Croquet – A Collaboration System Architecture

David A. Smith
104 So. Tamilynn Cr.
Cary NC, 27513
davidasmith@
bellsouth.net

Alan Kay [1]
1209 Grand Central Ave
Glendale, CA 91201
alan.kay@
viewpointsresearch.org

Andreas Raab
University of
Magdeburg, Germany
andreas.raab@
squeakland.org

David P. Reed
MIT Media Laboratory
20 Ames Street
Room E15-492
Cambridge, MA 02139
dpreed@reed.com

ABSTRACT

Croquet [18] is a computer software architecture built from the ground up with a focus on deep collaboration between teams of users. It is a totally open, totally free, highly portable extension to the Squeak [5] programming system. Croquet is a complete development and delivery platform for doing real collaborative work. There is no distinction between the user environment and the development environment.

Croquet is focused on interactions inside of a 3D shared space that is used for context based collaboration, where each user can see all of the others and what their current focus is. This allows for an extremely compelling shared experience. A new collaboration architecture/protocol called TeaTime has been developed to enable this functionality. The rendering architecture is built on top of OpenGL [13].

KEYWORDS

Croquet, collaboration, User Interface, 3D graphics, Squeak, Smalltalk, TeaTime, OpenGL, peer-to-peer.

INTRODUCTION

Croquet was built to answer a simple question. If we were to create a new operating system and user interface knowing what we know today, how far could we go? What kinds of decisions would we make that we might have been unable to even consider 20 or 30 years ago, when the current operating systems were first created?

The landscape of possibilities has evolved tremendously in the last few years. Without a doubt, we can consider Moore's law and the Internet as the two primary forces that are colliding like tectonic plates to create an enormous mountain range of possibilities. Since every existing OS was created when the world around it was still quite flat, they were not designed to truly take advantage of the heights that we are now able to scale.

What is perhaps most remarkable about this particular question is that in answering it, we find that we are revisiting much of the work that was done in the early sixties and seventies that ultimately led to the current set of popular system architectures. One could say that in reality, this question was asked long ago, and the strength of the answer has successfully carried us for a quarter century. On the other hand, the current environments are really just the thin veneer over what even long ago were seriously outmoded approaches to development and design. Many of the really good fundamental ideas that people had were left on the cutting room floor.

That isn't to say that they thought of everything either. A great deal has happened in the last few decades that allows for some fundamentally new approaches that could not have been considered at the time.

We are making a number of assumptions:

- Hardware is fast – really fast, but other than for booting Windows or playing Quake no one cares – nor can they really use it. We want to take advantage of this power curve to enable a richer experience.
- 3D Graphics hardware is really, really fast and getting much faster. This is great for games, but we would like to unlock the potential of this technology to enhance the entire user experience.
- Late bound languages have experienced a renaissance in both functionality and performance. Extreme late-bound systems like LISP and Smalltalk have often been criticized as being too slow for many applications, especially those with stringent real-time demands. This is simply no longer the case, and as Croquet demonstrates, world-class performance is quite achievable on these platforms.
- Communication has become a central part of the computing experience, but it is still done through the narrowest of pipes, via email or letting someone know that they have just been converted into chunks in Quake. We want to create a true collaboration environment, where the computer is not just a world unto itself, but a meeting place for many people where ideas can be expressed, explored, and transferred.
- Code is just another media type, and should be just as portable between systems. Late binding and component architectures allow for a valuable encapsulation of behaviors that can be dynamically shared and exchanged.
- The system should act as a virtual machine on top of any platform. We are not creating just another application that runs on top of Windows or the

[1] Sr. Fellow Hewlett Packard and President, Viewpoints Research Institute, Inc.

Macintosh – we are creating a Croquet Machine that is highly portable and happens to run bit-identical on Windows, Macintosh, Linux, and ultimately on its own hardware… anywhere we have a CPU and a graphics processor. Once the virtual machine has been ported, everything else follows; even the bugs are the same. Most attempts at true multiplatform systems have turned out to be dangerous approximations (cf. Java) rather than the bit-identical "mathematically guaranteed" ports that are required.

- There are no boundaries in the system. We are creating an environment where anything can be created; everything can be modified, all while still inside the 3D world. There is no separate development environment, no user environment. It is all the same thing. We can even change and author the worlds in collaboration with others *inside them while they are operating* .

