An Object Server System for 3D Digital Design Collaboration

Theodor G WYELD
School of Architecture, Landscape Architecture and Urban Design, University of Adelaide, Australia

Keywords: collaboration, multi-user, object sharing

Abstract: Moving through and between computer generated 3D objects we experience a 'virtual world'. Virtual Worlds have created a dream-like landscape. They also have facilitated a paradigm shift for architects working with CAD tools where they now desire to “design three-dimensional spaces in an immersed way” (Strehlke and Engeli 2001). Architects are already working in computer-moderated collaborative networked organisations. A 3D Virtual World offers a different kind of collaboration. To understand how architects might design in an 'immersed way', three experiments are described. The experiments attempt to investigate how participants *experience* the 3D spaces within which they collaborated. In particular, the last experiment makes use of 'shared objects' in the scene. The software chosen to create these Virtual Worlds was VRML and JAVA due to their flexibility and rapid prototyping. Where VRML differs from most CAD languages is in its openness. This paper describes an object sharing client-server architecture based on a simple multi-user system providing navigation in CosmoPlayer 2.11 ported through Netscape. The Object Server System allows multiple clients, as avatars, and objects to be manipulated in a 3D Virtual World. The system updates the transforms of the objects explicitly allowing their transform values to be shared across multiple browser sessions.

1 INTRODUCTION

The science of perspective, which arose in the 15th Century Renaissance, allows us to 'see' the world as a collection of points, lines, and surfaces. The computer frustum is a direct manifestation of the laws of perspective. Computer Aided Design (CAD) relies on the ability of computers to represent the physical world as a collection of points, lines, and surfaces. These abstract, mathematically derived, objects are configured on the screen such that they depict a reality which we agree reflects the physical world. We 'see' what they depict because of our 'perspectival knowing'. In other words, we have become accustomed to seeing objects in perspective and, hence, respond to the cues displayed on our computer screens as being 'real things'. Moving through and between these 'things' we experience a 'virtual world'. Virtual Worlds have created a dream-like landscape based on a mathematical certainty.

1.1 BACKGROUND

The origins of Virtual Worlds lie in text-based multi-user environments which first appeared in the 70's and 80's. In the 90's, with the advent of the World Wide Web (WWW), text-based chat channels with visual interfaces using 'Avatars' emerged. There followed a "movement to colonise Cyberspace and transform it into a galaxy of interconnected Inhabited" Virtual Worlds (Damer et al. 2000). And more recently there has been a paradigm shift for architects working with CAD tools where they now desire to "design three-dimensional spaces in an immersed way" (Strehlke and Engeli 2001).

Architects can choose from many kinds of interactive Virtual Worlds present on the Internet: Multi-user games (Quake et al.); those used for research purposes (Han and Turner 2001); educational Virtual Worlds (Clark and Maher 2001; Maher 1999); trade shows (Damer et al. 2000); and, home generated novice worlds (using VRML).

Increasingly common, in architectural practises today, are computer-moderated collaborative work ventures from networked organisations (Mitchell et al. 1998; Sudweeks et al. 1998). E-mail, chat, video conferencing and file exchange are still, however the most frequently used media for collaborative meetings online.

In a 3D Virtual World, different kinds of meetings can take place. Meetings such as redlining of documents (Do et al. 2001), moving objects to arrange meaningful assemblages, sharing video and audio streaming, and choreographed movement through space, virtual galleries; web casting 'walls' for streaming of video and audio data (Damer et al. 2000); and 'rooms' etc (Richens and Trinder 1999). What differentiates 3D virtual world space from its real-world counterpart is that one can use hyperlinked objects and text to teleport to other parts of the world or, indeed, to other worlds.

Many of today's Virtual Worlds and their communities, which once drew their roots from MUDs (Multi-user Dimensions) and text-based real time chat systems, now utilise the power of existing 3D rendering engines developed for gaming applications such as Doom and Quake for their success. Some of the features of these Virtual Worlds include impressive 3D effects, social interaction and hierarchies. Providing a compelling 3D interface encourages users from a wide non-technical background to participate and experiment in a common digital 'place'.

