Filters and Tasks in Croquet

David Smith
Croquet Project
104 So. Tamilynn Cr.
Cary NC, 27513 USA
davidasmith@bellsouth.net

Andreas Raab
Hewlett Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA 94304 USA
andreas.raab@hp.com

Yoshiki Ohshima
Twin Sun, Inc.
360 N. Sepulveda Blvd. S. 1040
El Segundo, CA 90245 USA
yoshiki@squeakland.org

David P. Reed
HP Fellow
Hewlett Packard Laboratories
One Cambridge Center, 12th floor
Cambridge, MA 02139 USA
dpreed@reed.com

Alan Kay
HP Senior Fellow
Hewlett Packard
1209 Grand Central Ave.
Glendale, CA 91201 USA
alan.kay@hp.com

Abstract

Croquet [11,12] is a collaborative 3D platform that allows users to work together to create and share ideas. From the beginning we have worked to ensure that the Croquet interface remain as modeless as possible. This allows the user to be most productive with the fewest errors. This is even more important in a collaborative 3D environment. The modeless nature of Croquet has allowed us a great deal of flexibility in how the user is able to both move around the environment while easily manipulating it. Certain kinds of applications, however, require some degree of intelligent pseudo-modal behavior. An example is using a CAD system to create new objects. This process forces the user into an object-creation/modification "mode" that can take control of the interface for a short duration. E.g. we might be in the "line drawing" mode. Clearly this is not a problem, but we also need to ensure that we do not get trapped by the CAD application itself. In a sense, it should have the same degree of "mode" as drawing the line. Our approach to ensuring that Croquet remains modeless is to utilize filter portals that modify both the views of the data in the 3D space and the actions that the user makes through these filter portals.

We are developing an architecture that incorporates the ideas of filters and controls for 3D to solve this problem. Our model uses the Croquet 2D portals [11] as view filters that can modify the nature of the content displayed on the other side of the filter. It also uses these view portals to act to translate the users actions and maintain the editing mode. This allows "through the 3D window" editing of shared content.

Another problem is that this collaborative sharing of interfaces tends to be complex. A new approach to this, an extension of the model-view-controller approach pioneered in Smalltalk [3,4,8], is described. This extension of MVC to collaborative 3D user interface design consists of interactors, tasks, and replicants. This architecture solves the collaboration UI problem in a way that makes it quite easy for the designer to create robust multi-user applications without having to manage the divergent states and goals of each user. The programmer can focus on the design of an extensible system as if he is dealing with a single user.

1. Introduction

As we begin to develop larger scale collaborative applications in Croquet, we find that traditional approaches to design, especially of the user interface, often do not scale to the needs of collaborative 3D environments. The two areas of particular interest are that Croquet has been designed from the start as a modeless environment, which we wish to maintain, and all actions are performed in a shared environment, which means that the interface is also designed to be shared to a large extent.

As an example and test bed for how we see these applications working, we are developing a collaborative CAD system in Croquet called "Wicket"[14] that is powerful, but extremely easy to use. This CAD system is the first major application in Croquet in that it has a significant degree of functionality, but is designed to work in a relatively modeless way. The Wicket interface is designed around a 2D portal acting as both a visual filter that can enhance and modify the image behind it, and can act as an action translator that both maintains the ongoing state of the user actions and can modify the meaning of the pointer as the user moves it through the portal. This translation of the users actions, and maintaining these actions is called a task.

Another aspect of this interface is that though these editing portals may share identical capabilities, they do not necessarily share the same local states. Each user may be focused on a different aspect of the design process. Their own states may not be replicated in the shared worlds, though their resulting tasks and the replicated target objects certainly are. This allows the user interface designer to focus on developing a rich capability for a single user, but still allowing this same system to be simultaneously used by multiple users.

For the user, this "through the window" editing capability gives him an easily accessible semi-modal interface that is easily manipulated and positioned, that is easy to utilize in a semi-modeless way, and is easy to disengage from and remove.

For the developer, this editing capability easily extends the functionality of the shared environment without having to add these capabilities to the environment itself. It also allows the developer to focus on designing for a single user and makes it quite easy to dynamically extend the capabilities of the system.