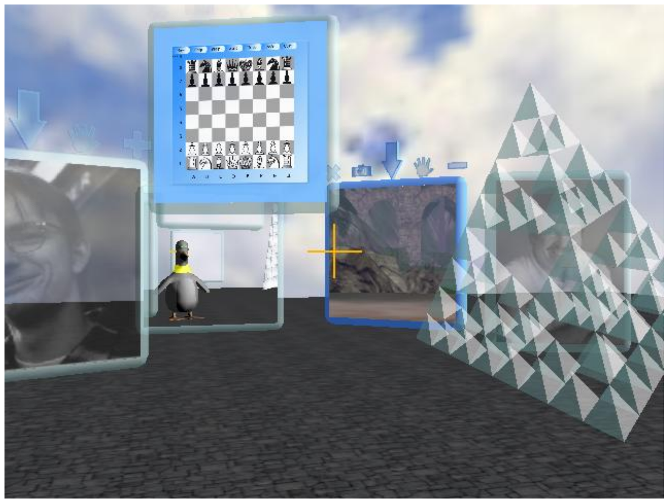
Figure 1. Croquet multi-user environment.

Croquet Is…

Croquet is a computer software architecture built from the ground up with a focus on deep collaboration between teams of users.

Croquet is a totally ad hoc multi-user network. It mirrors the current incarnation of the World Wide Web in many ways, in that any user has the ability to create and modify a "home world" and create links to any other such world. But in addition, any user, or group of users (assuming appropriate sharing privileges), can visit and work inside any other world on the net. Just as the World Wide Web has links between the web pages, Croquet allows fully dynamic connections between worlds via spatial portals. The key differences are that Croquet is a fully dynamic environment, everything is a collaborative object, and Croquet is fully modifiable at all times.

The current computer user paradigm is based upon a completely closed individually focused system. The user has a very low-bandwidth communication capability via e-mail, or instant messaging, but outside of some very simplistic document sharing capabilities, the user is quite alone on his desktop.

Croquet has been focused on high bandwidth collaboration from its inception. Simply put, the fundamental building block of the Croquet architecture is a system that makes every single object in the system collaborative.

Croquet's collaboration architecture is based upon the concept of replicated versioned objects coordinated by a universal timebase embedded in the communications protocol. This part of the architecture is referred to as TeaTime.

One way to think of the Croquet environment is as a high bandwidth conference phone call. Once a connection is made, the user not only has voice communication with the other participants, he also has the ability to exchange documents, collaboratively design systems, perform complex simulations, develop complex project plans, and manage complex projects.

Croquet utilizes OpenGL as the basis of its rendering /component framework called TeaPot. The architecture utilizes a semi-retained model, such that it uses a rendering hierarchy based upon dynamically composable objects, but each of these objects has full access to the OpenGL libraries and can extend the capabilities of the rendering engine in virtually unlimited ways.

Squeak is…

Croquet is built on top of Squeak [5], a modern variant of Smalltalk, hence it is a pure object oriented based system. This allows for significant flexibility for the design and the nature of the protocols and architectures that have been developed.

Squeak is a 21st century dynamic-object wide-spectrum operating and authoring environment derived from the 1970s Xerox PARC Smalltalk [4] system in which overlapping window GUIs, Desk Top Publishing, media authoring, and many other familiar software systems were first developed. Several of the authors of Squeak were principals at Xerox and were co-creators of many of the PARC inventions.

An essential part of our development process is Squeak's ability to keep the system running while testing and especially while making changes. Squeak allows even major changes to be performed incrementally and they take no more than a fraction of a second to effect. Another key feature of Squeak is its generalized storage allocator and garbage collector that is not only efficient in real-time (so that animations and dynamic media of many kinds can be played while the gc is collecting), but that allows reshaping of objects to be done safely.

Related Work

There are a number of seminal efforts over many years to which Croquet owes a great deal. Many of these early efforts were the first building blocks of the current popular windowing computer interface and usability paradigms. What is particularly interesting is that the focus of the Croquet project tends to be on the parts of these early efforts that were not picked up by what has become mainstream computing.

Sutherland's work [20] on direct manipulation and modeling of graphical object based entities clearly established the first true fundamental steps toward an interactive human computer user experience. Not only did he establish a great deal of the fundamental methods for how to create and manipulate interactive environments that are still quite relevant, but his focus was on creating a tool that would fundamentally amplify human capabilities.