2 VIRTUAL DESIGN SPACE

Attempts to use Virtual Worlds as a design space include the work of Richens and Trinder (1999), Clark and Maher (2001), Maher (1999), and Wyeld (2001) among others. To understand how architects might design in an 'immersed way', three experiments were conducted with students of a Masters program in Digital Media, the first experiment used both a physical and virtual 'kit-of-parts', for constructing a 'shape grammar' for a typical 1900's Australian Villa, and to explore constraints of the different media. In the second experiment notions of the 'bounded fragment' as the essence of architectural space as espoused by Oxman et al. (1987) was explored using a networked, multi-user, Virtual World, revealing a 'shared experience'. And in the third, notions of privacy in a networked, Virtual World, were explored using 'shared' objects, exposing different layers of immersion. In each case, participants communicated with each other via a networked, text-based, chat facility.

2.1 Design Space 1

The first experiment used ‘shape grammar’ for a typical 1900's Australian Villa as a motive for exploring the correlation between its physical and virtual understanding (Radford 1994) (Figure 1).

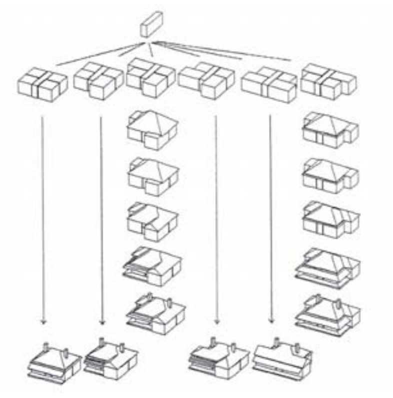

Figure 1 Shape grammar for a suburban house in Adelaide showing possible (though not exhaustive) permutations which generate an urban streetscape. (Radford, 1994, p-170).

The aim of this experiment was to encourage participants to reflect on how they interact with media, be it physical or virtual. Participants were required to determine how many possible and desirable permutations they could find given the virtual and physical blocks. Participants were located remotely and communicated with each other via a chat channel. Immersed in the task, participants spent most of their time trying out all possible permutations (see Figure 2).

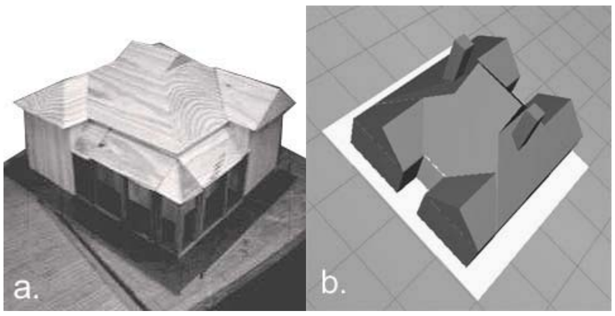

Figure 2 (a) Australian Villa using wooden blocks from a physical kit-of-parts and (b) a 'H' configuration using a virtual kit.

It was found that the Villa's virtual blocks opened possibilities that their physical counterpart couldn't. The infinite spatial relations created by merging virtual blocks, their ability to defy gravity, and gaining views not possible in the physical world, opened new avenues of imagination, fun and discovery. Participants agreed that the virtual kit showed more flexibility and assisted more rapid visualisation of design ideas than its physical counterpart.

2.2 Design Space 2

In the second experiment Oxman et al.'s (1987) exploration of the elements of plans in *The Language of Architectural Plans* lent itself as a case study for architectural exploration of multi-user worlds. The aim of this experiment was to reveal if multi-user environments foster participation and sharing in a common experience.

Participants explored the language of the plan of a virtual Barcelona Pavilion. The motive for encouraging participants to engage in the experiment was to test Oxman et al.'s (1987) 'bounded fragment' as the essence of architectural space. The task was to communicate effectively and arrive at some consensus about the validity of a bounded space in a Virtual World. Each participant had access to a networked virtual Barcelona Pavilion and common chat channel. As they entered, they were represented by an avatar.

From *inside* the virtual Barcelona Pavilion, participants attempted to identify the boundaries, real and implied from their *experience* of its space (Figure 3). A sub-game spontaneously emerged: hide-and-seek. Hiding in a virtual 'room'; the searching activity appeared to be the same as for a physical room (peeking around corners etc) (Figure 3.). This was indicative of the level of immersion.

