used to list component materials or procurement details (including a link to a provider site), or for a discussion between users on needed improvements.

Furthermore, the Wiki content and the components of a 3D model can be linked such that actions in the Wiki trigger events in the 3D view (and vice-versa). Using the 3D viewer, users can for instance select multiple components and define groups of inter-related parts. These groups can then be moved to various configurations (e.g. opened, closed, disassembled, etc.), which we call "states." Both groups and states can be linked to text in the Wiki pane. This allows users describing a technical procedure to easily highlight parts of a machine by linking to a group, or to interactively demonstrate a procedure by linking to a series of sequential states, which the platform is capable of animating using intelligent interpolation.

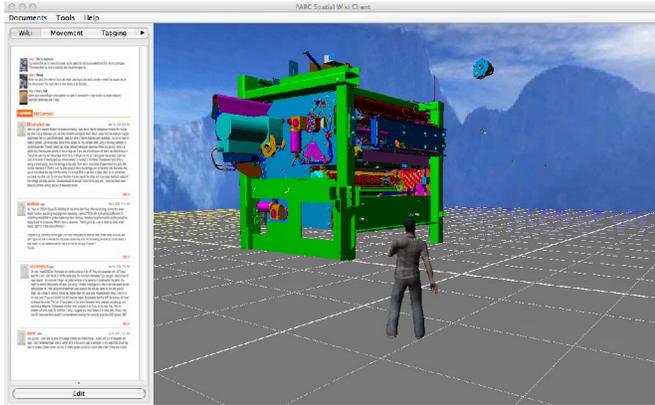


Fig. 1. The 3D Wiki user interface

3. 3D WIKI: TECHNICAL ASPECTS

The architecture of collaborative virtual environments, distributed virtual world systems, and multiplayer online games is typically thought of as a complex subject, involving the integration of components running across diverse systems, as exemplified in [4, 7, 21, 22, 24] and many others. Large commercial systems, such as massively multiplayer online games (MMOGs), operate at extreme levels of scalability, supporting thousands of simultaneously connected users and making optimizations in data models, networking architecture, messaging size, and frequency, plus other areas, as described in [23]. For the purpose of supporting our environment, however, intricate platforms are overkill. We therefore wanted to develop a platform similar in spirit to more complex commercial systems such as [9, 10] and previous work on web based virtual world systems [26, 27, 30], but "stripped down" to the bare needed

essentials and modernized to reflect current software trends. Others have previously created minimal VR environments in the same spirit [31].

Having previously used and developed a number of virtual worlds systems, we were very much aware of the limitations imposed on us by third party architectures and desired a modular system, comprised of small components that we could functionally modify using limited resources. Likewise, we wanted to use as many general-purpose off-the-shelf components as possible, provided these did not restrict the aforementioned goals. Most importantly, since we envisioned our system as more of a Web application than a full virtual world, we wanted our system to be based on Internet standards and protocols. This eventually led us to use a well-known platform for our backend: Apache Tomcat [14]. We describe in more detail below how such a choice easily supported many of the technical features we were looking for, as well as some of the limitations it introduced.

4. ADVANTAGES OF A WEB-BASED ARCHITECTURE

The use of a Web-based architecture allows us to functionally separate state-changing operations (for example, a change in the coordinates of an object) from more static data requests (e.g. downloading the base geometry of the same object) and aids integration with other Web applications and devices, a benefit which would not be as easily available were a standard networking engine like ICE [29] utilized. Indeed, static data requests are handled via a simple REST [2] interface connecting to an extensible array of Java Servlets communicating with the system's database. For instance, user account data can be obtained through a simple GET request to a well-formed URL. In this way, most of the basic information about a virtual space can be queried by clients issuing asynchronous HTTP queries, which greatly simplify the development of applications where a 3D space must be rendered, but not necessarily kept continually up to date with other views of the space.

For synchronous updates (such as joint manipulation of a machine's parts when two or more users are simultaneously present in a space), our system relies on Tomcat's "Advanced IO" [13] functionality (enabled via APR, the Apache Portable Runtime) to provide Comet-based messaging support in our server. Comet is a well-known system for achieving persistent TCP connections between a client and a Web-based server, and allows the server to asynchronously push data to the client. In doing so, we largely ignore latency issues associated with TCP and the Web-based implementation in favor of simplicity, though it should be noted that the APR system improves network performance enormously over the blocking Java sockets used in the default Tomcat HTTP networking layer. Newer versions of Tomcat can use Java NIO sockets to achieve a similar effect.

5. SPACES AND MODEL VIEW CONTROLLER

A single server can support a number of “Spaces” simultaneously. For simplicity, each space’s core implementation is currently a single threaded entity containing a variety of users and objects, which we refer to as “Things”. Such spaces are analogous to the model in the conventional MVC paradigm and are not spatially or functionally divided to provide enhanced performance. Instead, we exploit parallelism at the space level (each space operates in a separate thread) and also at the message processing level, with the networking functionality and rendering (for the client) being conducted in separate threads. Things can be organized into “Groups”, with sub-grouping allowed via the conventional scene graph paradigm. Similarly, Things can reference “Template” objects, which provide for object instantiation (many Things can reference the same Template). All objects are identified using 64-bit unsigned integers, assigned by the server at content creation time. Document objects provide access to textural documents, and can be linked to and from associated Things and Groups.

Spaces have an associated message queue, into which messages, generated from XML passed across the networking connections from clients, are deposited in the order received. Both the client and the server work broadly in the same manner; if, in processing the message, a state change is advised, the appropriate state variables are modified in the appropriate object model. Current views are implemented as scene graphs plus control logic and are capable of propagating changes to the model into a set of scene nodes, which provide for visual appearance. For simplicity, no message interest management, for example in the style of systems like Velvet [32], is applied.

Some messages describe operations that repeatedly change the state over time (for example, avatar movement messages). In the processing logic for such messages, the view takes responsible for implementing an appropriate dead-reckoning set of modifications to the view objects in order to secure a convincing illusion of continuous movement. For example, an avatar model knows only that it is moving to a specified location, and that it will reach the location at a particular time, with the view responsible for the interpolation of the intermediate positions.

6. MESSAGING SYSTEM

One design aim was to quickly support a variety of clients using the same underlying messaging and server system. Such clients are not necessarily full virtual world clients, but might include simple mobile clients interacting with the underlying server in a non-3D mode. We have currently implemented a Java-based client (described below) as well as a C++ client (on top of DirectX10) and a Flash implementation (currently limited to the networking layer).

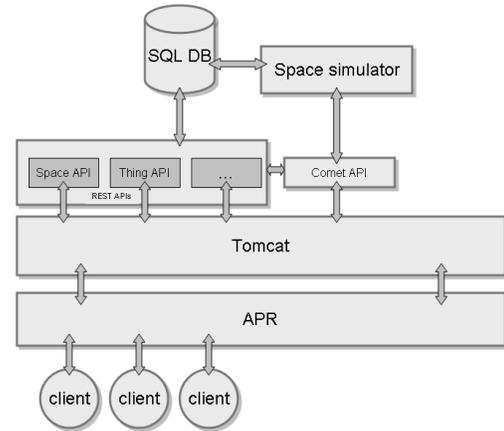


Fig. 2. Server architecture

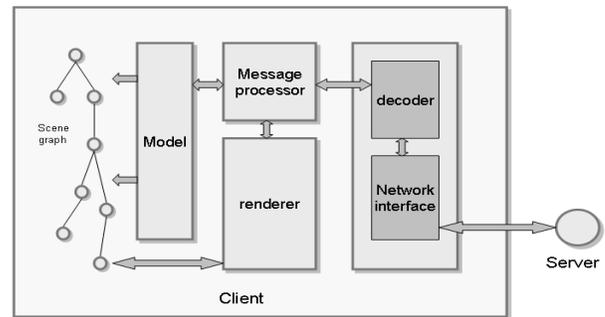


Fig.3. Client architecture

All of these clients can participate in a virtual world conversation together, connected through the same server.

To support this approach and eliminate the complexities inherent in cross-platform data type support, we chose to encode system messages using XML, with the aim that the implementation of a messaging layer on a new client architecture requires nothing more than an XML parser, base-level TCP and HTTP access, plus the development of the wrapper classes used to surface data from the XML messages to the client application. Figure 2 summarizes high-level components of the server architecture.

7. CLIENT AND VIEWS

The client differs from the server in that it has a view that exposes the state of the replicated objects to the user. To avoid unnecessary synchronization, patching of state from the model to the view’s scene graph nodes is interleaved with scene drawing operations in the main render thread. Thus, while message decoding and processing happens in an external thread, the render/view thread is solely responsible for the application of state to the scene graph. This simplification reduces the complexity of the state transfer

process, at a slight expense of small cost scaling with the number of incoming messages to the render thread. For rendering on the Java platform, we use JavaMonkeyEngine (JME) [5]. The client architecture is summarized in Figure 3.

8. ASSETS AND CACHING

Large objects, such as geometric models, avatar files, textures, or larger pages and documents are stored on disk in an asset store. References to disk objects and metadata are stored in a variety of entities, which are persisted to a backing indexing database. For example, the model's "Thing" object contains placement information, including position, orientation and scale, as well as a reference to a Template object. Templates store object identity/filename information as well as time/date stamp information, and thus reference on-disk objects in the asset store. Using this data, a client can quickly determine whether an object in a local media cache is up-to-date and if necessary download a fresh copy from a media retrieval API, described below, to which the identity information is provided.

Assets, along with other pieces of content, are added to the system via the aforementioned REST API. Separate APIs available for spaces, things, avatars, users, documents, wiki information and related objects. Subsequent calls to the media retrieval REST API can be used to retrieve an asset from the server.

For asset import, we have implemented custom Python export code to support the Collada [6] format from Maya 2008, as well as X3D [18], VRML [17] and Wavefront OBJ formats. A conversion pipeline provides us with access to industrial and architectural CAD data via conversion to Collada. Internally, all objects having a visual representation, apart from avatars, are stored in the OBJ format, with a separate file for either material definition or textures. Despite the limitations of the OBJ format, we felt its openness and relative ease of parsing would simplify implementation of clients across languages and indeed, most rendering engines we have used include OBJ parsers.

Avatars are maintained as skinned boned meshes in Collada format. We currently use avatar animation provided via captured motion data, similar to the approach used in [28]. Blending is used for transitions between animations. Currently, each avatar is separately instanced, with no shared data between avatars, even when the avatars share the same model. This is an area for improvement, as we will see in the performance evaluation section. Figure 1 illustrates a complex CAD model and avatar in the environment.

9. PERSISTENCE AND DATABASE CONNECTIVITY

For persistent world storage, we developed a database interface specific to our entities using Hibernate [3]. Using

this interface, world model entities can be persisted into any generic SQL database, which holds indexing information and other meta-data, which references assets in the asset store. The database interface allows us to store individual entities, and also to load all of the entities for a particular space in bulk when a space is instanced onto a server. During such a load, all entities relating to the space are extracted from the database and model objects constructed from them. For connecting clients, a snapshot of the spaces entity state is downloaded at client-login, and kept up-to-date with universally applied state change messages.

10. OPTIMISTIC STATE CHANGES

In certain circumstances, for example, when the user is in direct control of their avatar, it does not make sense to send a message to the server to affect a state change and then back propagate it to all connected clients. Doing so would introduce unnecessary lag for the user. Instead, such state changes are executed locally, and a message sent simultaneously to the server to indicate the change. Clearly, this does not make sense for objects that are potentially contested; all messages for such objects must be first directed through the server system in order to be properly ordered.

For state changes that generate frequent messages, for example avatar movements, we also implemented systems which interact with the users input movements and perform appropriate message optimizations, such as sending messages at fixed intervals, while an avatar is moving. Client side implementations interpolate positions, providing the illusion of smooth movement.

11. DOCUMENTS AND CONNECTIVITY

As mentioned earlier, large documents are stored in the asset store, which indexing and meta-data information available through the model objects. Smaller documents, including wiki information are stored directly in the indexing database. Client applications retrieving or modifying document information can do so via the REST API provided by the server. Bi-directional connections may be established between documents and other model objects, including thing, states, and other objects. This information forms a graph of connected documents and objects, which can be used to provide in context information for a given entity. Additionally, to support finding relevant documents from parts, a search interface is supported. This interface uses standard TFIDF indexing to dynamically issue searches against the document corpus, displaying search results in a separate window.

12. PERFORMANCE EVALUATION

Since the rendering system runs largely independently from the messaging system, the client frame rate is somewhat,

though not completely (as we will show) decoupled from the network performance. In our evaluation, we were primarily interested to know how our system would perform with respect to the number of users per space and also with respect to inter-object messaging. Both these metrics can provide useful insights as to loading parameters for the use of the Tomcat/APR/Comet interface combination.

To assist in our evaluation, we wrote a simple load-inducing simulator, capable of piloting a range of automated users. Simulators open a number of sessions (each simulating one user, with separate Comet/HTTP server connections and model objects) that execute a mixture of object movement operations and avatar movements along the ground plane. For testing purposes, we used a mix of 60% avatar movement messages, 20% object manipulation messages and 20% rest time; we believe this combination to be in excess of the normal load pattern for users in the system.

The messaging implementation was modified to include millisecond accurate time data on each message, indicating when the message was sent, when it was received, and time between message reception and processing (time in queue).

Testing was conducted using Intel Core i7 940 machines (Quad core, 2.93Ghz, 3Gb main memory) running Microsoft Windows XP sp3 on a 1Gb NVidia GeForce 285 based video card. We captured the client framerate, time taken to send a message from the client to the server (transit time/latency) and time taken to process a message (message processing time/latency). Avatars consisted of models with 42 hardware bones and 5138 triangles, textured with a 24 bit-per-pixel texture map at 2048x2048. All avatars used the same model file, but we did not use instanced geometry so as better to approximate a real-world test with diverse avatars. It is worth noting that our avatars are comparatively high polygon count and well textured, and that no Level of Detail (LOD) optimizations were applied.

13. RESULTS

13.1. Rendering performance

Figure 4 shows the framerate of the system under load with spaces of increasing complexity and density, ranging from 4,860 to 569,076 triangles. User load was progressively increased for each space and the average framerate observed. As can be observed, the framerate trends generally down with the number of users, caused mainly by the time taken to render a frame increasing with the scene complexity. During our initial tests, we were also surprised by a knee in the framerate curve at around 60 users in both the small and medium spaces. Since scene complexity varies greatly between these two spaces, we initially hypothesized that the performance degradation was due to a performance bottleneck at the server, with message processing time degrading severely at or beyond 60 users. Server-side data (Fig. 6), however, proved this was not the

case, with avatar rendering dominating the results, confirmed by increasing the number of objects to 3000, for a total of 569,076 triangles, and removing avatar rendering. The purple line on Fig. 4 shows the results of this test. As can be seen, there is a small effect on the framerate with the number of users, but the decay here is more linear. This decrease is likely attributable to the overhead of processing state change messages inside the render thread, suggesting that simpler avatars would provide further scaling.

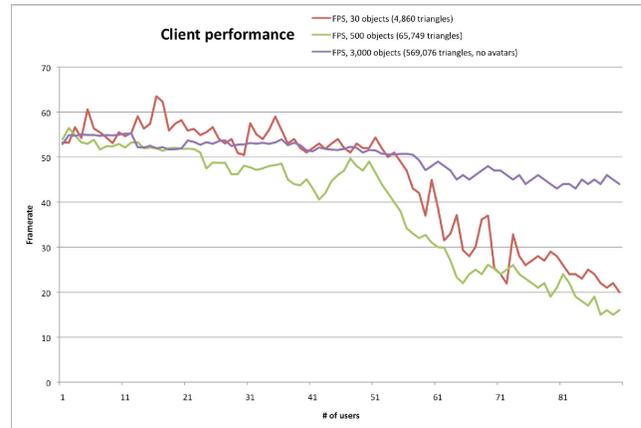


Fig. 4. Impact of user load on client frame rate

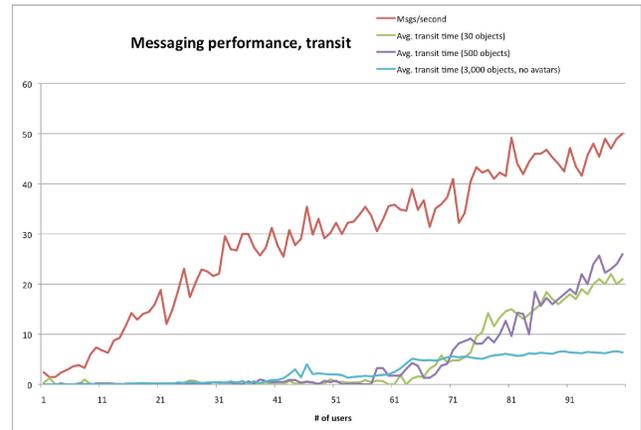


Fig. 5. Message transit time (in ms.) for the three spaces

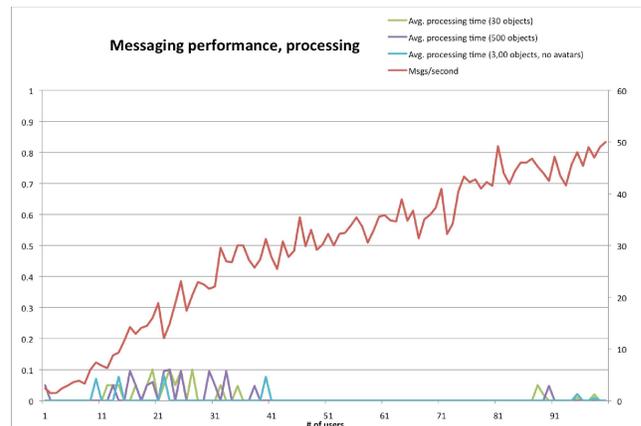


Fig. 6. Message processing time (in secs.) for the three spaces

13.2. Messaging performance

As mentioned earlier, we evaluated messaging performance by inserting timing information in the messages. Fig. 5 (captured concurrently with the data in Fig. 4) shows the number of messages flowing in the system increasing approximately linearly with the number of users. Transit time remains low and flat until approximately the same point at which the framerate had begun to dip at the aforementioned knee for both the small and medium spaces (with avatars present), but remains relatively low when avatars are absent (even for a more complex space). Therefore, we believe that the increase in transit time is not due to performance limitations at the server, but rather due to the client slowing down under the rendering load, with messages reaching the server more slowly. Should this be remedied, we expect server messaging performance to remain acceptable beyond the current test limit of 100 users. This is reinforced by the data in Fig. 6, showing that message processing time remains small even for higher loads.

14. CONCLUSIONS AND FUTURE WORK

This paper has described our 3D Wiki system, which integrates virtual world behaviors with Wiki-like functionality and document connectivity. Our current implementation indicates that such a blend can be achieved easily using simple off-the-shelf components (most notably, Tomcat) and Web standards (REST, XML). While the use of such components has important performance limitations, we showed that the messaging performance of the system is still reasonable on modern hardware, with the cost of rendering complex scenes introducing the most significant messaging delays. Optimizations at the client (esp. simplifying the avatar system) would most probably help the system scale beyond the current limit of about 60 users per space. The system currently forms the basis for our ongoing work on remote servicing, social awareness and application-specific virtual worlds; we expect to keep refining it as more applications are developed.

15. REFERENCES

- [1] Allard, J., Gouranton, V., Lecointre, L., Limet, S., Melin, E., Raffin, B., Robert, S. (2004): FlowVR: A middleware for Large Scale Virtual Reality Applications. *Lecture Notes in Computer Science*, 3149
- [2] Fielding, R.T. (2000): Architectural Styles and the Design of Network-based Software Architectures.
- [3] Hibernate.org: Relational Persistence for Java and .NET. <https://www.hibernate.org/>.
- [4] HLA: High Level Architecture. <http://hla.dmo.mil>.
- [5] Java Monkey Engine: JME. <http://www.jmonkeyengine.com/>.
- [6] Khronos Group: "Collada Specification". <http://www.khronos.org/collada>.
- [7] Macedonia, M., Zyda, M., Pratt, D., Barham, P., Zeswitz, S. (1990): NPSNET: A Network Software Architecture for Large Scale Virtual Environments. *Presence*, 3: (4)
- [8] Okino Inc.: NuGraf. <http://www.okino.com/nrs/nrs.htm>.
- [9] Project Wonderland: "Toolkit for Building 3D worlds". <https://lg3d-wonderland.dev.java.net/>.
- [10] Teleplace: <http://www.teleplace.com/>.
- [11] Krasner, G., Pope, S. (1998): A Cookbook for using the model-view-controller User Interface Paradigm, Smalltalk-80, Journal of Object-Oriented Programming, Volume 1, Issue 3
- [12] Rocketbox Studios: Complete Characters. <http://www.rocketbox-libraries.com/index.php?cat=cc>.
- [13] The Apache Software Foundation: "Advanced IO and Tomcat". <http://tomcat.apache.org/tomcat-6.0-doc/aio.html>.
- [14] The Apache Software Foundation: "Apache Tomcat". <http://tomcat.apache.org/>.
- [15] The Ogoglio Project: The Ogoglio project in 9 slides. <http://ogoglio.com/>.
- [16] Tran, F.D., Deslaugiers, M., Gerondolle, A., Hazard, L., Rivierre, N. An Open Middleware for Large-scale Networked Virtual Environments. *Proceedings of IEEE Virtual Reality 2002 (VR'02)*, IEEE Computer Society.
- [17] Web 3D consortium: VML97 and Related Specifications. <http://www.web3d.org/x3d/specifications/vrml/>.
- [18] Web 3D consortium: X3D International Specifications. <http://www.web3d.org/x3d/specifications/>.
- [19] Patel, K., Jung, S., Rosen, D., Bajcsy, R. Bailenson, J.N (2006): The effects of fully immersive virtual reality on the learning of physical tasks. Proceedings of the 9th Annual International Workshop on Presence
- [20] Capin, T.K, Pandzic, I.S., P., I.S., Magnenat, Thalmann, N., Thalmann, D. (1998): Realistic Avatars and Autonomous Virtual Humans, Proceedings VLNET Networked Virtual Environments
- [21] Pettifer, S., Cook, J., Marsh, J., West, A. (2000): DEVA3: Architecture for a large-scale distributed virtual reality system. Proceedings of the ACM Symposium on Virtual Reality software and technology.
- [22] Carlsson, C., Hagsand, O. (1993): DIVE: A platform for Multi-User Virtual Environments. *Computers and Graphics* Volume 17, Number 6
- [23] Smed, J., Kaukoranta, T., Hakonen, H (2002): Aspects of Networking in Multiplayer Computer Games. *The Electronic Library*, Volume 20, Number 2.
- [24] Greenhalgh, C. (1998) Awareness-based communication management in the MASSIVE system. *Distributed Systems Engineering*, Volume 5, Number 3.
- [25] Irschara, A., Bischof, H., Zach, C. (2007): Towards Wiki-based Dense City Modeling, Proceedings of the Eleventh IEEE International Conference on Computer Vision, Workshop on Virtual Representations and Modeling of Large-scale environments (VRML)
- [26] Wray, M. and Hawkes, R. (1998) : Distributed Virtual Environments and VRML — an Event-Based Architecture, *Computer Networks and ISDN Systems* 30
- [27] Reitmayr, G. and Carroll, S., Reitemeyer, A. Wagner, M. G. (1999): DeepMatrix – An open technology based virtual environment system, *The Visual Computer* 15, Number 7/8
- [28] Sannier G., Balcisoy S., Magnenat-Thalmann N. Thalmann, D. (1999): VHD: a system for directing real-time virtual actors, *The Visual Computer*, Volume 15, Number 7/8
- [29] ICE; The Internet Communications Engine, <http://www.zeroc.com/overview.html>
- [30] Barbieri, T. (2000): Networked Virtual Environments for the Web: The WebTalk-I & WebTalk-II Architectures, Proceedings IEEE for Computer Multimedia & Expo 2000 (ICME).
- [31] Fair'en, M., Brunet, P., and Techmann, T., (2004): Minivr: A portable virtual reality system, *Computers & Graphics*, Volume. 28, Number 2.
- [32] Very Large Virtual and DeOliveira, J.C. (2002): VELVET: An Adaptive Hybrid Architecture for Very Large Virtual Environments, *Presence*, Volume 12, Issue 6.