His further work on the "Ultimate Display"[21] – the first immersive 3D experience, demonstrated the potential, still unrealized, of 3D interactive environments as the basis of a complete user experience to display and interact with computer data, creating "a looking glass" into what he described as a "mathematical wonderland." His vision of the system would represent data in 3-dimensional form, allowing the construction of entirely believable 3-dimensional, computer controlled, virtual worlds. He went much further than this in his description of the potential. "The ultimate display" he wrote, " would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked."

The efforts at Xerox PARC under the leadership of Alan Kay that drove the development of both pure object oriented development environments in the form of Smalltalk and powerful bit-mapped display based user interfaces was key [7]. In some ways, all we are doing here is extending this model to 3D and adding a new robust object collaboration model.

Douglas Engelbart's videoed first demonstration in 1968 of everything from a mouse to hypertext, object addressing and dynamic file linking, and especially shared-screen collaboration involving two persons at different sites communicating over a network with audio and video interface has been a major inspiration to this project. It is telling that this level of rich, deep collaboration between widely separated physical environments has still not been properly achieved. [2]

The Croquet component model architecture is similar to the OpenDoc system developed by Apple [1] and the Squeak 2D Morphic architecture developed by John Maloney [8] Both of these were designed around composable 2D objects – or components. The main ideas behind these systems are that the majority of the environment interactions that the components have to deal with are already available in the base classes that make up the system. The programmer's task is simply to override the behaviors of the objects – how they render themselves, and how they respond to "stimuli" from their surrounding components and from the user. Then the programmer and ultimate user can "compose" these intelligent blocks to form a useful document or application.

Smith's work on ICE – the Interactive Collaboration Environment, a multi-user shared component environment and later the Virtus OpenSpace architecture [17] acted as an important guide to the resulting Croquet system and in a sense Croquet is a far more complete result of this work.

Fisher et al [3] developed a powerful, totally immersive 3D working environment. This system included the ability to dynamically interact with the system via 3D menus and window documents, and the ability of the user to directly manipulate his position and orientation inside the world and interact with the objects that inhabited it. Further, the system could interact with the user as if he were just another object inhabiting the space. The best example of this was the virtual escalator that the user could step on that would then carry him up to another floor.

The TeaTime time based collaboration protocol/architecture is directly based upon early work of Reed [14, 15]. Jefferson's work [6] is related, but does not include the idea of maintaining a partial history, managing replicated objects, or incorporating two-phase commit. Miller and Dennis's Timewarp [9] protocol extends Jefferson's work to support "multiple versions", a central concept in TeaTime. Mirtich [11] employed the timewarp model to implement a graphics simulation architecture for maintaining complex physical modeled state.

CROQUET ARCHITECTURE

Like any complex system, it is impossible to account for the architecture of Croquet without describing its various pieces and the interrelationships they share with each other.

Croquet has been designed to operate as a peer-to-peer architecture. This ensures the greatest level of flexibility in the design of the system and its ultimate usability. All objects are symmetric in their ability to act both as local and remote recipients of the same messages. There is no intermediate step to be performed to interpret and rebroadcast these messages. A central server can be used to establish the initial connections.

The peer-to-peer architecture is also in keeping with the philosophical roots of Croquet, which is to act as a broad-band phone call (or better, as a conference call). Though a central server may play some role in a phone connection, its role is strictly limited to redirecting the pertinent information with no changes. Any messages sent from a

Croquet user will arrive and be used by all other users without intermediate interpretation. This has the additional benefit of dramatically simplifying secure data transfer.

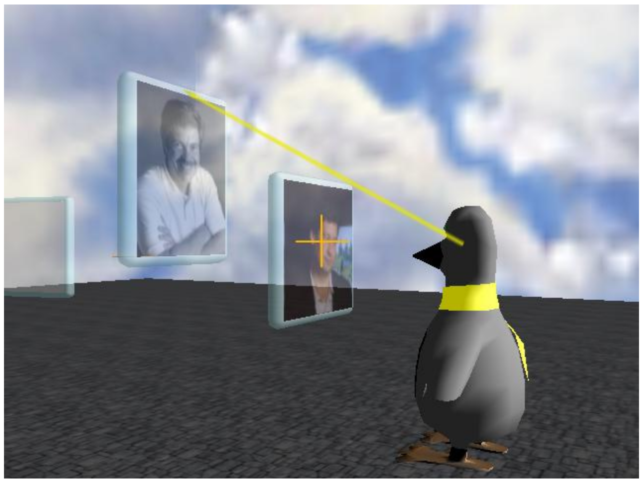

Figure 2: The other user is dragging a window up into the air.

A remote user can do anything in this peer-to-peer shared world that you can. The environment acts as a single shared place that every inhabitant can manipulate and modify. You have a context to see and understand where the other user's interest is and further, what it is they are doing.

The key part of the architecture making up Croquet enabling this rich level of peer-to-peer interaction is TeaTime, which is the basis for component object-object communication and world/object synchronization, including initial content synchronization.

The Croquet component architecture is an extension to the base TeaTime class structure to support user/system interactions with the object and the objects ability to perform internal TeaTime based simulations and external rendering.

A user interface has been developed that allows the user to create and manipulate objects inside the virtual world as well as easily traverse it. It is important to note that even the user interface is more a property of the object that represents the user in the space, and that this can easily be modified to support fundamentally different approaches.

The Teapot graphics engine is based upon OpenGL. The system is built around the ideas of a retained graphical engine pipeline, but with the developer retaining the ability to make direct calls to the OpenGL library at any of the nodes.

A scripting language has been developed that is focused on extremely high-level control of objects that make up the Croquet environment. Though this scripting language is ideal for relatively naïve users of the system, it is intended to be used for virtually all high level control and interaction. It is designed to be implicitly collaborative, so that the users need not even be aware of the fact that all scripted actions result in synchronous responses across the peer-to-peer network.

TeaTime: A SCALABLE REAL-TIME MULTI-USER ARCHITECTURE

TeaTime is the basis for object-object communication and synchronization. It is designed to support multi-user applications that can be scaled to huge numbers of users in a common space, concurrently interacting. The most directly visible part of this architecture is the TObject class which is used to define and construct subclassed Tea objects. A Tea object acts with the property that messages sent to it are redirected to replicated copies of itself on other users participating machines in the peer-to-peer network. All of the interesting objects inside of Croquet are constructed out of subclasses of TObject. This messaging protocol supports a coordinated "distributed two-phase commit" that is used to control the progression of computations at the participating user sites.

In one way, the protocols we have developed are simply an extension of the message passing model employed by Squeak. This is really a meta-protocol, as any message may be dynamically redirected to the entire group of users while maintaining the appropriate deadline based scheduling.

TeaTime is designed to allow for a great deal of adaptability and resilience, and works on a heterogeneous set of resources. Rather than develop highly specific, optimized algorithms, TeaTime is a framework of abstraction that works over a range of implementations that can be evolved and tuned over time, both within an application, and across applications. This approach is based on the work of Reed [14, 15] since the mid 70s.

Elements of approach

- A coordinated universal timebase embedded in communications protocol.
- Replicated, versioned objects – unifying replicated computation and distribution of results.
- Replication strategies – that separate the mechanisms of replication from the behavioral semantics of objects.
- Deadline-based scheduling extended with failure and nesting.
- A coordinated "distributed two-phase commit" that is used to control the progression of computations at multiple sites, to provide resilience, deterministic results, and adaptation to available resources. Uses distributed sets.
- Time-synchronized I/O. Input and output to real-time devices are tied to coordinated universal time.

Synchronous vs. asynchronous computing?

The key issue is that we want to be able to provide a continuous experience that coordinates users at multiple locations, interacting in a tightly collaborative way. It isn't unreasonable to expect that all users can see the effect of actions at other sites within 10's of milliseconds.

Consequently, the approach we propose is an architecture that is synchronous to the degree that I/O is synchronized, but at the same time allows for adaptation of computational strategies.

The key idea for I/O coordination is that input and output events (to interactive devices) are synchronized with global universal time, which is coordinated among all sites involved in a computation.

At the same time, objects behave like processes that exist in time, and each object's behavior is implemented by methods that explicitly manage the temporal evolution of the object. In a sense, object internal states are maintained as ordered histories, and operations are performed at "pseudo-time" instants that are properly ordered with respect to I/O operations whose data connect with the objects.

Device I/O is temporally ordered as well. I/O events exist in real time, and provide the coordination between real time and "pseudo-time" that is necessary and sufficient to achieve the proper user interface behavior. This provides an adaptive approach to real time programming that is not limited to "real time programming".

A Perspective On This Approach

The standard view of a networked virtual environment implementation describes the system as a set of state variables that represent instantaneous system state. [16] Temporal changes are reflected as a sequence of updates to elements of state, and communications distributes the updated state values. This essentially decouples processing from "static" state - that is state that does not change without operation by an external processor that reads and updates it. The model separates processing from storage, and treats consistency as a property of the stored state. Displayed information is then derived from a snapshot of the stored state.

Our view takes Alan Kay's original idea [7] of objects as entities that have behaviors, where messages affect the behavior (state variables are invisible outside the object, and equivalent behavior has meaning independent of how, or even whether, state is represented in any particular way). This allows us to think of self-contained objects that have dynamic behavior even when not driven by external processors. In essence, objects exist in both space and time. Croquet objects interact by exchanging messages. The Croquet view of objects easily incorporates I/O devices, and even real-world objects outside the system, as first class objects in a natural way, whereas modeling objects as abstractions of storage only cannot represent such things as normal objects.

In Croquet, computational time and real time are loosely coupled. The code that executes the dynamic behavior of objects typically can execute a lot faster than the real-time behavior represented, so an object can carry out many seconds worth of behavior per second, if left to itself. The Croquet system's job is to coordinate the execution of objects so that all behaviors that can have a visible effect are completed in time to communicate those effects through the system interfaces.

Since this is the only constraint, objects in the Croquet environment are free to implement a wide variety of strategies for computing their behaviors. This kind of object-specific strategy dramatically reduces the need for lock-step coordination among distributed concurrent activities. Because they maintain some element of past history in the object representations, this kind of approach requires additional storage overhead per object. But the benefit of dramatically better scalability and reduced latency far outweigh the cost of extra storage.

The other key idea in TeaTime is our approach to resilience and fault tolerance. Most large scale distributed virtual environments are quite difficult to handle because at any point in time some elements may become disconnected and other elements may be dynamically added. We recognize this issue in the Croquet object model - each object is responsible for maintaining sufficient information to recover from system disruptions. The key idea in TeaTime is that the state of objects evolves through a distributed *two-phase commit* protocol. Behaviors of all objects that influence each other are first computed, contingent on completion of all dependent object behaviors, and then those behaviors are atomically *committed*. If the behaviors are not completed in time, all contingent calculations are undone by the individual objects.

The principle of giving an object responsibility for its own behavior allows for a wide variety of strategies for individual objects to implement the proper resilience and recovery. In a networked virtual environment, these strategies can include dynamically adaptive behavior that can cope with heterogeneous hardware, wide variations of delay, and so forth. Applications programmers can tune applications to use new strategies that derive from the unique requirements of their application objects, or use packaged libraries that embed those strategies in abstract object classes that can be specialized for specific implementation

COMPONENTS

We use the term "component" to describe the basic unit of composition in the Croquet 3D environment The TeaPot suite of component level classes are built on top of the TObject base class. The base class of these components is TFrame. The subclasses of TFrame act as frames in an OpenGL rendering hierarchy, as event handlers, and as time based simulation objects as described above as part of TeaTime.

Rendering Engine
The philosophy behind Croquet's TeaPot rendering engine is based on allowing the programmer complete access and control of the underlying graphics library, in this case OpenGL, while at the same time, providing a rich framework within which to embed these extensions with a minimal level of effort. This allows the naïve graphics programmer and 3D artist a way to easily create interesting graphic artifacts with minimal effort, yet allows the expert the ability to add fundamental extensions to the system without the need to modify the underlying architecture.

A rendering frame includes a transform matrix which defines the orientation and position of the object in a 3D space relative to its parent object in the hierarchy as well as the ability to render itself in that position in global space. A rendering message is sent to the object when its position in the hierarchy is reached. The object then calls the appropriate OpenGL library functions to render the object.

Event Manager
An event handler can respond to user events such as keyboard and mouse/pointer events. Again, this interface is quite extensible by the programmer, but the default is that the TCamera carries a TPointer object which tracks the objects that are underneath the current mouse position. A TPointer is a 3D analog to the mouse event object. Instead of being just a 2D position on the screen, the pointer includes vector information, in this case from the camera to the selected object in both global and local (to the selected object) frame transforms.

Keyboard events are also forwarded to the currently selected object. This model allows us to embed 2D objects into a scene, where the containing 3D object simply converts the TPointer vector data back into a 2D mouse position on the surface of the 2D object.

CROQUET SCRIPT
Our goal in developing a scripting language is to provide ways to dynamically adjust the complexity that's exposed to the user. E.g., we know we are going to have a variety of users, starting from kids over graphics designers up to hardcore hackers. All of them need different levels of accessibility and the key question is how we can give them access at the level they can deal with.

The way we are addressing this problem is by conceptually "slicing" the system along various boundaries. The first "slice" is what we find in the TFrame hierarchy - a user at this level has access to the "guts of the system" being able to manipulate Croquet objects at a very low level. The second one is a side-ways "slice" which effectively encapsulates the core notions of the framework and exposes a simplified interface to scripting users. Most of what can be done here in terms of "modifying the framework" happens by parameterization; the intrinsic behavior of frames (such as rendering or picking) cannot be touched from here. Since hardly any user of the system will ever have the need to do this, this seems to be the place where most scripting activities are going to happen. The third "slice" is one that provides an even more simplified view on objects with a different user interface for scripting (tile-like interfaces such as that found in Squeak's eToys). Here, we aim for supporting users who start learning the system by providing them with the essential vocabulary for manipulating aspects and the core properties of these objects.

By providing these different ways to access the system we are able to define learning curves in terms of what users have to know when and where. As we grow in our explorations of the system we are able to teach users more of the aspects that make it tick. We are aware that many users will not go "deeply enough" into the system to (say) manipulate the core framework notions, yet if there is a need to do this, they can. The scripting system provides a learning curve with "intermediate plateaus" along the path.

USER INTERFACE
A key part of the Croquet architecture is that the user interface is just another collection of objects that can easily be replaced or enhanced. The way the user controls his position in space and how he manipulates objects inside of it is controlled by a camera object which is easily replaced.

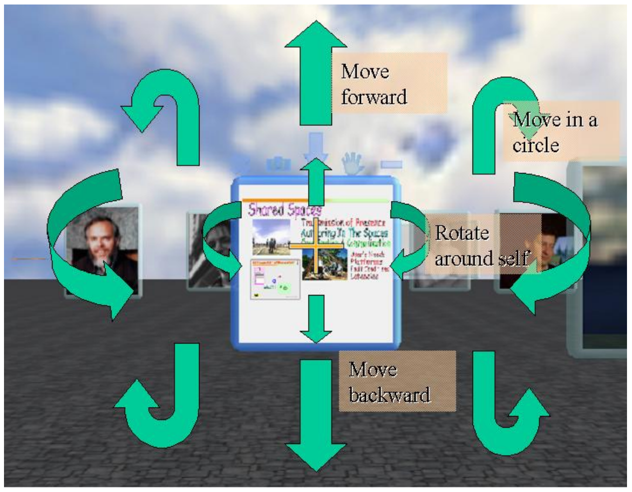

Figure 3: The user moves forward and back by moving the mouse up or down, and rotates by either moving the mouse to the left or right.

In the current system, to move around inside the world, just click and hold the right mouse button. Where you click relative to the cross hair determines how you move. The closer you are to the cross hair, the slower you will move. To move forward, move the cursor on top of the cross hair and click. The distance from the center determines your forward velocity. Click and hold just above the cross hair and you will move forward very slowly. Click far from the cross hair and you will move quickly.

If you move the mouse underneath the cross hair, you will move backwards. Moving it right rotates you to the right. Again, distance determines velocity – in this case, angular velocity, or the speed at which you rotate. If you are directly over or under the cross hair you will move in a straight line with no rotation. If you move directly to the left or right of the cross hair, you will rotate around your center without any forward or backward motion. If you put the cursor in just above and to the right of the cross hair you will move forward a bit and rotate to the right a bit – all at the same time. This allows you to walk in a circle.

Objects can be selected and manipulated just as they are in 2D space. Simply click and drag with the left mouse button. A 3D window can be dragged, resized, and even rotated – depending upon where on the window's frame you select. Additional controls are available through 3D buttons on top of the window. Simply click and release as you do in a 2D environment.

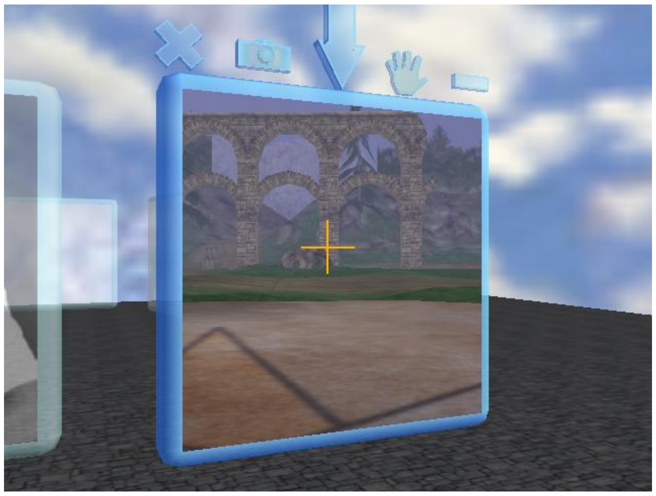
Figure 4: An open portal. We can see into the linked Space, in this case the entrance to a multiplayer game.

SPACES and PORTALS

Simply put, a Space is a place. In Croquet, a space is a container of objects, including often the user. A good example of a space might be a child's play room. All of his toys are objects that happen to be lying on the floor, or perhaps put away. A child can always come into the room to play, or even pick up a toy and carry it outside. In Croquet, Spaces can act like rooms, but they can also act as landscapes, or virtual conference rooms, or any kind of 3D container of any size.

Portals are simply a 3D spatial connection between spaces. If you place one portal in one space, and a second portal in a second space and link them, then you can view from one space into the other. In the example of the child's room, a portal is simply the door to the hallway. The hallway is just another space. One key difference between Croquet portals and spaces and the real world of course is the concept of actual versus virtual location. In the real world, the hallway must be physically next to the child's play room, or the door simply won't go anywhere – at least it won't lead into the hall. In the virtual world, a portal can connect ANY two spaces, even if one is located on a computer half a world away. Physical location doesn't mean anything. Connections are all virtual. Consider as an example, the mirror. In Croquet, a mirror is actually a portal that happens to be linked back to itself. In other words, it is actually a door that happens to open into the room it is leaving from.

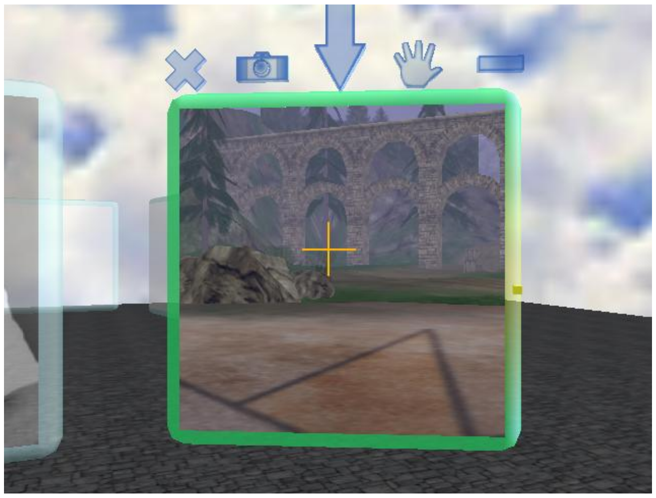
Figure 5: Here the portal has been rotated toward the user. Just like a mirror, we get a slightly different view into the game world by rotating its "container".

One of the key aspects for Croquet is the ability to have a portal dynamically move around in a space, while allowing the proper view through the portal. This is a bit strange, but it works like this: when you look through a window, what you see is determined partially by your position relative to the window. If you move to your left, you can see more of the space to your right (and vice versa). But, if you could pick up the window and move it relative to your position - instead of you moving relative to its position, the exact same thing should happen. It should be much like picking up a box and looking through a hole in it. You turn the box around to see different areas.

The big win for portals is that they allow the user to jump from one virtual space to another by simply walking through the portal, just as the child walked through the door from his play room. What is different in Croquet is that the portal can lead to anywhere in the virtual world. In turn, portals that are contained within these spaces can themselves lead to other worlds. This essentially replicates the workings of the World Wide Web as these portal "links" can point to any other connected machine on the net.

FURTHER WORK

Croquet will include both a robust IP telephony capability, in keeping with our philosophy of the system acting as a broad band telephone call, and an instant messaging system that will act as the dial-tone and ring tone – where other users can indicate their interest in initiating a collaboration session.

A name space and security model is being developed. Our early study seems to indicate that a capabilities model similar to that used by the "E" [10] language is the proper course.

Preliminary work has been done on a multi-dimensional matrix package. The focus of this work is to develop a very high-performance mathematical package for use in simulation and rendering.

Sound is a crucial part of any immersive environment. We are studying the developing industry standards for 3D sound. In particular, the work in OpenAL [12] seems to be promising.

CONCLUSION

Croquet has been designed from the ground up with a focus on enabling large scale peer-to-peer collaboration inside of a compelling shared 3D environment. A number of new technologies have been developed to support the robust deployment and development of this system including the TeaTime communication/collaboration architecture/ protocol, TeaPot, a semi-retained graphics engine based upon OpenGL, a powerful space and portal paradigm which mirrors web pages and links on the web, and a scripting engine that enables relatively naïve users to develop powerful multi-user applications using Croquet.

ACKNOWLEDGMENTS

We thank Kim Rose, Michael Rueger, Bill Cole, Michael Moody, and of course the Squeak Central team of Dan Ingalls, Ted Kaehler, Scott Wallace, Andreas Raab and John Maloney. The Croquet project was generously supported by Hewlett Packard Corporation, Chris Cole, and Applied Minds, Inc.

REFERENCES

1. Apple Computer, Inc. OpenDoc Programmer's Guide for the Mac OS. Addison Wesley, 1995.
2. Engelbart, Douglas, 1968 NLS Demonstration Video: http://sloan.stanford.edu/mousesite/1968Demo.html.
3. Fisher, Scott S., James Humphries, Michael McGreevy, Warren Robinett, "The Virtual Environment Display System," *1986 ACM Workshop on Interactive 3D Graphics.*
4. Goldberg, Adele, David Robson, *Smalltalk-80: the language and its implementation*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1983
5. Ingalls, Dan, Ted Kaehler, John Maloney, Scott Wallace, Alan Kay. Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself, in *Squeak: OpenPersonal Computing and Multimedia* ed. Guzdial, Mark and Kim Rose. Prentice Hall, 2002
6. Jefferson, David R. "Virtual Time". ACM Transactions on Programming Languages and Systems, 7(3):404-425, July 1985.
7. Kay, Alan. "The Early History of SmallTalk", in Bergin, Jr., T.J., and R.G. Gibson. *History of Programming Languages - II*, ACM Press, New York NY, and Addison-Wesley Publ. Co., Reading MA 1996, pp. 511-578,
8. Maloney, John. "An Introduction to Morphic: The Squeak User Interface Framework", in *Squeak: OpenPersonal Computing and Multimedia* ed. Guzdial, Mark and Kim Rose. Prentice Hall, 2002.
9. Miller, John A. and Aideen Dennis, "Hybrid Time Warp (HYT): A Protocol for Parallel Database Transaction Management," UGA-CS Technical Report, University of Georgia (December 1996) 42 pages.
10. Miller, Mark and Marc Stiegler. "E" Website: http://www.erights.org.
11. Mirtich, B., Timewarp "Rigid Body Simulation", *ACM SIGGRAPH 2000 Proceedings*, pps 193-200, July 2000.
12. OpenAL Web Site: http://www.openal.org/home/.
13. OpenGL Architecture Review Board, Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, *OpenGL Programming Guide*, Third Edition. Addison-Wesley. 1999.
14. Reed, David Patrick, *Naming and Synchronization in a Decentralized Computer System*, Ph.D. thesis, M. I. T. Dep't of EE & CS, September 15, 1978. Available as MIT Laboratory Computer Science Technical Report MIT/LCS/TR-205.
15. Reed, David P., "Implementing atomic actions on decentralized data", *ACM Transactions on Computer Systems* 1, 1 (February 1983), pp. 3-23.
16. Singhal, Sandeep, Michael Zyda. Networked Virtual Environments: Design and Implementation. Addison Wesley, 1999.
17. Smith, David A. *OpenSpace: A Small System Architecture.* Virtus Corporation. Unpublished Document. 1994.
18. Smith, David A., Andreas Raab, David P. Reed, Alan Kay. *Croquet User Manual* v.0.01 Web site: http://www.opencroquet.org..
19. Squeak Web Site: http://www.squeak.org/
20. Sutherland, I. *Sketchpad: A Man-Machine Graphical Communications System*, MIT PhD Thesis, 1963.
21. Sutherland, I. "The Ultimate Display"., *Proceedings of IFIPS Congress 1965*, New York, New York, May 1965, Vol. 2, pp. 506-508.