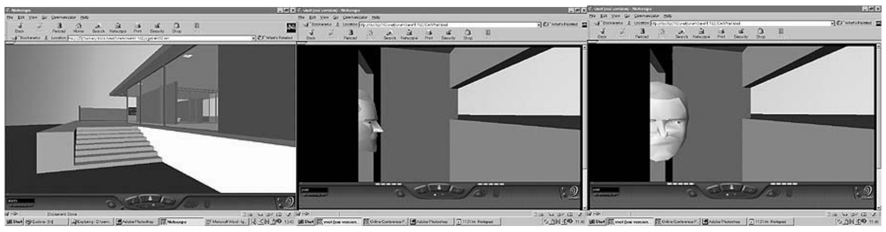

Figure 3 Entry to the Virtual Barcelona Pavilion, and the Hide and Seek Game: Found!

At times there appeared to be a complete overlap of physical-space protocol and virtual. Participants agreed that being able to cluster in a corner of the 3D Virtual World and know that everyone else is in exactly the same location and looking the same way you are, meant you knew they were looking at the same thing. It was postulated that this might be a better way of collaborative design discussion than having a co-collaborator looking over your shoulder at a monitor merely displaying a CAD model. Or worse, if your co-collaborator is remote with only a CAD model to look at, the level of synchronous communication is seriously diminished. Instead, the act of virtually moving from location to location 'within' the design was a clearly coordinated, 'shared' experience.

2.3 Design Space 3

In the third experiment investigating notions of privacy was used as a motive for participants to immerse themselves in a 3D Virtual World. Participants were located remotely and communicated with each other via a chat channel. They where represented by avatars in a networked multi-user scene which included eight different types of shared construction elements (see Figure 4). All participants could see the elements move in real time as actioned by any participant. Participants urged each other to occupy their viewing position so that the other could see what they saw. They communicated effectively and arrived at consensus about the validity of private space in a Virtual World. Using the virtual construction elements provided, participants constructed a pool house which included a private space for males and females to change into swimming costumes. By communicating textually and with the corresponding movement of virtual objects a design idea was both realised *and* experienced.

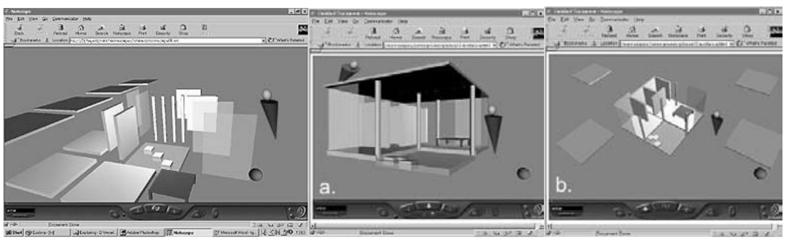

Figure 4 Virtual construction elements for a virtual Pool House, its view constructed at floor level (a), and the 'god-like' or above view (b). Note red button on lower right of frame disables the various sensors to facilitate smooth movement through the scene.

The experiments attempt to investigate how participants *experience* the 3D spaces within which they collaborated. It highlights how real time navigable Virtual Worlds lets one navigate a scene to see for themselves what is "over there", facilitating collaboration with others in real time, within a virtual design space.

The experiments follow a logical sequence, building on the progressive learned experiences of each. In particular, the last experiment makes use of 'shared objects' in the scene. It brings together the notion of the participant's relationship to the objects in the scene and their co-relationship with the objects as shared with another. It is in this final synthesis that we catch a glimpse of the practical application of this technique. Cooperation and participation resulted in a collaborative design outcome. It concluded the worthiness of real time design collaboration in networked multi-user environments.

It is the system architecture of this last experiment which is described in detail in this paper.

3 WHY VRML WAS CHOSEN

In all the experiments real time navigation of the virtual space allowed participants to go places which the rendered still images and controlled frame animations of CAD programs don't. And in the final experiment the Virtual World allowed participants to share the manipulation of objects in a common scene whilst also navigating the spaces created. In choosing software suitable for creating these networked Virtual Worlds, VRML and JAVA were chosen for their flexibility and rapid prototyping.

VRML was chosen because as a real time navigable virtual environment scripting language it supports distributed multi-user applications. While there are a number of languages which support this function they suffer from varying degrees of support and platform dependence. In any distributed environment one finds differing performance in differing platforms. A criteria, of which only VRML satisfied, was that the language needed to be platform independent and useable on machines with varying performance characteristics.

VRML runs above JAVA and as such is essentially platform independent. The base VRML specification supports access to JAVA scripting and classes via the External Authoring Interface specification (EAI). As such it supports the creation of sockets, hence, multi-user capabilities. Much work has been advanced by the Sony Research Labs in creating stable multi-user interactive environments (Lea et al. 1997). VRML is an ASCII text based language and files created for viewing in a VRML browser can be opened by any text-based editor. There is no proprietary license over the language and most VRML browsers are Freeware (an important feature for researchers and students alike). Because the language is open and able to be edited in any text editor, this means one can gain an understanding of the code's structure very quickly. Based on JAVA, VRML is intuitive and easily comprehensible using common terms and recognisable procedures/declarations. Where VRML differs from most CAD languages is in its openness.

As real time navigation of 3D worlds continues to emerge as the basis of a new way to visualise architectural space, real time renderers such as viewers of VRML are gaining currency because they are easy to use and understand (Do 2001; Jung et al. 2001). VRML operates on a low-bandwidth internet connection. The added value in the easy editing of VRML code is that one can see very quickly how the objects in a scene are controlled and constructed.

4 AN OBJECT SERVER SYSTEM

In the final experiment, the object sharing client-server architecture was based on Lea et al.'s (1996) simple multi-user system. Where the system described here differs is that its architecture allows for both object serving and navigation in CosmoPlayer 2.11 ported through Netscape. Prior to this Lea et al.'s (1996) architecture relied on Sony's Community Place Browser (standalone version). This was found to be too restrictive as it does not fully conform to the VRML97 specification. Also CosmoPlayer's renderer provided better quality real time imagery than Community Place. However, using CosmoPlayer in Netscape presents security issues when executing communication threads. Han and Turner (2001) overcame these Netscape security restrictions by using the JDBC and ODBC to link a list of users to a database to obtain the necessary permissions to create sockets. A simpler approach, used in the Object Server System described here, embeds the VRML file in an HTML page and accesses Netscape's PrivilegeManager.

4.1 System Overview

The Object Server System allows multiple clients, as avatars, and objects to be manipulated in a 3D Virtual World. Transformations of objects and avatars are directly communicated to each participating session. Clients are able to participate in the virtual environment and 'see' other client-avatars in the same scene. Avatars are simply-modified, primitive, VRML objects. Once the server is started, launching a session of the client application informs the server of new data. This data is in turn communicated to subsequent sessions. Each avatar and object has a unique identifier. VRML is the language used to construct the scenes "and JAVA is the glue that animates the world and links them to the" LAN (Han and Turner, 2001). The World model is constructed using pure VRML code. And all shared objects use VRML primitives in an attempt to reduce the rendering overheads of large polygonal models. In tandem with the Virtual World representation is a CGI scripted chat channel. Chat channel logs are saved for later analysis. Clients navigating around the Virtual World use the standard CosmoPlayer controls.

4.2 System Architecture

Based on Lea et al.'s (1996) multi-user system architecture, the Object Server System modifies the MuClient and MuReceiver code respectively. Whereas in Lea et al's basic application the avatars use a proximity sensor to update their position in the scene, using the proximity sensor means that *only* navigation through the scene triggers an event. This works for avatar-only multi-user worlds but when we add object manipulability any interference with the proximity sensor updates *all* objects in the scene.

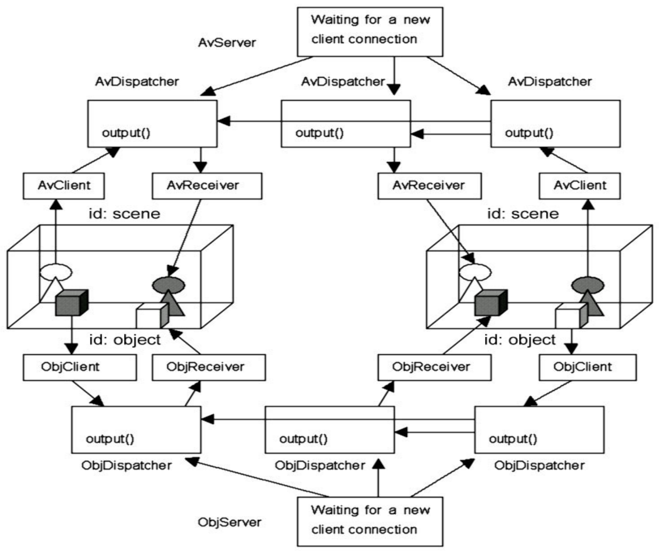

Figure 5. Object Server System Client-Server Architecture

In other words, the scene is assigned a unique identifier rather than the objects it contains. The system described here updates the transforms of the objects explicitly allowing their values to be shared. Shared objects use a plane sensor to provide output to the server. In this case multiple objects can communicate through a single client-server combination. A separate server is used for both avatars and objects. Six principle files were used to communicate and update multiple client scenes: Server, Dispatcher, Protocol, Client.wrl, Client.java, and Receiver.

4.2.1 Server Side

A separate server is used for both avatars and shared objects respectively. The two servers, AvServer and ObjServer communicate on different port numbers. Once the server is started it listens for communication from a number of sources. And upon starting a session of the multi-user world in a browser the Client Script contained initialises an instance of the class Client for avatars and objects alike. These classes communicate information about the position and orientation of avatars and objects in the scene. The respective servers track and communicate their locations until terminated. Other classes that communicate through the server are Receiver and Dispatcher respectively. The class Receiver receives information about other clients and passes this on to the local scene. The server creates a separate instance of class Dispatcher for each client. Once initialised it starts a new thread. A new identifier is assigned to each client and is stored in Dispatcher for client management. A single thread is used to handle that client. Parallel multiple threads manage connections from all clients which wait for requests from individual clients. When there are no requests, the thread blocks and waits; at the same time other threads can be running and handling requests simultaneously.

Server Side Java Application
```
import java.net.*;
import java.io.*;
import java.util.*;
class ObjServer{                            // opens sockets for communication
   static Vector clients = new Vector();     // allows multiple communication threads
      public static void main(String[] args){
         ServerSocket server_socket;
         Socket client_socket;
         int id;
         new ServerSocket(ObjProtocol.PORT); // open socket on PORT
         client_socket                      // input stream from clients
         new ObjDispatcher                  // starts dispatcher stores id.
}
```

The class Dispatcher creates a thread for input and output to and from its client. It reads messages from its client and dispatches messages to all the other clients.

Server Side Java Thread
```
import java.net.*;
import java.io.*;
import java.util.*;
class ObjDispatcher extends Thread {
   DataInputStream;
   DataOutputStream;
   Vector clients;                       // list of other dispatcher instances.
   public void run(){}                   // reads messages from a client and
                                         // dispatches them to other clients.
   public synchronized void output(){}   // ensures only one thread executes
                                         // at any one time.
```

The class Protocol provides a common port number and move or rotate for server and client alike.

Server Side Java Protocol Constants
```
public class ObjProtocol {
   public final static int PORT = 4130;
   public final static int MOVE = 1;
   public final static int ROTATE = 2;
```

4.2.2 Client Side

When the VRML file is started in a browser an instance of the class AvClient and ObjClient is initialised. Moving through the scene triggers an event by changing the ProximitySensor values. This information is communicated to the AvClient script which in turn is communicated to all other sessions operating. Selecting and moving a shared object in the scene trigger the PlaneSensor values which is communicated to the ObjClient script which similarly communicates transform information to all other sessions operating. A Heads Up Display (HUD) is provided with a toggle switch to turn on and off sensors associated with objects in the scene. This facilitates easier navigation through the scene by avoiding the triggering of unwanted object transforms.

Client Side VRML Virtual World

```
#VRML V2.0 utf8
DEF PS ProximitySensor {}              // tracks avatar movement through the scene
DEF AvClient Script {                  // launches client script for communicating
                                       // avatar transform updates
  mustEvaluate TRUE
  directOutput TRUE
  url "AvClient.class"
  eventIn   SFVec3f  AvPosition
  eventIn   SFRotation avOrientation
  field     SFNode   avatar
DEF AVATAR Transform {}                // defines model of avatar
ROUTE PS.position_changed TO AvClient.AvPosition
ROUTE PS.orientation_changed TO AvClient.AvOrientation
DEF HUD Group {                        // provides console for toggle switch
   children [
      DEF HUDProx ProximitySensor {}
      DEF SWITCH Transform {
      DEF TOUCHSENSOR TouchSensor {}
              }
         ]
       }
DEF TOGGLE Script {                    // toggle script used to disable plane
                                       // sensor of shared objects in scene
   eventIn   SFBool  inBool
   eventOut  SFBool  outBool
   field     SFBool   start FALSE
}
DEF ObjClient Script {                 // launches client script for communicating
                                       // object transform updates
   mustEvaluate TRUE
   directOutput TRUE
   url "ObjClient.class"
   eventIn   SFVec3f  ObjPosition
   field     SFNode   object }
DEF OBJECT Transform {}                // defines model of shared object
DEF MOVE PlaneSensor {}                // sensor used to transform shared objects
   USE OBJECT
ROUTE HUDProx.orientation_changed TO SWITCH.rotation
ROUTE HUDProx.position_changed TO SWITCH.translation
ROUTE TOUCHSENSOR.isActive TO TOGGLE.inBool
ROUTE MOVE.translation_changed TO OBJECT.translation
ROUTE MOVE.translation_changed TO ObjClient.objPosition
ROUTE OBJECT.translation TO MOVE.offset
ROUTE TOGGLE.outBool TO MOVE.enabled
```

The class AvClient, once initialised, communicates avatar movement to its corresponding Dispatcher class. Both AvClient and ObjClient initialise their own Receiver classes. The proximity sensor in the VRML file tracks the user's position in the scene. By reading the fields in the AvClient Script node a socket is created with the Server. This sets up the input and output streams. Receiver is used to read data from the Server and pass it on to the objects in subsequent scenes. AvClient gets permission from Netscape to create a socket using the Privilege Manager. Where AvClient and ObjClient differ is in the way an identifier is assigned. For AvClient AvReciever calls methods from AvClient and updates the assigned scene, whereas ObjClient methods are called for each assigned object.

```
Client Side Java Script
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.net.*;
import java.io.*;
import java.util.*;
public class ObjClient extends Script{
   public final static int OBJECTS       // number of shared objects in scene
   private final static ObjReceiver      // initialises ObjReceiver
   SFVec3f ObjectTransform;
public void initialise(){}               // gets value from exposedField
                                         // of shared object.
public void processEvent(){}             // sends id and current transform to server.
public void updateTransform(){}          // communicates transform update from
                                         // receiver to browser.
```

The class Receiver is different for both avatars and shared objects. Typically it receives position update events from the VRML scene-client and dispatches them to the Server. In particular AvReceiver receives messages from AvDispatcher via the server and all other dispatchers and calls the relevant method in AvClient updating objects in the scene. For the shared objects', ObjReceiver assigns a unique identifier to each object within the scene allowing the sharing of more than one object by subsequent scenes. It gets permission from Netscape to create a socket using the PrivilegeManager.

```
Client Side Java Thread
import vrml.field.*;
import java.net.*;
import java.io.*;
import java.util.*;
import netscape.security.PrivilegeManager;
import netscape.security.Target;
class ObjReceiver extends Thread {
   public final static String HOST      // server host name
   private static int nextId;
   DataInputStream;
   DataOutputStream;
   Socket;
   new ObjClient                        // get ObjClient.Objects from ObjClient class
   ObjReceiver(){
      super("ObjReceiver"); }           // creates sockets for data in and out.
public int getId() {}                   // starts thread
public void run(){}                     // Receive client's id and its
                                        // transform from server.
                                        // Then updates the object's
                                        // transform node.
                                        // To reduce the amount of
                                        // communication traffic
                                        // float values are converted to integers.
```

Where the architecture of this multi-user Object Server System primarily differs from Lea et al’s (1996) is that instead of the class ObjReceiver simply passing on the information for each session it now passes information related to individual objects within the scene allowing for multiple callbacks. When the update methods are called from within class AvClient only one instance can be called. In other words, only one operation is performed within the world – in Lea et al's (1996) World, that is movement and/or rotation. If ObjClient were constructed similarly to AvClient a conflict would occur when more than one object is controlled by any individual scene. To facilitate the manipulation of objects, we need to be able to perform multiple operations on multiple objects. Each object needs to be served and in turn this information made available to all other browser instances. To achieve this, each object's unique identifier communicates directly with the class ObjReceiver.

5 CONCLUSION

VRML lent itself as a collaborative design research tool because it was easy to understand and its primitive rendering - sans shadows and reflections - assisted smoothly interpolated multi-user worlds. And while some computer games, such as Quake, are supported by world-building tools, VRML's simplicity facilitated rapid prototyping of Virtual Worlds. VRML proved to be a robust, online, multi-user environment using 3D real time navigation.

Future directions for the use of the Object Server System includes the sharing of other transforms such as scale and rotation. Indeed, any ExposedField can potentially be shared. The addition of a HUD console to place controls on can assist the switching of these field types. And creating multi-user 3D Worlds where participants can manipulate objects *on the fly* encourages collaborative exploration in architectural form, space and place without the risk and constraints of real-world production. Furthermore, the increased speed and bandwidth of fibre optic cabling makes practical extension of this system, beyond the LAN.

REFERENCES

Clark, S. and M.L. Maher. 2001. The Role of Place in Designing a Learner Centred Virtual Learning Environment. *Proceedings CAADFutures 2001,* eds. B. de Vries, J. van Leeuwen, and H. Achten. Eindhoven: Kluwer Academic Publishers

Damer, B., S. Gold, K. Marcelo and F. Revi. 1998. *Inhabited Virtual Worlds in Cyberspace*. Http://www.digitalspace.com/papers/virtual Worldpaper/virtual World98chap.html (20-05-2000).

功能變數代碼變更

Do, E.Y.L. 2001. VR Sketchpad: Create Instant 3D Worlds by Sketching on a Transparent Window. *Proceedings of CAADFutures 2001,* eds. B. de Vries, J. van Leeuwen, and H. Achten. 162-172. Eindhoven: Kluwer Academic Publishers

Han, S.H. and J.A. Turner. 2001. An Architectural Approach to Virtual Reality Support of Multi-user Environments. *Proceedings of CAADFutures 2001*, eds. B. de Vries, J. van Leeuwen, and H. Achten. 439-452. Eindhoven: Kluwer Academic Publishers

Jung, T., M.D. Gross and E.Y.L. Do. 2001. Space Pen: Annotation and Sketching on 3D Models on the Internet. *Proceedings of CAADFutures 2001*, eds. B. de

Vries, J. van Leeuwen, and H. Achten. 258-270. Eindhoven: Kluwer Academic Publishers

Lea, R., K. Matsuda and K. Miyashita. 1996. *Java for Three-dimensional and VRML Worlds*. Indianapolis: New Riders.

Lea, R., H. Yasuaki, K. Matsuda and S. Matsuda. 1997. *Community Place: Architecture and Performance*. Tokyo, Japan: Sony Architecture Labs. http://www.csl.sony.co.jp/person/rodger/VRML/PAPER/vrml97.html (20-11-01).

Maher, M.L. 1999. Designing the Virtual Campus as a Virtual World. *Computer Supported Collaborative Learning (CSCL99),* 376-382.

Mitchell, W.J., S. Yee, R. Naka M. Morozumi and S. Yamaguchi. 1998. The Kumamoto-Kyoto-MIT Collaborative Project: A Case Study of the Design Studio of the Future. *Cooperative Buildings, Integrating Information, Organisation, and Architecture, Proceedings of the First International Workshop, CoBuild'98, Darmstadt, Germany*, eds. N.A. Streitz, S. Konomi and H-J. Burkhardt. 80-93.

Oxman, R., A.D. Radford and R. Oxman. 1987. *The Language of Architectural Plans*. RAIA, ACT.

Radford, A.D. 1994. Local Architectural Language and Contextualism, *Design Review: Challenging Urban Aesthetic Control,* eds. B.C. Scheer, and W.F.E. Preiser. 165-174, Prentice-Hall

Richens, P. and M. Trinder. 1999. *Exploiting the Internet to Improve Collaboration between Users and Design Team, The Case of the New Computer Laboratory at the University of Cambridge, Proceedings of CAADFutures 1999*, eds. G. Augenbroe and C. Eastman, Dordrecht: Kluwer Academic Publishers.

Strehlke, K. and M. Engeli. 2001. [roomz] & [connectionz]: Scenarios in Space and Time. *Proceedings of CAADFutures 2001*, eds. B. de Vries, J. van Leeuwen, and H. Achten. 173-186. Eindhoven: Kluwer Academic Publishers

Sudweeks, F., M. McLaughlin. and S. Rafaeli. 1998. *Network and Netplay: Virtual Groups on the Internet.* London: MIT Press.

Wyeld, T. 2001. What does it Mean to be a Column?: A Day in the Life of an Avatar, *Proceedings of Playful Design Learning Forum*. (CD-rom). South Australia: Adelaide University.

功能變數代碼變更