Another simple example of this kind of behavior is the 3D window in Croquet [11]. The buttons and frame of the window respond when the user drags his pointer over them with the button up, but this action is not replicated. Only the local user sees this change in how the window looks. However, any significant action such as dragging, resizing, or rotating the window are replicated. What is more, these actions are orthogonal, in that any number of users can modify the state of the window in this way without any problem. This means that the designer of the window system need not be concerned about managing multiple users. This is done automatically simply by the nature of the design. In this case, the tasks are simply messages.

In a sense, the window was an accidental success in this regard. The early architecture of Croquet allowed us to experiment with asymmetrical systems like this, which in turn lead to the realization that there was a need to formalize this approach as a more robust collaboration framework becomes available. This formalization is called an interactor-task-replicant architecture. It is somewhat similar to the model-view-controller architecture used in Smalltalk.

2. Related Work

Perhaps Engelbart showed one of the first demonstrations of windowed filtering of data in 1968[2]. More generalized versions of this idea appeared in the work on Sutherland's head mounted display [16] and the Flex Machine. [5]

A more formalized description of filters and task translators applied to 2D editing was developed at Xerox PARC by Bier et.al. [1] They named these models of interaction Toolglasses and Magic Lenses. The Toolglass widget is an interface tool that appears as a transparent sheet of glass over the users work area. It translates the users actions when he manipulates his cursor over it, essentially turning the cursor into a dynamic tool that is applied to the content below the Toolglass. Magic Lenses are visual filters that can change the presentation of the object to reveal hidden information, or enhance or modify the underlying object. A virtual magnifying glass is an example of this kind of interface. The actual magnified image is an example of a Magic Lens, while the ability to properly interact with the magnified object with the appropriate scaling of the cursor interactions is an example of a Toolglass.

This work on Magic Lenses was extended into 3D by Viega et al.[15]. They modified the idea to include bounding volumes as well as flat planes. These bounding boxes are used to render the information contained inside them in a different way, just as the image rendered through the Magic Lens can be different. Stoakley et al's [14] "world in miniature" are also examples of this kind of lens. Ropinski et al [9] extended the 3D Magic Glass work to enable arbitrary 3D convex shapes as a bounding space.

We are extending the concept of view filters like 3D Magic Lenses to enable their use in a collaborative space and we are adding the concept of task translators like 3D Toolglasses in a way that makes this approach a central part of a 3D application in Croquet. This requires a new approach to how we manage the individual users input and how it is applied to the resulting replicated model. We refer to this architecture as interactor-task-replicant.

This interactor-tasks-replicant approach is similar in many respects to the model-view-controller or MVC architecture originally pioneered at Xerox PARC in Smalltalk [3,4,8]. The MVC breaks an application into three different parts. The model is basically a smart database containing the core data of the application. The view is a particular interpretation of this data. The controller is an object that interprets the users actions to modify the model. In this approach, there is only one

model, but there can be any number of views and controllers.

The actual invention of modeless interface design with its modern meaning was also done at Xerox PARC. This definition was that a mode was something you had to issue a command to get out of to do what you wanted to do, "modeless" meant you could always do the next thing you wanted to do regardless of the state of the system. For example, windows were "modeless" even though they gave you a kind of pseudo-mode, because you could always go to anything else on the screen and issue a new command (and the system would automatically clean up after you). The Smalltalk style of text-editing was "modeless", because you weren't trapped in an "insert" or "replace" - the insight was that selecting between characters would always give you a "gap" of some size, from 0 to n, that would be "replaced" by type-in followed by the cursor - hence "modeless".[5]

3. Wicket

Wicket is the exemplar collaborative CAD application we are building in Croquet to test out the ideas of the interactors-tasks-replicants. Many of the ideas behind "Wicket" came from earlier work on Virtus Walkthrough[10], as well as more recent efforts inside of Croquet itself[13]. The first attempts at Wicket utilized some simple drawing and extrusion capabilities, but this required an intelligent object already in the space, and forced all users of the system into the same mode of interaction. The advantage was that this mode was localized to that object, but we were extremely limited in the capabilities and extensibility of these tools.

Further, it was difficult to create a simple and accessible framework for changing modes to access additional toolsets. We certainly did not want to have an entire editing suite as part of the users main interface everywhere he went. Nor did we want to complicate the object itself with a complex editing framework. What we needed was a dynamically extensible toolkit for creation and editing of these objects that would allow for any level of complex interactions without overcomplicating the rest of the system. Further, we wanted to easily be able to "put away" the editing mode. Finally, we needed a way of visually packaging these tools so that the user would immediately recognize their role and would be able to utilize them without having to memorize how they would be used. In short, we needed the equivalent of a discrete application inside of a seamless shared 3D environment.

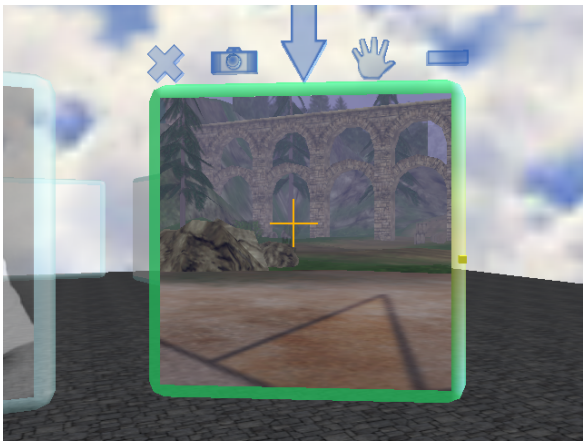

Figure 1. A typical portal.

We will use the Wicket CAD system to ground this discussion in an actual user interface problem, and how this approach solves it.

4. Wicket Overview

Wicket is a modified Croquet portal filter that is carried inside of a 3D window. This portal displays an enhanced view of the content on the other side. The user can select an action button from a palette overlaid on the window that determines the task that is executed when the user clicks the pointer through the window. The user can see and interact with the replicated editing surfaces and objects that are created because of these actions through the portal filter, though they are not visible outside of this view. Of course, the user can easily carry around the filter, even with it filling the entire screen, allowing him to reposition the interface as necessary. Here, the buttons and the portal filter are interactors - objects that the user can see and interact with, but which do where the state of these objects is not replicated. The actions are tasks -replicated objects that have a relatively short life span that are used to communicate and modify other replicated objects. The replicated surfaces and objects that are created in the space are replicants.

5. Filters

The development of the 3D portal in Croquet has proven to be extremely useful. Originally, it was intended to act as a gateway between environments. Croquet allows fully dynamic connections between worlds via spatial portals. 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.

Figure 1 shows an example of a typical portal linking one space to another. Note that users actions are directly transferred to the other space with no translation of the actions into tasks in this case. The user can pick and move remote objects as if they are local.

We discussed the idea of utilizing the portals as view filters, and the Japanese National Institute of with one of us, implemented a system where notations on 3D models of archeological artifacts can either be made visible or removed when viewed through the portal filter. This was essentially a 3D Magic Lens described above.

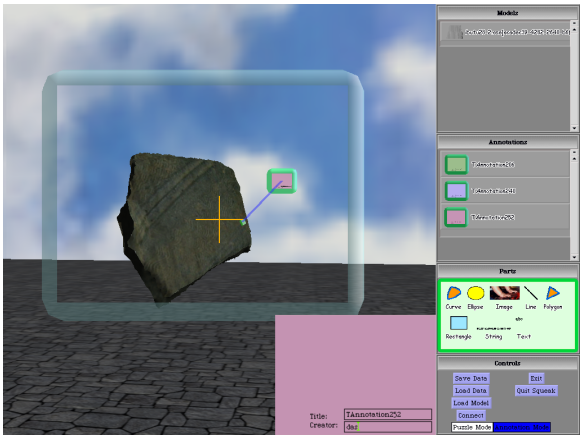

Figure 2. A portal as information filter.

Figure 2 is an architectural artifact that has a note attached to it. In this case, the filter acts to turn off unwanted annotations, though it can easily be used to add them.

Other examples are displaying the environment in wireframe mode. A global render flag is set when we traverse the portal, which forces everything through it to render as a wireframe image.

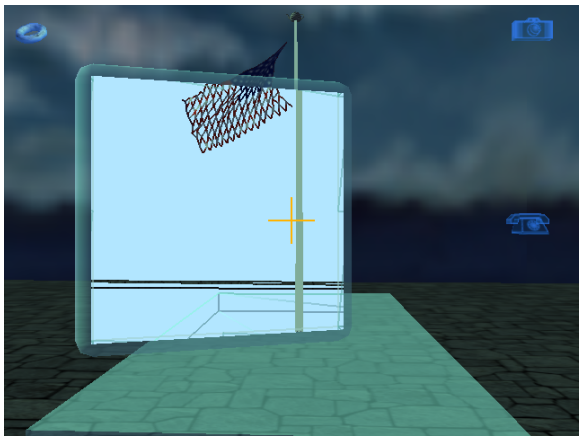

Figure 3. A filter Displays a wire frame View

The filter in Figure 3 displays the content through the portal as a wireframe image.

These filters are also composable. In this example, when two wireframe portals co-exist in the same space, the first wireframe portal will render the world as a wireframe image, where the second one, when viewed through the first, will render it as a normal image again.

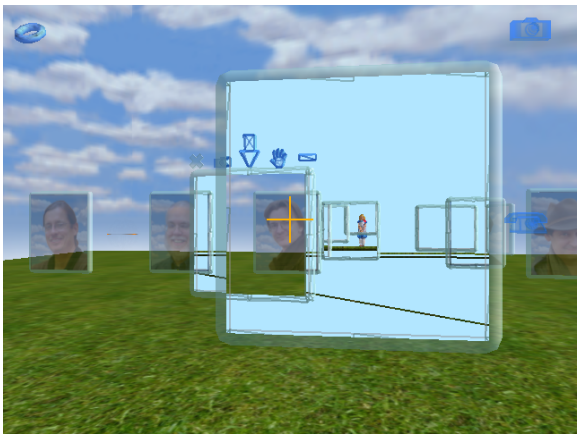

Figure 4. A view with composed two filters.

In Figure 4, we see two wireframe portals, one in front of the other. Notice that when viewing the second portal through the first, it changes back to normal rendering.

Croquet filters are created by subclassing the portal class. The portal can modify the way the content on the other side is rendered, or it can set a flag that is used to turn objects rendering on and off. It can also be used to scale object, as in the case of the Croquet 3D scrolling portals.

Figure 5. A scrollable 3D portal.

This is an example of a scrollable 3D portal, or "World in Miniature"

6. Interactors

Interactors are local non-replicated objects that exist in a world. There may be, and usually are duplicates of these objects in all of the shared spaces, but each of these acts independently so they are not full replicants as described below. Note that interactors can be fully replicated as well. If that occurs, then all users would share the same state.

In the case of Wicket, the interactor is the 2D filter portal and it's associated interface. The filter portal is used to display the additional controls, editing surfaces, and incomplete objects that the user is constructing in the space.

Figure 6 shows a prototype of the Wicket interface. The user is selecting one of the buttons that will trigger the creation of a replicated task.

When viewed from outside of the Wicket portal, all that the user sees are the completed objects without the associated editing framework.

As shown in Figure 7, the editing controls and surfaces are only visible through the Wicket portal.

The Wicket portal can even be used in full-screen mode and act as if it is the entire scope of the interface. This allows the user to have a more standard kind of interface while still maintaining the ability to "walk away" at any time. Further, the user can even "lock" the interface to his own frame and carry it around the environment so that he can easily change the viewpoint from which he is working while still having his entire CAD interface with him.

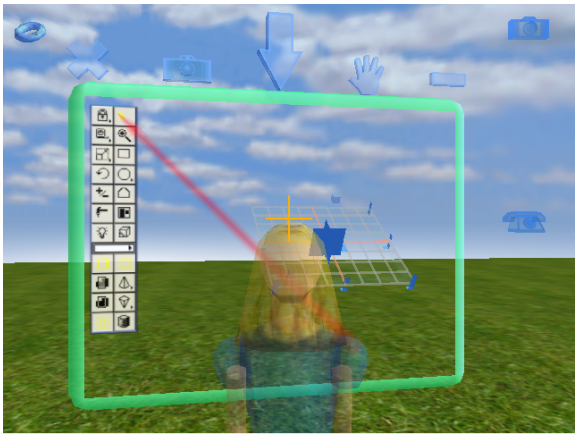

Figure 6. A prototype of wicket interface.

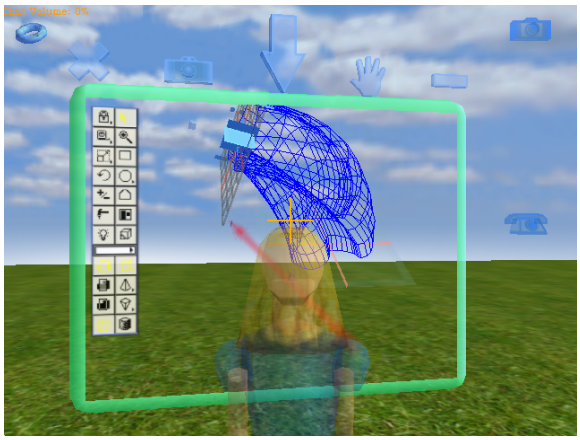

Figure 7. Editing controls and surfaces.

Figure 8 shows the view the user has when the user zooms to the window and it takes up the entire screen. This interface can even follow the user as he moves around his designs.

In some ways, an interactor is similar to the view in the model-view-controller architecture. In a sense, it acts as a costume for the CAD system, giving the user a system that he can view and can manipulate. Further, just as there can be multiple views with the MVC architecture, there can be multiple interactors focused on the same replicated objects (the replicants). However, a replicant can also be a visual object and itself can act as an interface to another replicant. This is a generalization of the MVC model.

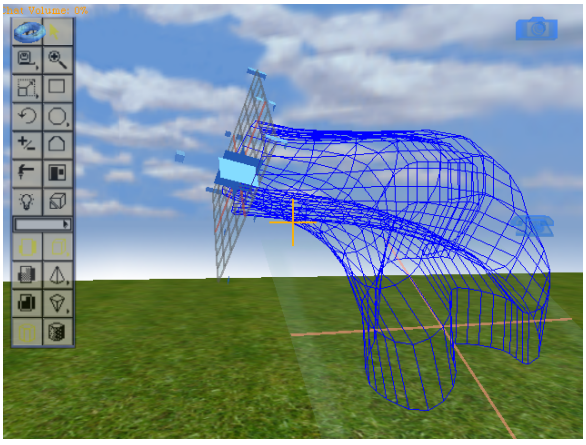

Figure 8. Wicket interface in full screen.

7. Tasks

Tasks are replicated action objects that are generated by the users actions interpreted by the interactor. In the case of the Wicket system, a task is generated when the user clicks through the Wicket

portal. The actual task that is generated is determined by which pseudo mode has been selected by the user view the button interface.

As seen in Figure 9, the action of the user (in gray) is captured by the Wicket portal and the action is modified to generate a new replicated task object (in black) whose actions are applied to the target replicated object. Tasks can be as simple as a message send to an appropriate object. For example, sending a message such as #removeSelf will cause the object to itself from the scene graph. Tasks may also be far more complex allowing for the generation of complex 3D objects based upon parameters and extrusion methods specified by the user in the Wicket interactor. The task may be as involved as a pointer-down on a surface followed by tracking the drag of the pointer across the surface to generate a rectangular solid, followed by an up-pointer event which terminates the construction sequence.

Note that the image in Figure 9 shows the CAD surfaces displayed outside of the window. This is for illustrative purposes only. In fact, these surfaces are clipped to the view portal.

Here we see the user manipulating the surface object that controls the path of how the newly created object will be extruded.

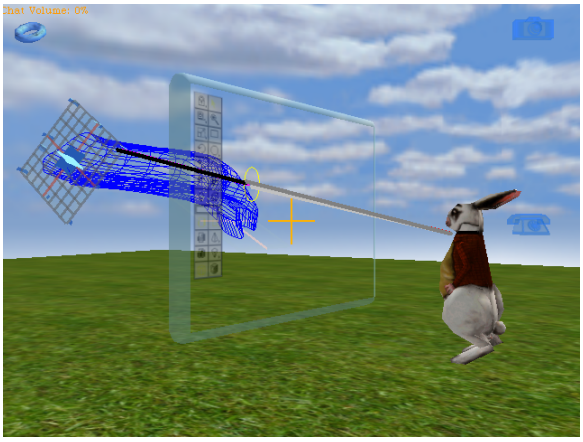

Figure 9. A view of modified task.

8. Replicants

The final member of our triad is the replicant. This serves the same niche as the model in the MVC, except it is typically a replicated object and of course will most often have its own very visible representation. Further, the replicant may itself act as an interactor. In fact, this is one of the major ways we see the Croquet infrastructure developing over time. The interactor suite is used to create replicated content which in turn is scripted to make it into a useful tool which can be used to create even more content.

9. Future Work

Many of these interfaces are still preliminary designs. In particular, Wicket has only scratched the surface of the potential of this approach to CAD. The goal is to develop an industrial quality system that will allow teams of users to work together in the design process. In a sense, Wicket is the ultimate example of the power of deep collaboration in Croquet.

We are currently developing a more formal structure for using the interactor-task-replicant architecture. One of our goals is to avoid the complexity trap that made the MVC model so difficult to work with, yet also avoid the unstructured complexity that Morphic [5] developed into in Squeak.

The concept of the interactor can also be applied to the entire interface. Currently, the user moves through the space utilizing the mouse cursor position relative to the center of the screen. Alternative models of interaction are the "mouse-look" interface favored by many first person computer games, and of course there are many other approaches. We can use this interactor model to translate the mouse position into changes in the users position in the same way we translate the meaning of the pointer interactions.

10. 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. The interactor-task-replicant extension to the model-view-controller architecture is a powerful and easy way to develop complex applications that scale nicely with multiple users.

This architecture makes it straightforward to to create orthogonal tools that can operate on the same data without interfering with each other, and allows us to have a powerful system without either making the objects themselves more complicated or by adding complexity to the user's standard capabilities. In short, this approach gives the user the capabilities he needs when he needs them, and allows a great degree of control over the replicated objects without having to modify their inherent capabilities.

We also hope that Wicket can act as a centerpiece for the Croquet user community, not just as an exemplar application like MacPaint and MacWrite were for the original Macintosh, but as a focus of open source development in its own right.

Acknowledgements

We would like to thank Kim Rose, Patrick McGeer, and Patrick Scaglia for their help and insight. The

Croquet project is generously supported by Hewlett Packard Corporation, and Applied Minds, Inc.

References

1. Bier, E.A., M. Stone, K. Pier, W. Buxton, T. DeRose. Toolglass and Magic Lenses: The See-Through Interface. Preoceedings of SIGGRAPH '93. 73-80.
2. Engelbart, Douglas, 1968 NLS Demonstration Video: http://sloan.stanford.edu/mousesite/1968Demo.html.
3. Goldberg, Adele, David Robson. A Metaphor for User Interface Design, *Proceedings of the University of Hawaii Twelfth Annual Symposium on System Sciences*, Honolulu, January 4-6, (1979) pp. 148-157.
4. Goldberg, Adele, David Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley. 1983.
5. 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. 1996. pp. 511-578.
6. Krasner, Glenn E., Stephen T. Pope. *A Description of the Model-View-Controller User Interface Paradigm in Smalltalk-80*. ParcPlace Systems. 1988.
7. Maloney, J. The Morphic User-Interface Framework. Squeak: Open Personal Computing and Multimedia. Ed. M. Guzdial, K.Rose.Prentice-Hall. 2001.
8. Reenskaug, Trygve. Thing-Model-View-Editor. