Creating
Collaborative Virtual Environments
Using Unicon

Wynn H. Winkler

New Mexico State University
Department of Computer Science

December, 2004

Project Report for the Degree of
Master of Science in Computer Science

Abstract

The NMSU CS department has begun a project to build an open source, interactive, multi-user, networked 3D environment which is suitable for distance learning and group collaboration. This project is described in detail at www.cs.nmsu.edu/~jeffery/vcsc/. The platform for this project is the Unicon language and associated libraries.

Several persons have contributed code to various phases of this project – these include some basic 3D architectural graphics, event handling, client-server networking, a 2D chat window and commands, a texture library, and animated avatars. Each of these has been done as separate experimental prototypes that demonstrate how to get certain features to work. The task of getting these pieces integrated and re-organized into one coherent whole constituted a project in itself. This paper describes this overall integration phase of the project and the resulting prototype Collaborative Virtual Environment (CVE).

Table of Contents

1 Introduction

3D graphical environments as seen in today's computer games and virtual world simulations provide the user looking into the terminal with the amazing experience of moving around in and interacting with a simulated world. This has stimulated a great deal of interest in using such technologies for purposes other than simulated mayhem – such as learning, teaching, and collaboration.

This is the overall motivation for the Unicron project described at www.cs.nmsu.edu/~jeffery/vcsc/ . The goal of this project is to produce an online virtual world with a 3D graphical interface running on the user's local machine - an interactive, networked 3D extension of a university computer science department. The two most immediate uses will be distance learning within the New Mexico state college network and group collaboration.

For a project such as this there are a large set of issues involved. There is quite a large search space of platforms which might be usable, and searching is expensive in this case due to the need to actually work with and learn how to use the candidate platforms. In the first semester of this project, substantial investigation was done in the areas of Virtual Reality Modeling Language (VRML) software, and open-source 3D game software to see if any viable environments could be found.

VRML can render 3D graphics and provide some interactive behavior via javascript. In the early days of the web, VRML plugins in web browsers were expected to be a major line of development, and a great deal of work was done with this language. Unfortunately, it ran into major problems with browser standardization, performance limitations, and platform portability.

VRML did not turn out to be a major factor in web software, but it is used in many contexts. Its open standards make it a popular file transfer format for 3D editors, and many examples, tutorials, and some software are still available for it. A major motivation for VRML was to create an open standard for 3D graphics files. The desire for open 3D graphics standards is still very powerful, and the next generation of this effort centers around the X3D language. The VRML archives are hosted at http://www.web3d.org/x3d/vrml/index.html. See the section of this paper entitled Related Work for an extended discussion of the search results. More details on this search are included as an appendix to this paper.

The major conclusions of this search are:

1 There are very few 3D editors which produce VRML code that would be suitable for our purposes.

2 Given a file of VRML code, it is either necessary to run it within a proprietary and closed-source browser, or find a way to display and manipulate it within an open-source coding environment. This latter is a major project itself.

3 The open-source and near-open-source game engines are a possible platform for this work. Their major drawbacks are that they are written in C/C++ and require a large committment of low-level coding time and effort. Also, most of the graphics and avatars are still proprietary and/or inappropriate for our application. Thus it is

generally necessary to re-do all the graphics work, particularly the avatars and textures in their own proprietary formats, which is a major effort.

4 Other open source approaches include the Alice project (www.alice.org) based on Python and later ported to Java via Java3D. Also worth noting is the Croquet project (www.opencroquet.org). This is based on the Squeak/Smalltalk languages. The latter looks interesting but is still not quite released. The current incarnation of the Alice project is not quite what we want, either.

Ultimately, although there are a lot of related projects, we did not feel confident that any of them were stable, portable, and easily extensible enough for us to meet our goals using their platforms.

This led to the challenge of putting together a complete CVE using pieces of demonstration code which have been previously developed within the Unicron project here at NMSU and this is the path which has been taken.

Unicron is written in the Unicon language. This is one of the less well-known languages in circulation, and thus some comments on the this language are worth including at this point. The roots of Unicon begin with the SNOBOL language which was moderately well-known in the 1960's and 1970's. SNOBOL was well regarded for its string and pattern matching capabilities and for general text processing work. Proponents of SNOBOL have gradually migrated to its successor, Icon, during the 1980's and 1990's. The home of the Icon language is at http://www.cs.arizona.edu/icon/. The Unicon language is an extended dialect of the Icon language developed by a separate group at http://unicon.sourceforge.net/. Roughly, its goal is to modify/add to Icon the features that are expected for modern languages.

These features include:

1 Open standards, and Open-Source Licensing
2 Multiple platform support – Windows/Unix/Linux
3 Support for an Object-Oriented code structure
4 2D Graphics (note that these have also been added to Icon)
5 GUI interface
6 3D Graphics – with OpenGL being the library of choice
7 Networking via standard internet protocols
8 A respectable IDE environment including an interactive debugger, an editor or an interface to one, and object browsing capabilities.

Over the recent few years, serious implementations of 1-7 of the above have been developed and incorporated in the main languge distribution. The IDE and debugger are probably best considered a work-in-progress.

While these capabilities have been developed and included directly in the language, or as libraries, there has not been a lot of application development done which exercises them extensively. This is particularly true of the 3D Graphics API, which was developed

at NMSU in 2002-2003. Thus, a parallel motivation for this project is to build an application which exercises the 3D libraries and addresses the following issues

1 Unicon is a compiled language, but has many features characteristic of scripting languages –dynamic variable typing, built-in data data structures for strings, lists, hash tables, records, and automatic memory management and garbage collection, are some notable ones. It is often believed that scripting and/or interpreted languages do not have adequate performance for interactive 3D graphics applications –particularly games and simulated worlds. Since it is compiled, it is possible that Unicon can deliver adequate performance while creating a more programmer-friendly environment than the classic C/C++ languages generally chosen for this type of application. This issue needs to be addressed with demonstration applications which provide adequate subjective response times at the keyboard and mouse.

2 Similar questions arise for the networking libraries – can they handle the bandwidth required to give respectable performance to a 3D simulated world.

3 It is to be noted that Unicon's 3D graphics and networking libraries are not direct wrappers for the underlying system-level OpenGL and sockets API's. Considerable effort has gone into providing a simplified programming interface to the Unicon developer while still including access to the most useful features of these API's. This can create two problems. First, simplifying things for the programmer often means that there is extra overhead buried within the programming environment. Second, some of the lower level capabilities of the API become inaccessible. Thus, there are questions as to whether the gains in programmer time and effort are consistent with adequate performance for the end-user, and even whether certain application features can be implemented at all.

4 In addition to performance issues, the libraries are currently documented at the function call level. For developers who want to learn how to put those functions together into working applications the major resource is studying the small demonstration applications from the Unicron project. These are generally proof-of-concept implementations which address only one aspect of a complete application and often have inconsistent coding styles. Issues such as the best way to structure event-loops, and how to manipulate the Unicon interface to the 3D OpenGL display lists, and how to integrate 2D and 3D graphics in the same application, are still topics for considerable experimentation. Some extensions to the Unicon

graphics libraries will come out of this work as a side-benefit.

5 There is also an issue of software development methods. Is it a useful exercise to jump in and start hacking together prototypes, even partial ones, with a philosophy of "just make it work" and then do more systematic design later vs. stand back, look at everything and try to do a systematic design process before trying to do much of the code. This project definitely emerges from the first approach rather than the second. It should be noted that some of the early prototypes were done because it was uncertain that these features could even be implemented in Unicon - under those conditions, top-down design is probably an unworkable approach. This issue is re-visited in the Evaluation section.

From a programmer's standpoint, Unicon “feels” more like a scripting language, with Python being perhaps the closest or most obvious comparison. In this project, the string/text handling features derived from SNOBOL were little used. The major similarities with Python are in the control structures, dynamic typing, automatic memory management, class structures, and a fast compile/link process. The major differences are that Unicon's compiler creates a stand-alone executable (the VM is actually embedded in this executable file), Unicon has higher-level interface libraries to graphics and networking, and the size of the user base.

The work in this project entailed taking all the pieces of demonstration code which were done in the last year or two and pulling them together into one consistent coding framework and adding a lot of features. A separate section of this report will discuss the modifications which were needed here, but basically, several sections of the code have been nearly completely re-written – this includes the 3D graphics, server, command/chat window, and event loops. Avatar networking was added from scratch using the avatars which were designed by another student.

2 Requirements

The above considerations, plus an educational focus, suggest the following as a reasonable set of initial requirements for the CVE prototype:

a. The initial implementation will create a 3D environment as much like the NMSU CS department as possible.NMSU's Science Hall is in many respects a generic modern office building with generic modern offices, classrooms, and computer labs . This may seem dull when compared to the computer games today's students play. On the other hand:

- It does get you ready for the real world.
- Distance learning students from other parts of the state will know what to expect if they visit or transfer here.
- We are programmers, not graphic artists or industrial designers - working in a software modeling environment allows us to avoid pitfalls and challenges involved in the architecture of the space that is being constructed..

b. The file formats involved should be non-proprietary and based on open standards. Small files are a big plus due to network transmission time and local disk storage requirements. The file size/disk storage issues are of particular concern since distance learning efforts will be initially targeted at other state schools and community colleges where the computer hardware may be 1-2 generations behind our development equipment

c. Following open source philosophy, cross-platform support for Windows, and Linux is a goal. Later, the project will survey the systems in use at other state colleges and it is quite possible that cross-platform support will become not just a philosophy, but will be required due to our in-state variability in computing environments.

In accord with these general requirements, the set of features which have been implemented so far include:

1. 3D/OpenGl rendering of a few rooms of the Science Hall environment. This includes tiled textures, moveable doors, interconnecting rooms and corridors, and basic classroom features such as windows and whiteboards.
2. Each user has a separate avatar which is controllable from their workstation.
3. Support for multiple networked users, with each of their avatars being visible to other users while being controlled from their own terminal. Avatar movements are automatically mirrored on remote terminals.
4. A 2D chat/command window is available for chat with other users. and simple commands have a standard parse and execute sequence.
5. A server which supports a client/server architecture and implements the message passing and processing required for the above.

6. An event handling paradigm which integrates all the events and messages for the above.
7. All of the above in one clean and consistent body of open-source code.

3 Related Work

Both 3D games and graphics and CVE's are a very active area of software development and there are numerous related and semi-related projects which may be of use in constructing a CVE. These include:

3.1 VRML software

This is interesting for two reasons. First, lots of develoment effort went into this area in the late 1990's and should have produced some useful results. Second, it is the only mature open-source 3D file format standard. Quite a few 3D editors will export some form of VRML code, and there are viewers available free. This is the reason so much effort was spent in finding, testing, and evaluating the available VRML software. An appendix goes into detail on this work.

In terms of our requirements, VRML has four problems:

a. It is decent for the architecture but very difficult to do moving avatars

b. There are very few 3D editors that produce well structured, readable, and efficient VRML code – we were only able to find one – the Pharus editor from www.int3d.com.

c. There are very few browsers which can be used to display VRML, and most are closed-source and/or proprietary, and run as plug-ins for a web-browser. This means that if you want to add features, they cannot be added at the viewer level, instead you must find some way to code them in the VRML itself, generally in Javascript, and there are significant limits to what can be done here.

d.Converting VRML code to render in some other 3D engine is a major project in itself

For these reasons, VRML is not currently being pursued for this project. See Appendix A for more details.

3.2 Open-Source 3D Game Engines

This is also a major problem with 3-d game editors.

Quark (quarkplusplus.sourceforge.net)is a large open source project to produce a 3D game or level editor. There are 2 or 3 others also available. These output the proprietary formats required by the games to which they are targeted. The commercial game companies have never objected to this, so the formats are probably de facto open, but they are still on slightly questionable legal ground from the open-source standpoint.

ID Software has released Quake II/III as open source software. This includes only the engine code written in C, not the textures, level maps, animations, monsters, sound, etc. There is not much documentation available.

The Crystal Space project is at http://crystal.sourceforge.net is written in C++/OpenGL

and is an active community project. It has most of the features expected for a modern 3D game. The project is actually targeted to building a toolkit from which to build 3d games and applications. This means there is no one standard engine - there are several demonstration applications, but you may have to build your own or do substantial modifications to one of the demos to get what you expect.

The Planeshift (www.planeshift.it) is an opensource project with the goal of building an EverQuest-like online virtual world. They use Crystal Space as their 3D toolkit. The project has very impressive graphic art work, and does work over the internet, even on a dial-up connection, but it is has not yet implemented very many activities beyond a chat room.

The Torque engine (www.garagegames.com) is commercial, but cheap and has possibly acceptable licensing terms. It has a very impressive demo, and has the features required for a 3D networked, multi-user CVE.

The main advantages of these environments is that they can provide a complete working display with user interaction and networking already in place. Some of the file formats have become "open" in a practical sense, and there are good open source level-editors available for them.

The main problems with all of the above are the spotty documentation, and the need to work with C/C++, which requires large amounts of programming time and effort, the need to rewrite chunks of the engine to eliminate monsters and mayhem, and the need to develop your own graphics from scratch. In summary, there are some interesting possibilities here, but they require more resources than we have available.

3.3 Other CVE Engines

We have done only cursory experimentation with these, or read academic papers on them. The will be described mostly by quoting from their websites in order to give a sense of what may be available. The main problems with them are that no source code is available, or they are just starting to release their first version, or as with the Alice project, they aren't quite doing a CVE in this version of their work.

a In the academic field there have been numerous papers written, and a few sites have published working CVE's. The Swedish Institute of Computer Science is well-known for the Distributed Interactive Virtual Environment (DIVE) project (dive.sics.se). They release binaries but apparantly not source code. The following description is taken from their web site:

Begin quotation:

Users navigate in 3D space and see, meet and collaborate with other users and applications in the environment. A participant in a Dive world is called an actor, and is either a human user or an automated application process. An actor is represented by a "body-icon" (or avatar),to facilitate the recognition and awareness of ongoing activities. The body-icon may be used as a template on which the actor's input devices are graphically modeled in 3D space.

The dynamic behaviour of objects may be described by interpretative scripts

in Dive/Tcl that can be evaluated on any node where the object is replicated. A script is typically triggered by events in the system, such as user interaction signals, timers, collisions, etc. In a typical Dive world, a number of actors leave and enter worlds dynamically.
End Quotation.

b Another project which has attracted interest in the academic community is the MASSIVE (Model Architecture and System for Spatial Interaction in Virtual Environments). Some notes from its website:

Begin Quotation:
http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE/
http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-2/

MASSIVE (version 1) is always and only a teleconferencing system and an example implementation of the spatial model of interaction - its not a general-purpose VR application development environment. It is a multi-user distributed V.R. system which runs on Sun and SGI platforms. MASSIVE is not generally available except to our project partners as noted above.
The current version of the CRG Virtual Environment (CVE) is also referred to as MASSIVE-2 . CVE is a distributed multi-user virtual reality system, current features of which include: networking; an extended spatial model of interaction, including third parties, regions and abstractions; and multiple users communicating via a combination of 3D graphics, real-time packet audio and text. We expect to make a developers version available (initially for SGIs only) around the middle of February 1997. A port to Windows NT 4 is expected around April 1997.
End Quotation

c The Alice project (www.alice.org) based on Java/Java3D and Python/Jython. This system introduces students to programming via scripting the actions of 3D characters and environments. From their web site:
Begin Quotation:
Alice v2.0b is the next major version of the Alice 3D Authoring system, from the Stage3 Research GroupCarnegie Mellon University. It has been completely rewritten from scratch over the last two years.The focus of the Alice project is now to provide the best possible first exposure to programming for students ranging from middle schoolers to college students.
End Quotation:

d The Croquet project (www.opencroquet.org) based on squeak/smalltalk is just starting to release its software.
Begin Quotation

CROQUET IS....
a combination of open source computer software and network architecture that supports deep collaboration and resource sharing among large numbers of users. Such collaboration is carried out within the context of a large-scale distributed information system. The software and architecture define a framework for delivering a scalable, persistent, and extensible interface to network delivered resources. The integrated 2D and 3D Croquet interface allows for co-creativity, knowledge sharing, and deep social presence among large numbers of people. Within Croquet's 3D wide-area environments, participants enjoy synchronous telepresence with one another. Moreover, users enjoy secure, shared access to Internet and other network-deliverable information resources, as well as the ability to design complex spaces individually or while working with others. Every visualization and simulation within Croquet is a collaborative object, as Croquet is fully modifiable at all times. Users and groups of users can author and publish their individual resources within a persistent 3D knowledge architecture. They may build any number of private or shared "worlds" instantaneously, making them immediately accessible for others to explore by providing spatial portals. These portals function much like hyperlinks do within the World Wide Web. But unlike the Web, Croquet enables the user to find and get to other individual worlds through the larger context of Croquet's persistent common spaces. Croquet is also a complete development and delivery platform. Its infinitely scalable architecture provides it with enormous possibilities as an operating system for both local and global informational resources.
End Quotation.

3.4 Other Potential Languages

There are other languages available which may be capable of meeting the requirements of this project while working at a higher programming level than C/C++. Possible candidates would by Python with the available C graphics and networking libraries, and squeak/smalltalk. These have not really been compared to the current approach due to the resources which would be required.

4 Design

4.1 Introduction

Figures 1 is a screenshot of the CVE showing several avatars in a group photo in a simple room which is designed for testing tiling of textures, movement of doors and avatars, and connections to other rooms. The faces are gif files derived from actual photographs, the avatars are constructed from basic geometric shapes, and can move their limbs a bit as they walk. Since a given instance of the CVE client has only one avatar of its own, group shots require several clients to be logged in simultaneously on the same server.

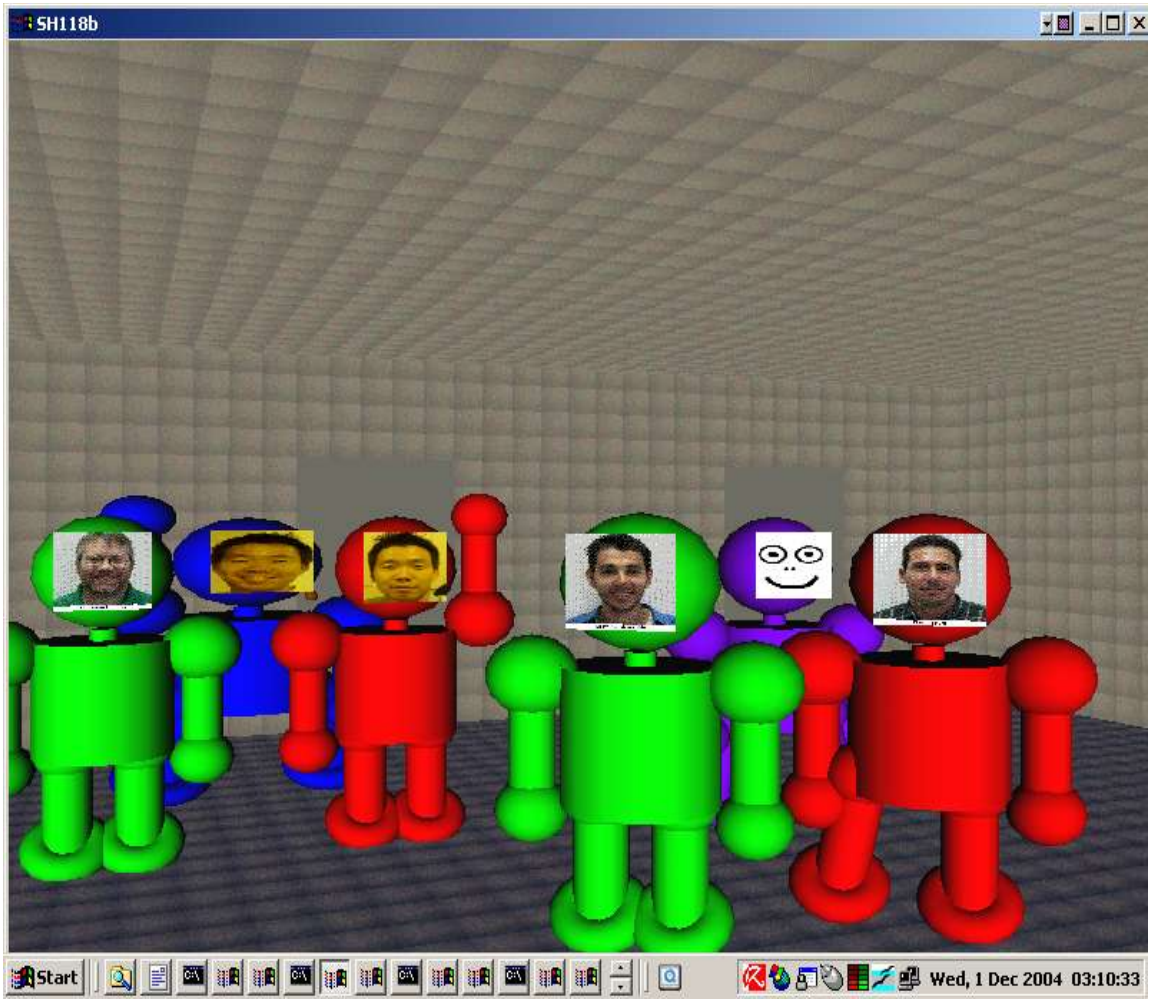

Figure 1 Screenshot

The code is structured in a strictly object-oriented manner. UML diagrams for the primary class relationships are presented in Figure 2. A more detailed discussion of the inner workings of the classes is presented below.

Coordinates

The coordinate system for this model sets the origin at the NorthWest corner of the NMSU Science Hall building. This is the location of Frank Harary's (Professor emeritus and noted graph theorist) office so these are known as Frank Harary Normal (FHN) coordinates. The positive x direction runs to the East, the positive z direction runs South and the positive y direction is vertical upwards, all of them conceptually parallel to the existing walls rather than exact compass points. The unit of distance is the meter and all x,y,z coordinates are expressed in meters. The unit of angle is normally the degree, but this value needs to be converted back and forth to radians when calling the trigonometric functions (see Unicon rtod(), dtor(), sin(), cos()). This places all of Science Hall, and the CS department, in the positive quadrant. This makes visualizing the rooms and their relations in the code somewhat easier, but means that the code has not been tested well with x,y,z values in the other quadrants.

3D Architecture

Start with 4 points that make a rectangle. Thats a wall, a floor, or a ceiling. 6 of these rectangles make a box. A room is a box with some added features - exits, obstacles, and decorations like whiteboards and windows. To make an exit - first make an opening by subdividing a wall into smaller rectangles and then delete the one you want open. A door can be put in the opening if desired. Keep the walls parallel to the axes to simplify the math. The OpenGL API pretty much handles all the 3D transforms internally.

The data structure for the architecture could well be described as the simplest thing that works, and does not use any of the more esoteric tree structures which are common in modern game engine code. The fundamental element is an instance of the Wall class. This is a set of 4 3D points which define a rectangle, and an associated texture which will be tiled over the surface at rendering time. The 4 3D points are maintained as a list of 12 numeric values. Walls (and polygons) are rendered with both faces visible. Texture tiling factors are determined by dividing the wall's length in each direction by values which represent the real world length and width covered by the texture gif file. These texture sizes are currently hard-coded as [0.2, 0.2] meters, but will become a data field in the Texture object class which is currently being developed. Every wall must be parallel to one of the major coordinate planes , xy, xz, or yz , thus since no diagonal walls are allowed, all structures must be rectangular and aligned with the major axes. Floors and ceilings are just wall polygons which are parallel to the xz plane. These conditions work adequately with the Science Hall building, but their lack of generality is notable, and they may be changed in the future.

For the rendering and tiling of walls to work correctly, their defining points must be maintained in a standard order. Certain operations on walls - particularly creating openings and doors sub-divide walls into smaller rectangles (which become wall objects themselves). To preserve a standard order, the points are sorted by their distance from the origin by method Wall.sort_coords(). This has worked adequately so far, but is not a general solution to the problem.

The next conceptual structure is given by the class Box, which is basically a list of 6 wall objects which form a rectangular box with all right angles (the rectangular parallelopiped from geometry). The Box is the parent class of Room and is normally constructed by the Rooms constructor. The constructor for a box takes the x,y,z coordinates of the *southwest* corner of the box, and a length, height, width, which run in the positive x, positive y, and negative z directions respectively. From these it computes the coordinates of the 6 walls which compose the box and stores them in a list. This is the basic data for a Room. Note the use of the southwest corner as the base point of the room - this is somewhat arbitrary and is not intuitive given that the overall coordinate system is based on the northwest corner of the building.

The class Room is a sub-class of Box which adds lists of exits, decorations, and obstacles. Exits are either doors or openings without doors. Decorations are windows and whiteboards. Obstacles are objects such as tables, pillars, and file cabinets, where avatars cannot move or be placed. They are not currently in use and have not been tested recently.. Windows are not cutouts in the walls, they are special *.gif textures which overlay the wall and typically show some outdoor scene. Whiteboards are currently fixed textures which overlay a wall, but this will change in the next generation of the code which will support dynamic whiteboards which will update in real time as users draw in a separate 2D window.

The Room constructor creates a box. A major operation on the walls of the box is the method add_opening() which is used to create openings and doors. This is done by finding the wall which contains the coordinates for the opening, and then splitting that wall into 3 smaller wall objects/rectangles - in particular, the main wall is removed from the list of walls in the box, and replaced with sub-rectangles for the area above, the area to the left, and the area to the right of the door or opening. This leaves no wall segment in the opening itself, and the rendering process will produce an open view into the adjoining room.

The class Opening should be understandable from the above - its constructor takes a base point which must be located in an existing wall, and has default values for the width and height of the door. Since an opening must connect 2 rooms, the class maintains a list of the these 2 rooms in the field named 'rooms'. Note that when a door or opening is created, each of the rooms it connect must execute the Room.add_opening(0) method so that the opening is cutout of the walls in both rooms. See the code in SH118b.icn for examples of how to do this.

Class Door is a sub-class of Opening whose main additional feature is the ability to fill the opening with a moveable door. The door is rendered with the DrawCube function and elongated with the Scale() function. It can be textured, but the texture does not tile, so the results are often not as expected. The display list element for rotating the door about its hinge point is captured, and is later used by the methods which open/close the door.

The design does not currently have an external file format for the architectural data. This information is implicit in the parameters and the calling sequences of the room/opening/door/whiteboard constructors in the World.world_create() method and the the build() methods of the individual rooms. This may change as the project progresses.

The World class maintains a field for current_texture because the Unicon 3D graphics API can only have one texture active during a rendering operation. Since loading textures

can trigger disk operations with large time delays, the code tries to minimize re-reading textures. This project has motivated some additional development in the texture cacheing that is done in graphics library.

4.2 Class Structure

The code for this application follows the object model - there are no global variables nor procedures outside of the classes. The classes are used mainly to group related data and methods -There is very little inheritance among the classes listed in the previous section. Class Door inherits from Opening, and Room inherits from Box. Also,within the avatar file there is an inheritance structure.

There is however, a great deal of communication and cross-calling of methods between classes. It is typical for most classes to have fields for object references to the world object as well as several others. The world reference is often used to access the system parameters that it contains, as well as the other major objects such as the active window and the file that gives write access to the network.

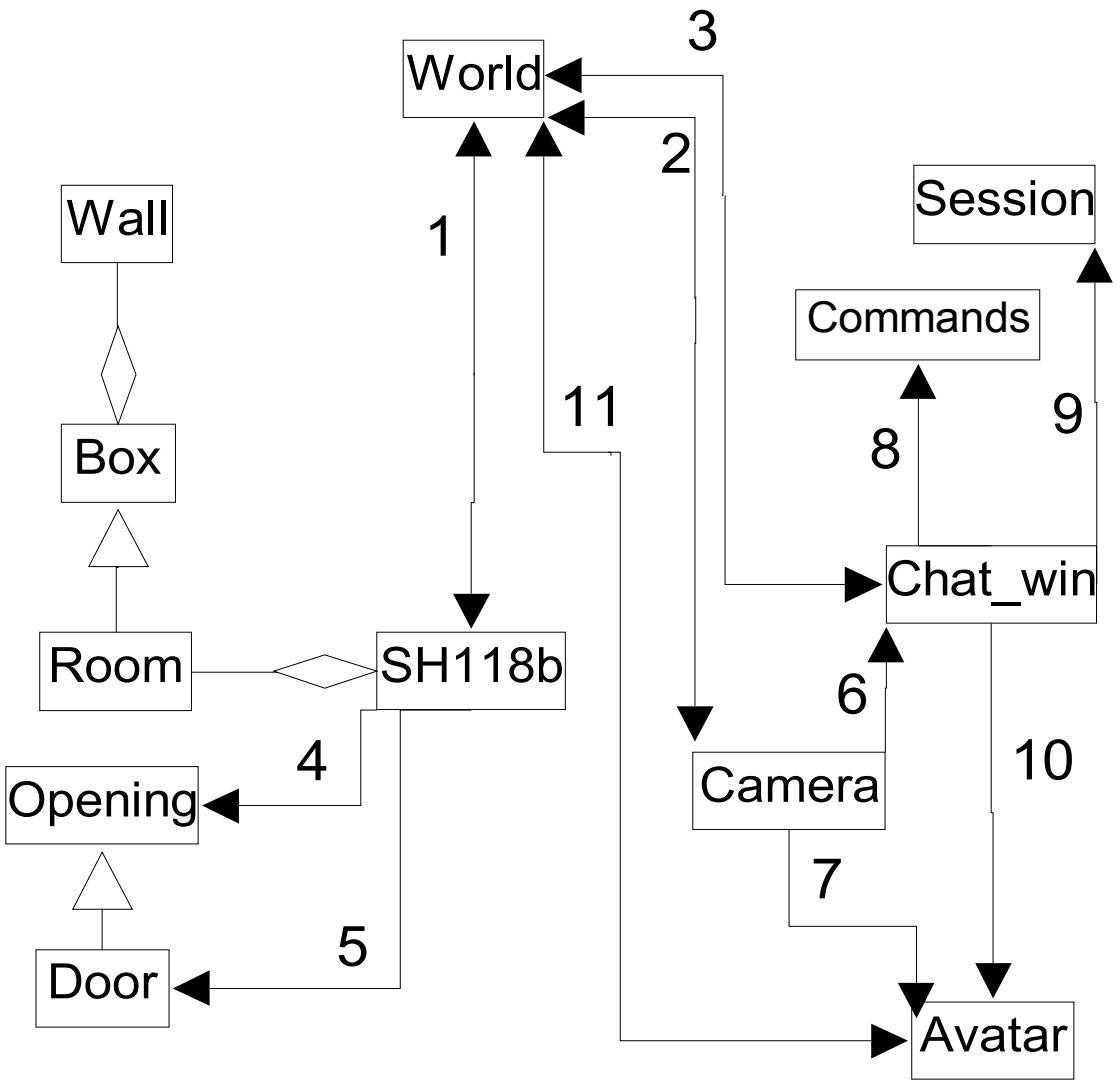

Figure 2 - UML Diagram of Major Classes

Description of Class Associations in UML Diagram Figure 2

1 SH118b is called from World to create the rooms, link their openings, and store their data back into the World.Rooms list.

2 World creates the graphics window and then calls Camera methods to respond to keyboard commands,and move the avatars, doors, and camera. Fields in world provide the system level parameters which initialize and control movement. World also passes events to Camera.handle_keyboard_input. Also, World contains the data structures for the architecture manipulated by camera.

3 World keeps the pointer to the chat window, and contains the

system level parameters which control its behavior. The Chat window accepts commands which set/reset several of these parameters, including the network file.

4 SH118b uses Opening to create openings in the walls of the rooms it has previously created.

5 SH118b uses Door to create openings with doors in the walls of the rooms it has previously created.

6 Camera creates or raises Chat window in response to keyboard input.

7 Camera moves avatars in response to keystrokes.

8 Chat window uses Commands to parse and validate input lines.

9 Chat window uses Session to store the state of the user's online session (currently only the userid).

10 Chat window commands create and delete avatars.

11 World maintains the table of Avatar pointers.

After reviewing the design, it appears that the following changes could be made:

a. Everything related to creating/maintaining the graphics window is gathered in Camera which is renamed as Graphics_window.

b. A class is placed between World and SH118b for gathering the calls to create the individual rooms and the calls to link them via openings and doors.

Internal Class Operation

a nsh.icn

This contains procedure main() which is a very simple top-level driver which creates an object of class World() and runs its main methods

b class World()

This is the major top-level object. It creates the 3D graphics window. It is a factory class for all of the rooms and all of the avatars. All of the parameters which control the application are gathered here. The render() method drives the OpenGL render step for all architecture. The event_loop() method is the initial starting point for event processing.

c class sh118b()

Each room in the simulation is maintained in a separate file and class.The major method in this class is build() which creates a Room object and then

functions as a factory class to populate the exits list with openings and doors, and to populate its decoration lists with items such as whiteboards and windows. Since openings and doors maintain lists of the rooms they conect, those are filled in here.

d class Opening(), Door(), Wall(), Box(), Room()
These are the classes which create the static 3D architecture. These classes typically have a create() method which initializes the data structures, a render() method which is called to do the OpenGL rendering, and an allows () method to do collision testing. The Room class includes method select_nearest_door() which is used to determine which door to open when the 'd' key is pressed.

e class Avatar(), Skeleton() and its sub-classes
This class and its sub-classes create, render, move, animate, and delete the avatars. These are called either from the Camera for the user's own avatar, or from Chat_win.handle_network_input() for the avatars of remote users.

f class Camera()
The handle_keyboard() method processes keystrokes in the graphics window and calls other classes as necessary. The camera and door movement methods are located in this class.

g class Chat_win()
This class provides the methods to create and draw the chat/command window using the Unicon 2D graphics library. This requires keeping track of and calculating the pixel positions to display individual characters and this can be font-dependent. The method handle_keyboard_input() parses the input lines and responds to them, usually by calling other methods in this class. Since the network commands are given from the chat window, most of the network methods are here as well.

h class Commands()
This is a utility class. It contains a list of commands accepted by the chat window and server and some parsing/validation methods and the set of valid chars accepted on a command line.

i class Session()
This is a small class which is intended to collect the data relevant to managing the users session between logon and logoff. Currently, it only stores the logon userid, but it may grow in the future as new features are added.

4.3 Event Loops

All external events enter the system via World.event_loop() which does a blocking read from any of a list of event_sources - this list includes the chat window, the graphics window and the network, depending on which of them have been activated. All this can be coded with one simple call to select(), which vastly simplifies what would have to be written in C sockets code. The event_loop() method then routes the events to handlers in the relevant classes - class Camera for the graphics window, and class Chat_win for both the chat window and the network. The chat_window and network both expect input lines formatted with a simple command syntax which they parse by splitting them on delimiters and running them through large "if" or "case" statements. This is pretty straightforward.

The handling of events in the graphics window (class Camera) is more complex and deserves description. All these events are keystrokes which are handled with a large case statement. Some events are single keystrokes which trigger one operation and are done. But the keystrokes which move the camera, avatars, and doors are somewhat more complex.

For the camera and avatar, the desired behavior is to move the object at a reasonably constant speed for as long as the key is held down, and then to quit moving immediately when the key is released. This is currently done by processing the keystroke for the key press and then putting anothe copy of this keystroke back on the end of the pending event list. To prevent this from being an infinite loop, a test is made for the key release code which corresponds to the key which was pressed - when one of these is detected, the pending event list is purged of all pending key-press codes for that key - this stops the loop and gives a rapid response to the user's key release. This technique of fiddling with the pending event list is somewhat unusual, and it may be revised in future releases.

4.4 OpenGL Display List

When OpenGL renders an object or a scene, it can build an internal list of the transform and draw operations which take place. This is done in some internal format which can be re-run through the hardware pipeline much faster than re-executing the original operations - even if they are written in a low-level language such as C. The Unicon 3D graphics API provides a similar capability by maintaining its own display list using its own data structures, which are separate from and independent of the OpenGL display list.. The Unicon display list can be accessed and manipulated at the Unicon source code level.

Manipulating the Unicon display list is a key technique for achieving several forms of dynamic behavior without re-rendering the entire scene (which is a much slower process). When the avatars are first rendered the Avatar field render_list is used to capture the sub-list of items which which are included on the Unicon display list. This sub-list contains only the items for one avatar. When that avatar disappears from the scene, say at \logout time, the sub-list for the avatar is matched against the entire Unicon display list and matching items are deleted. This takes the avatar out of the scene without re-rendering the entire scene. Other students have used a similar technique to produce

dynamically updated texture areas, such as whiteboards that can be drawn on in real time. Additional support for display list manipulation will probably be incorporated into the Graphics API.

In addition to entire sub-lists, the Unicon 3D API allows capturing individual OpenGL transforms as Unicon methods. Objects can then be moved within the scene by calling these methods with updated parameters and then calling the Refresh() procedure. This prevents re-rendering the entire scene for each avatar motion, and is the standard technique for providing animation.

4.5 Network Commands

It is also useful to describe some of the transmit/response protocol sequences that occur through the network interface. There is currently no network class. The field World.net is a file type field in class World. Network commands are sent out by doing a standard write() with this file as the first parameter - anything written in this way goes out over the network interface. This is done in the code directly at several points rather than encapsulating it in a network object. As noted above all network input comes in via World.event_loop() and is passed to Chat_win.handle_network_input(). All network transmissions must have syntax which is acceptable to the parsing utilities in the Commands class, and use command words and characters which are gathered in lists in that class. The Commands class is used by both the client and the server, and parsing and validation are done on both sides.

1. At login the command

\login uid

is sent to the server. After accepting the login the server will send an ACK/NAK message and some confirmation messages to uid. This will trigger uid to send the message

\avatar uid x, y, z

back to the server where x,y,z is the current location of uid's avatar.

The \avatar uid command notifies other users that uid is now logged in and their avatar code will read the uid.txt file and render it in their graphics window. The avatar data in uid.txt is not currently transmitted over the network, so this file must be available in the \avatars sub-directory on their machine. Keeping these files in sync across the network is currently done manually, and is a prime candidate for automation in future versions.

Our original user still does not have its copies of the avatars from users who are already logged in. The server does not yet store avatar state information - especially the current location of the avatar. To handle this, the server will also broadcast an update message to all logged-in users.

\update

this causes the other logged-in users to broadcast an \avatar uid x,y,z command for their own avatar and its current location. Users who already have this avatar displayed will ignore the message, while for the new user who began the logon process this will cause all the other logged-in avatars to display on their terminal.

2. The command

\say text

is simply read by the server, and broadcast to all logged in users with the originator's uid prepended. No filtering for groups or other criteria is done currently, but this is also a prime candidate for change in future releases.

3. When the user types

\logout

in the command window, the client appends the uid and sends it to the server. The server updates its data structures and then broadcasts

\delavatar uid

to all other users. This causes them to delete their copy of uid's avatar.

The server also sends back to the original uid a second

\logout

command, which confirms the action, and causes the client to close the socket, and delete all avatars except its own.

4. If a user is logged in, any keystroke which moves the user's avatar will generate sequences of either or both the following messages to the server:

\move uid body x y z angle

\move uid part_name dir angle

where **part_name** can be **right_arm, right_leg, left_arm,** or **left_leg**.

These will be broadcast by the server to all other users where they will cause the following actions. The first will move uid's avatar to point x,y,z, oriented in the xz plane at angle. The second will move a body part, such as an arm or a leg, to the position angle. Sequences of these body part commands are used to produce some animation in the movement.

5 Implementation

5.1 User Interface

The unicron\README.txt serves as a general gathering point for release notes, changelog information, known bugs and limitations, user instructions, and todo items. It is a good idea to consult it first. The steps for running the client are as follows.

1. Check the avatar directory and choose an avatar **[userid].txt** file which suits you.

2. Execute unicron\bin\nsh.exe - this opens in a 3D graphic window showing one of the rooms in NMSU's Science Hall. The mouse can be used to resize this window by grabbing/dragging its edges, and to give it keyboard focus. Otherwise,the mouse is not used inside this window, and only single keystrokes can give commands. See below for a list of keystrokes and their effects. No avatar is present yet. The arrow keys move the

camera position.

3. Use the 'c' key to bring up the 2D chat/command window. The bottom line of this window acts like a simple command line and supports the commands listed below.Then give the command

\avatar [userid]

from the command line. This will create an avatar in the 3D graphics window, and when that window has focus the avatar will respond to its keystrokes. The user interface currently has two separate windows for chat/commands and graphics. You can switch between them using the mouse, or if in the graphics window, the 'c' key will bring up the chat window. Both windows can be resized by grabbing/dragging their edges. The chat window can have its colors and fonts set with the \attrib command described below.

4. To logon, be sure you have a network connection, then switch to the chat window and check the server you want to connect to with the command

\server

if this is not correct, you can change it with the command

\server host:port

if you have started a local copy of the server and want to use it, then the command is

\server :4500

then to logon give the command

\login

this will log you into the server using the [userid] part of your avatar's parameter file name. The avatars of all other currently logged in users should appear in your graphics window, but their movements are remotely controlled by the user who created them. Your keystrokes will affect only your avatar.
The comand

\say text

broadcasts the text to all other online users, with your userid appended. This creates a basic chat capability.

The keystroke commands which work in the 3D graphics window are:

up arrow - move camera forward
down arrow - move camera backward
left arrow - rotate camera left
right arrow - rotate camera right
' w ' - look up
' s ' - look down
' d ' - toggle door open/closed
' o ' - reset original camera position (use if lost)
' c ' - open chat window
' v ' - toggle view-mode - inactive
' t ' - move avatar forward
' g ' - move avatar backward

' r ' - rotate avatar left
' y ' - rotate avatar right
' e ' - raise/lower avatar left arm
' u ' - raise/lower avatar right arm
' f ' - move avatar sideways left
' h ' - move avatar sideways right
' b ' - return avatar arms/legs to start position

The commands which work in the chat/command window are:

\avatar userid
creates the users avatar - only works if logged out

\delavatar userid
deletes an avatar - only works if logged out

\attrib
change window attributes - colors fonts ,this uses syntax documented for the WAttrib() function and can crash the appl if the syntax is invalid

\close
closes chat window - only works if logged out graphics window stays open

\logout
close net conection but keep chat window open this will delete all other users avatars

\say text
sends text as a message to all other logged in users

\quit!
quick kill everything

\server [host:port]
query server:port, or set server:port if not logged in

\set parm [value]
query/set speed control parameters

The following network commands are generated automatically and generally should not be given by the user:

\move uid body x y z angle

this moves uid's avatar to point specified by x,y,z, and oriented at angle, this is sent to the server and then broadcast to other users automatically when uid's avatar is moved

\move uid part part_name dir angle

this moves the part_name body part of uid's avatar indirection dir with the increment angle

\update

causes all logged in users to re-broadcast their avatar this is sent by the server when a new user logs in and will cause the new user to receive a complete set of avatars

5.2 Directory Structure

In the following discussion, the path delimiter is the Windows '\' backslash, and the terms camera and eye are often used interchangeably for the users viewpoint in the 3D graphic window.

The overall directory structure is as follows. This should be pretty self-explanatory for experienced programmers. The only notes to add are as follows. This layout will be used on all platforms and in the CVS. For network security, the system will use SSH tunneling. This requires a client executable from external sources. This will be under an acceptable license, and should be automatically configured and started from within the client executable. It will not be compiled here.

The avatars currently have a parameter file of the form [userid].txt where [userid] is automatically used as the network logon id - thus such a file must exist prior to logon. The avatars look roughly like human-shaped robots and can use a small *.gif file to display a face as a 2D rectangle. There are several faces available in the current distribution. The parameter file and *.gif file must exist on all machines which are intended to display them - they are not currently transmitted over the network to new users, but this capability will be added in the future.

Textures are organized into one master directory, which is intended to be for those which can be used anywhere, and sub-directories which are intended to be special items which only apply to specific rooms (this is a suggested organization, but it is not enforced in the code, so they can be used elsewhere if needed)

An IDE may be included in the future. This is not intended for compiling the application, but will be the functional part of the application which permits collaborative viewing/editing/debugging of code.

\unicron	top level directory - contains README file
\bin	client and server executables and any ssh clients
\dat	data files
\avatars	avatar parameter files, and face *.gif files
\textures	textures which can be applied anywhere
\sh107	textures for specific rooms
\src	all the source and make files
\shared	code compiled into both client and server
\server	server code
\client	client code
\ide	ide code (for future use)

5.3 File Structure

Most of the files contain one class of the same name. The exceptions are:

\client\architecture.icn

which contains classes Opening(), Door(), Wall(), Box(), Room() and

\client\avatar.icn

which contains class Avatar() and all its sub-classes

in addition, the following files are utilities:

\client\keycodes.txt

This is a small reference file of the press/release keycodes for the lower case alpha keystrokes. Release keycodes play an important role in managing event loops - see the Event Loop section above.

\client\amaker.icn

This is a small standalone console application which will prompt the user for various avatar parameters and then write an avatars\[userid].txt file. It is scheduled for replacement in the next generation of the avatar code.

6 Evaluation

1. The CVE does work - multiple users can log on to a single server and have the remotely controlled avatars of other users appear on their terminals in a 3d graphic environment. 2D chat is also working over the network. In some sense this is both a working proof-of-concept, and the platform for a lot of future work which will be done here.

2. Network response time seems to be marginally acceptable in a subjective sense. Now that we have a workable application, there will be many opportunities for measurment, testing, tweaking, and improvement.

3. Server robustness in handling client crashes and other network events needs attention. Some anomalies were encountered in informal testing.

4. The demonstration code samples were a very useful starting point in building this application. The techniques they use are not too complex, but are just not obvious from the textbooks and reference manuals. This is a general comment on most computer languages, not just Unicon. It is also true that as an application grows in features and complexity, the simplicity of many of the starting points gets obscured in the mass of code, and large complex applications become a difficult starting point for understanding the workings of the major techniques.

5. The higher level language features of Unicon, and the simplified graphics and networking API's have resulted in much smaller and probably, more readable code than the ususal C/C++ implementations. This translates into a faster development time, less code to maintain, and an easier-to-grasp application - which are the normal claims of scripting environments. This is certainly one of the more painless ways to bring 3D graphics and networking together and get it working.

7 Conclusions

1. It is possible to do 3D graphics of simple architecture with surprisingly simple data structures and algorithms using the Unicon API interface. The results are adequate for the intended uses, and it is likely that adding other capabilities will have higher priority than re-working the 3D graphics data structures.

2. The Unicon language and API's have adequate performance for developing multi-user, interactive, networked, 3D graphical applications on mid-range hardware, at least for applications that do not require high-speed interactions.

3. Hacking, re-factoring, and prototyping from code samples are a workable development method. There are two caveats - after something is working, serious effort has to be devoted to getting the code clean, consistent, and trying to simplify the logic.

4. This is not a true conclusion, but a general bias from experience: A from-scratch, top-down design should only be attempted by designers who have produced at least one completely working version of the desired application. This is another version of the "write one to throw away" school of development.

8 Future work

There is a nearly endless list of things which can and should be done to this prototype to make it into a usable CVE. Several other students have produced pieces which are almost ready to be incorporated - these include voice transmission capabilities, a dynamic whiteboard, a GUI interface for customizing avatars, and much more state persistence on the server side. Then there is a big wish list of items which are needed to do actual collaborative work, such as code viewers/editors/execution environments which permit multiple users to see the same view as if they were all looking at the same terminal. Then there are all the things which ought to be done to the code, from minor features, to a lot more cross-platform testing. These are discussed in more detail below.

8.1 Major functional additions

Audio transmission capabilities so that logged on users can speak to each other.

A dynamic whiteboard - the user can open a 2D window with basic drawing capability and the ability to manipulate simple shapes. This 2D canvas will be mapped in real time to the whiteboard areas in the classrooms on the 3D canvas of remote users. The intent here is to create something like a normal classroom blackboard for sharing text and simple drawings.

A GUI interface for allowing the user to set the characteristics of their avatar - this includes size, color, facial image, and other details. This will then be stored in a named file, and available for transmission over the network. This extends to other configuration parameters.

The chat system, including audio, probably should implement a concept of groups where groups can be defined by membership in some particular university class, interest in particular topics, location, or on an ad hoc basis, much like IRC chatrooms. This would permit a user to send messages only to specific individuals or groups, and similarly for voice communications. The infrastructure for this will have to be done on the server side. Similarly, the server should maintain much more user state information, permitting users to start up where they left off. A system of user authentication using accounts and passwords is another necessary addition on the server side.

At least several rooms of Science Hall with connecting corridors need to be included in CVE for it to begin to take on a more realistic feeling.

The above items are either close to being working demos or have had preliminary work done on them, and should be incorporated in the coming months. Beyond, this there are a host of additional features which exist mostly at the conceptual stage at the moment. High on this list, since this is a CS department, are features/applications which allow multiple remote users to view/edit/execute/see results/debug code.

More complex and more realistic avatars is a line of development which will be very much appreciated by everyone working on this project.

At least some items of computer equipment and office furniture should be coded so that multiple copies can be included in the architectural layout, either for ambiance, or for actual use as active objects.

8.2 Programming and Design Changes

All or several windows may become widgets within an overall 2D GUI interface.

The API suport for manipulating display lists and event lists could be upgraded.

MIME file types could be transmitted, automatically detected, and trigger external programs

The movement of the doors is not currently echoed to remote machines. This obviously invites anomalous situations where the scene on the remote machine does not correctly reflect all the actions going on, and may have other users avatars walking through doors which are apparently closed.

Additional view modes for the camera are needed. Typical first-person games place the camera at the avatar's eye so the user sees only a weapon pointing in the direction which they can fire or move. Then there are over-the-shoulder views where the camera moves in lock-step with the avatar and always sees the avatar from a constant angle. And other variations are possible.

The student work on textures should be brought in and used to enhance the looks of the rooms. Also, a more complete set of rooms needs to be incorporated in the model.

The current architectural model of wall/box/room is has many limitations - walls must be parallel to the xy or zy planes , they cannot be placed at arbitrary angles. Currently, everything in the model is rendered into the OpenGL display list and run through the graphics pipeline at each update, not just the items which are visible from the camera position. This is generally inefficient, and the wasted computation grows as more rooms are added to the model, since only one or two rooms are normally visible from a given camera position. Within the context of the current data structures, this can be improved

upon by adding a list to each room of the other rooms which are adjacent to it, and thus should be in the render list when it is the current room. Avatar movement to a different room would then cause the newly visible rooms to be added to the render list, and the ones which are no longer visible to be dropped. It is also possible to completely drop the current data structures and adopt the more complex algorithms and data structures which are used by the major game engines.

The parameters which a user might need to adjust need to be gathered in one easily accessible place, perhaps as part of the avatar creation process, and should be stored in a data file on the client. This includes the avatar parameters, speed parameters which may need adjustment for the users hardware, and possibly, the ability to remap the keyboard action keys to the user's liking.

Currently, only avatars which are included in the distribution will be correctly rendered on all machines. Local modifications and additions to the available faces will not show up on remote machines. What is needed is a way to encode and transmit avatars to remote machines when a user logs on. In addition, for a truly persistent world, the server needs to keep a complete record of the avatar's current state, and restore that state to the user at logon. This state includes the parameters which define the avatar, its current position and, in the more distant future, the state of any collaborative work it was engaged in.

Avatars do not have collision detection with each other, and this should be added.

All floors are currently at 0.00 elevation, and avatars cannot move in the vertical plane to move up stairs or ramps. There are rooms in Science Hall with elevated seating as in some theatres, so this is a possible addition.

SSH connections currently involve some manual fiddling at the command line outside of the CVE environment. This needs to be automated.

Numeric data sent over the network to update the avatar's position uses formatting which is probably inefficient.

The Unicon 3D graphics API may benefit from creating an object to encapsulate the display list and allow several operations to be done on it in an atomic fashion. In particular, it would be nice to have methods for saving and deleting portions of the entire scene, such as single rooms and avatars. Currently, each developer writes their own code for these operations, and having standardized methods would save time and produce better structured code. Similar comments to the management of event lists.

There are issues with updating graphics windows when multiple instances of the CVE are run on a single client.

It will be helpful to get the developers working from a single CVS repository, and to standardize at least some aspects of coding styles.

8.3 Testing

Most of the work so far has been done on Linux and Windows environments. Because the Unicon code and libraries are designed to be cross-platform, it should port pretty quickly to other Unix platforms, but this needs to be done and tested.

Testing with multiple users has just begun, and bandwidth issues have only been looked at in a cursory fashion.

This work has identified various bugs in the Unicon libraries which need attention.

Very little testing has been done over dial-up connections, but they will likely be very common for distance learning work within the state.

In general, server robustness in the face of transmission failures needs additional work.

Identifying a suggested minimum recommended hardware configuration for adequate performance should be done.

9 Appendix A - VRML and Level Editors

The overal motivation for this work is the Unicron project as described in detail at www.cs.nmsu.edu/~jeffery/vcsc/ . Briefly, the goal of this project is to produce an online virtual world with a 3D graphical interface running on the user's local machine essentially an environment that works like an interactive, networked 3D video game that simulates a university computer science department. The two most immediate uses will be distance learning within the New Mexico state college network and group collaboration. This open source project will include tools which allow users to modify and build their own variations of the virtual world.

3D graphical environments as seen in today's computer games and virtual world simulations provide the user looking into the terminal with the amazing experience of moving around in and interacting with a simulated world. However, the coding behind these graphics is another story. The graphical primitives which are computed and displayed are really primitive – rectangles, cubes, triangles and meshes of triangles, and some basic curved surfaces such as spheres which are actually a mesh of triangles.Getting from rectangles and triangles to a scene which looks even modestly realistic is typically very tedious and produces a lot of code most of which consists of long lists of 3-d coordinates for each of the items. This is hard to write, hard to visualize when working with the text, and very difficult to read and modify. It takes very little time working with such code to send one off in search of software which will make the process more palatable. 3D-editors allow the user to manipulate 3D objects in a 3D environment using familiar point/click/drag techniques and produce the code and the coordinate lists automatically behind the scenes. The 3D editor's output code or file is then typically viewed from within some other software module, generally called a client, a browser, or a game engine, much as HTML is produced in text editors or GUI editors and then viewed in a web browser.

The process of getting 3D graphics to appear on a 2d screen has spawned a long list of editors designed for several variants of this task, corresponding to specific types of use. There are 3D game or “level” editors, CAD/CAM editors specialized for machine parts or for architectural design, editors tuned for animations, and several other categories. An interesting new category comes from companies that enable web content providers to create virtual worlds for purposes ranging from architectural walk-throughs to personal creativity. These worlds are then hosted under various business arrangements. These companies supply the editors and some web space and/or graphic libraries which can be used to simplify creation of the users virtual worlds.

Editors, including 3-d editors, store their output in a wide assortment of file formats. Much like the early days of word processing, there are a plethora of 3D graphic file formats in existence – most of them proprietary and specific to particular vendors and/or

software environments. The legal, compatibility, and cost issues involved here preclude most formats, and hence most 3D editors, from use in our project.

A basic goal of this project is to produce completely open source code. In searching for editors we focused on those that support open file standards and gave strong preference to editors which are themselves open source.

The work that has been done up to this point has consisted of reviewing almost every non-game 3D editor that could be found on the web. These reviews consisted of reading the products web site, reading comments by other users – either on the web or on usenet's comp.lang.vrml newsgroup, downloading anything that was free and looked interesting, and buying a couple of cheap ones. Next we worked through the documentation and tutorials, and constructed elementary models of the NMSU PLEASE lab with a couple of them. Game editors generally proved unsuitable beyond the reading stage because of their proprietary file formats. Professional CAD/CAM editors were similarly unsuitable due to their cost.

We had unexpected difficulty finding an editor which met the project goals, We scanned a wide variety of 3D editing enviroments and pursued just about anything that looked cheap, workable or could be downloaded for free. Many of the editors that were easiest to use were either commercial or produced disappointing VRML output or both. Finally, we found Pharus, a web-based editor that produces outstanding output and is free for non-commercial use.

If the Pharus editor had been discovered earlier, it is likely that a lot less searching and a lot more useful work would have been done by this point in the project.

The next section of this paper discusses the goals and objectives of the main project as they apply to the selection of 3D editors. Next it provides some detail on the editors which were examined and the reasons they were rejected. A section is then devoted to discussing the Pharus editor.

9.1 The VRML Language

Currently, the only completely open 3D file format standards are VRML and X3D. Several proprietary formats could be considered semi-open in various ways but there are obvious pitfalls involved in using them.

The VRML (Virtual Reality Modeling Language) language was developed in the mid 1990's primarily to allow 3D graphics to work within web browsers. Two official standards have been released as open standards - versions VRML 1.0 and VRML 97.

There was a great deal of activity around this language in the mid to late 90's – several VRML-capable browser plug-ins were released, a fair amount of graphics demos and documentation were written, and other software begant to support VRML. This activity seems limited currently, however much of the older software and demos are still both available and free.

There is current activity around the X3D 3D graphics language which is a dialect of XML which maintains backward compatibility with VRML 2.0 and will be an open standard. This work looks promising but is far from complete. The X3D standard is not yet finalized, and the available software is limited, difficult to install, and generally not very mature. The same is true for tutorials, documentation, demos, and support from other software. For these reasons, the VRML format is currently the project choice for a graphics file format.

The VRML file format is an ascii text file with an extension of *.wrl. This means that it can be developed, viewed, and modified in any ascii text editor. Generally, VRML is written in an editing environment and viewed in a separate browser environment. As with HTML, there are several variants of this editing environment, from a plain text editor, to enhanced text editors to full wysiwyg editing environments, with several mixtures in between. Thus, there are VRML editors which are oriented to doing most of the work in text mode or by filling in parameters in text boxes and then rendering the result. We have been looking for editors that work in the opposite direction - build 3-D models on the screen using well-known point/click/drag editing, and then output the VRML code without manually fiddling with the text. It is our hope that such editors will be easier for our users to learn and use for building their own virtual worlds. Such editors require less knowledge of the internals of the VRML language and can be more visual and, hopefully, more intuitive to use. This also should translate into reduced training time as the Unicron software is rolled out to other state universities.

Having chosen VRML there are still a number of issues to discuss. In particular, a given scene can be written in VRML in multiple ways – some good for particular purposes, and some not so good. This is true for any non-trivial program in any computer language. Two issues which we have found to be of particular concern are the following:

1. Polygon mesh vs. structured objects

At the lowest level, 3D scenes are processed as polygon meshes – generally meshes of triangles. At some level within an editor, these are the data structures. When exporting the scene into VRML it is easy to produce a file which is structured as one or a few large polygon meshes. Such files can view nicely and may be convenient for transporting to other software. They have 3 problems, however – they tend to be very large, they are very difficult to understand or modify, and they tend to grow rapidly with with the number of objects such as office furniture included in the scene. As noted above, this was one of the problems with both the punchsoftware editor and the Lightwave editor. Such editors produce can produce nice-looking graphics but their VRML outputs are not as useful as

we would like.

2. Protos/externprotos for repeated objects

Many scenes use repeated copies of the same object - in our world, office furniture such as chairs, desks, PC's, bookshelves, etc. are obvious cases. VRML provides an EXTERNPROTO construct, which is much like the concept of an external function in standard programming – code it once, use it many times. This technique should be used as much as possible to structure a VRML file. Not all 3D editors do this, however, and the documentation on this issue is limited, so only hands-on experimentation shows what is happening. Thus, a key part of this work is will be finding or building a library of pre-built objects - particularly office furniture, and perhaps landscape items visible through the windows. In our experience, decent looking and realistic ones are very tedious to build, and require a bit of artistic skill.

Recommended VRML Viewers/Browsers:

VRML code requires a viewer to display it. There are only a few of these available, and we recommend the following ones, with Blaxxun being the preferred one, and GLView as a second choice which is useful to produce a second view/opinion on a file.

Blaxxun Contact 5.1
http://www.blaxxun.com/

This is one of the companies trying to make a business model out of creating software for CVE's of various types. Contact is a VRML browser which runs under Windows IE, and other web browsers. It is a commercial product which has free versions for non-commercial user and has currently active support.

GLView
http://home.snafu.de/hg/

This is the home page for GLView 4.4 - a very nicestandalone VRML browser for W2k. The browser has not been updated since 11/2000

Cosmo
http://ca.com/cosmo/

This is the homepage for the Cosmo browser, It runs on IE and Netscape under several versions of Windows and is considered one of the best. Originally written by SGI, it has migrated to the ownership of Computer Associates. They have kept it available but do not do any maintenance or development on it.

9.2 Survey of Existing 3D Editors

The existing editors can be roughly grouped as follows:

a. Video-game editors

There are lots of these in existence. Most major video game environments provide there own, and 3rd parties have produced compatible ones. Video games almost always use proprietary file formats and these editors are short on VRML import/export capabilities. These editors were not examined in much detail, but if we find one with good VRML support it would be of interest.

b. Professional CAD/CAM editors

These are generally very expensive and not available as packaged software. Their VRML support is spotty. Autodesk is the market leader and is moving toward proprietary file formats. The cost of such software would prevent the average open-source user from using them, so we have been looking elsewhere.

c. Intermediate-level CAD/CAM editors

Most of the editors in this category are commercial, although several have free downloads. Typically, they have several versions and price levels, and the ones which import/export VRML or *.DXF are in the $150-600 price range. The usefulness of their VRML code is unknown. CAD/CAM editors have a reputation for producing high-polygon count mesh-structured files – this is generally not a problem in professional CAD/CAM enviroments, but is undesirable for virtual worlds.

Cycas
http://www.cycas.de/

This is targeted at the architectural and interior design fields. It is very precise and can do top quality rendering. It has a free download, but the price is around $250 for the VRML-capable edition.

Design Workshop Lite
http://www.artifice.com/dw_lite.html

Originally written for the Mac and ported to Win32 where it works very well. It has a very nice 3D UI design. The VRML-capable edition is ~$250 but there is a free download which exports only *.dxf files. The lite edition appears not to have been updated since 1998 .

Octree
http://www.octree.de/ and look for OCTREE-CAD

This is apparantly a product of a professional architectural firm rather than a software house. There are several unix versions and a less-capable windows port. It is free for non-commercial use, but the VRML export feature is not planned for the future and not available currently. This looks very interesting and we will monitor it.

Internet Space Builder
http://www.parallelgraphics.com/products/isb/

This is commercial but has a free download which does export VRML. The free

version is limited to 1400 faces in the output file and that doesn't go very far when several of their basic items of furniture take 100-300 faces. They have an unusual approach to the UI - everything starts with cubes, or other basic geometric figures, and then uses set intersection to cut out pieces and form the model. It works and has a pretty good UI, but looks very tedious.

d. General 3D and VRML editors

Most of these do not seem to be aimed at doing building interiors, but several could do the avatars.

Blender
http://www.blender3D.org/

This is open source, has a large community, and lots of documentation. However, none of the documentation points to architectural interiors as a standard use - instead it seems to be especially good for building and/or animating polygon-mesh objects. It has a widespread reputation for being difficult to learn.

mjbWorld
http://www.euclideanspace.com/

This is a 3D editor which can export VRML files. It appears to be a decent effort, but there appears to be only one programmer supporting it. The home page has several good tutorial pages on 3D issues. It appears to be open source and has a Java version.

Milkshape
Wings3D
White-Dune

e. Retail Home Design Packages

These are available via retail channels and will cost $70-250 to get the versions which can export VRML. IMSI and Broderbund have packages, but from scanning reviews and comments it appears that the that the best of the lot are the ones from www.punchsoftware.com so the following was purchased to get some hands-on experience.

Punch Professional Home Design Suite – Platinum version

This is probably the nicest and most capable consumer-level package if you want to design a new home or to do some home re-modeling. It has many good features for doing architectural interiors, including a very easy to learn and use GUI, a nice collection of furniture, architectural elements such as doors and windows, and extensive textures.

What works:

It is fairly easy to create architectural interiors, color the walls, create transparent

windows, possibly put landscaping outside, and put basic office furniture in place. This is done in a combination of 2-D and 3-D editing, with the 3-D results visible in separate windows. The camera can then walk/fly/orbit the resulting model from both the outside and the inside. There is no collision detection in the punch editor, so the camera can just move through walls, etc.

What doesn't seem to work (yet):
a. Cannot color the ceilings for some unknown reason.
b. It has a very nice set of textures, however they seem to be in a proprietary format and do not export with the VRML. Thus, only flat colors can be used in the exported model. Many objects such as windows and doors are automatically created with textures such as wood-grain. These look nice, but must be converted back to flat colors for export.
c. The office furniture objects are only approximately what we would like to have. Also, they use a lot of textures which have to be converted back to flat color for export.
d. Does not seem to be able to create working lights. Also, there is no way to put fluorescent panels on the ceiling.
The exported VRML models seem too dark in the glView browser.
e. No obvious way to create blackboard/whiteboards on the wall. No classroom seats, if we want them.
f. Some of this may be fixable by fiddling with the VRML in external VRML-oriented text editors.

It uses a proprietary internal file format. Its VRML output does work, but was disappointing on several issues:
1. No use of PROTOS or EXTERNPROTOS – each piece of furniture was modelled in place leading to very high polygon counts and huge file sizes. For example, each office chair in a scene adds 100kb to the file size. A basic layout with 3 rooms, 6 desks, and 9 chairs generated a 2Mb file .
2. Door and window openings were not real openings, but simply a thin black polygon marking the opening.
3. Lighting had to be completely re-worked manually after VRML export.
4. An avatar node had to be added to get decent navigation.
5. Some of the above items can be re-worked by manually editing the *.wrl files, and re-building the furniture as EXTERNPROTO files, but fixing the door/windows looked like a lot of work. Identifying which polygon in a100kb text file matches which polygon on the screen is very tedious.
6. Overall, the VRML output was only marginally satisfactory. However, this editor did an excellent job of demonstrating the pitfalls of VRML and that an editor with a VRML export feature still requires hands-on experience to judge the usefulness of the output in our context.

f. Commercial Virtual World Environments
Octaga
Blaxxun
Int3D

Pharus from www.int3D.com

Having reviewed the pros and cons of many editors listed above in various levels of detail, the best available choice seems to be Pharus. This is a commercial product which is free for non-commercial use. They hope to induce users to build virtual VRML worlds which run from their own web servers. Commercial uses could include architectural and interior design and furniture firms building 3D walkthroughs of their designs which their clients could view on the web. From our standpoint it has the following pros and cons:

Pros:
1. This editor is designed for producing virtual worlds based on typical real-world buildings and interiors.
2. The VRML code produced is very small and is structured to use libraries of objects such as doors, windows, and office furniture by calling them as externprotos.
3. Subjectively, the learning curve for using this editor is quite reasonable compared to its capabilities. On this issue of ease of use in producing buildings/interiors I would scale the editors roughly as:

Group 1 – easiest
Punch
Group 2 -
Pharus
Design Workshop
Cycas
Group 3 and up -
all the rest

where group 3 is not very close to group 2.

4. It should be possible to use local copies of all required externproto objects thus creating virtual world environments which are completely independent of the systems and libraries on the int3D servers.

Cons:

1. It is not fully open source but we can probably still use it. If int3D maintains the free for non-commercial use, our users can use it to build their own modified worlds.

2. The legal conditions which control what can and cannot be done with its library of furniture and architectural objects are not entirely clear.

3. The furniture library is mostly focused on elegant designer furniture for a home

environment. There are a few items that fit an office theme, but others will be needed. This leads to a concern for techniques for using outside libraries.

4. VRML objects from other sources can be incorporated into the 3D worlds which are created. It does not look like this can be done by including them in the Pharus editor menus - it will probably have to be done by manually editing the VRML output code.

5. The file save procedure from the Pharus editor is a bit of a kludge on their part. This is due to the editor itself being written in VRML/javascript and running only in a specific VRML environment. On the other hand, this means the editor code is essentially open for examination and study, although not for modification.

Overall, its not perfect, but it seems workable for the overall project. The combination of free for non-commercial use, a focus on buildings and interiors, pretty easy learning curve, and very good VRML output has not been matched by any other editors which were examined. At a minimum, it has given us confidence that small, efficient VRML files can meet the graphical requirements of the Unicron project. This provides the basis for further work aimed at completing a basic scene and getting usable tools into the hands of the final users.

10 Bibliography

1. “The Annotated VRML 2.0 Reference Manual”,
Rikk Carey and Gavin Bell, Addison-Wesley, 1997,
ISBN 0-201-41974-2 .

2. “The VRML 2.0 Handbook”, Jed Hartman and Josie
Wernecke, Addison-Wesley, 1996,
ISBN 0-201-47944-3 .

3. “The VRML Sourcebook”, Andrea Ames, David Nadeau,and John Moreland, Wiley, 1996,
ISBN 0-471-14159-3 .

4. "The OpenGL SuperBible", Richard S. Wright Jr.and Michael Sweet, Waite Group Press, 1996, ISBN-1-57169-073-5.

5. "Graphics Programming in Icon", Ralph Griswold, Clinton Jeffery, Gregg Townsend, Peer-to-Peer Communications, 1998, ISBN-1-57398-009-9.

6. "Programming with Unicon", Clinton Jeffery, Shamin Mohamed, Ray Pereda, Robert Parlett, draft manuscript, contact jeffery@cs.nmsu.edu

11 Notes and web sites

Much of the research for this paper was done on the web. Here is an annotated bibliography of web resources.

http://www.sics.se/dce/dive/dive.html
SICS is the Swedish Institute of Computer Science which is highly respected. This project sounds very similar to Unicron. The current level of activity is unclear. Do they release their editor?

http://www.web3D.org
http://www.web3D.org/TaskGroups/x3D/faq/
http://www.web3D.org/fs_weblinks.htm
This is a very good starting place for all 3D issues on the web. It has a large and well-organized links page:

http://web3D.about.com/?once=true&
http://virtuality3D.co.uk/main.php?p=home
These are other good starting points.

http://www.faqs.org/faqs/graphics/fileformats-faq/part3/
This is a master listing of information on graphics file formats.

http://www.linuxartist.com/3D.php
This contains a large listing of linux modeling links. Half of the links are dead.

VRML

http://virtuality3D.co.uk/main.php?p=home
This contains a good list of links to vrml software:

http://web3D.vapourtech.com/tutorials/vrml97/index.html
This is an excellent VRML tutorial, especially for beginners.:

http://web3D.about.com/library/weekly/blatozvrml.htm
This is a listing of free VRML models. Several of them are buildings. It is good for an idea of what can be done.

http://www.octaga.com/vrml/chamber.wrl
This is a very impressive VRML demo of a building interior. Unfortunately, it was done in the Autodesk Lightwave editor which is very expensive, and the file is structured as one huge polygon mesh which is intractable for study or editing.

http://www.openvrml.org/

This group is building a cross-platform VRML browser which will include linux. The project is active but the software is labeled alpha.

http://www.gertstein.org/hal/vrmledit.html

This is a listing of VRML editors with brief comments on each. It is a few years old and somewhat outdated.

X3D

http://www.web3D.org/TaskGroups/x3D/translation/README.X3D-Edit.html

This is the main (or only?) X3D editor currently available.

http://web.nps.navy.mil/~brutzman/

Don brutzman is a CS professor who is very active in developing X3D and getting it through the standards process. There are also notes from his class in VRML programming.

Retail Home Design Packages

http://www.b4ubuild.com/links/cadd.shtml

This is a very good overview site for consumer home-design software, and the whole site is devoted to the issues of home building and remodeling.

http://www.consumersearch.com/www/computers/ home_design_software/fullstory.html

This is a good summary of reviews of consumer home-design software. The short story: Broderbund 3-D Home Architect and the packages from www.punchsoftware.com are the only ones worth looking at.

Professional CAD/CAM/Architecture

http://www.tenlinks.com/CAD/PRODUCTS/AEC/design.HTM

This is a good overview site for expensive and proprietary architectural software:

File Formats

The most popular or commonly implemented format for CAD/CAM and architecture is the *.dxf format created by AutoDesk in the early 90's. Version R12 has public documentation. Version R13 which is later, has documentation which is available, but expensive. AutoDesk seems to be tightening its proprietary hold on the next generation of the format. The R12 format is easy for programs to parse, but the difficulty of rendering is unclear at this point. The public documentation for R12 is at:

http://usa.autodesk.com/adsk/servlet/ps/item?siteID=123112&id=2882295&linkID=2475323

3-D File Format Conversion

There does not seem to be much open-source software available in this category, and not much commercial software here either. The $150 version of IMSI Hijaak Pro claims these capabilities - ie. VRML <--> *.dxf, but their website gives very little useful detail on the issues. In particular, it is likely that such conversion has a lot of imperfections - just as moving word-processing documents between editors seldom works perfectly. IMSI does have a free 15-day download available.

Others sites worth looking at:

http://www.coin3D.org/

COIN is not an editor, but a C++ API released under the GNU license. It runs on top of OpenGL and appears to provide the next higher level of graphics operations to make using OpenGL easier. The site links to some interesting applications which have been built using the COIN library.

http://ftp.arl.army.mil/brlcad/

BRL-CAD is a free constructive solid-modeling package which was developed by the US army for use in its own projects. It is in source-code only and its image gallery does not include any architectural interiors.

http://www.ac3D.org/pages/gallery

This is a 3D editor for unix, linux, and windows It has a free trial and costs $50. It exports VRML, *.dxf and other formats. It is questionable for doing architectural interiors.

http://moonlight3D.net/faq.php?sid=70babde64cc174d5917f6fd2f2c2a467

Moonlight3D is a 3D editor with some respect in the open source community. It exports VRML, runs on linux only, and is supposed to have an easier interface than blender. Its architectural capabilities are unclear. Its project stability is questionable – it started out as open source, went to closed source, and died. New developers took over the last open release and are working to improve it.

http://ayam.sourceforge.net/

This is another free 3D modeling environment for the RenderMan interface and format. It supports linux and Win32. There were no architectural samples among its demos. From reading the faq, it may be very messy to install. The lead developer is a student and there may be some dependencies on his university system.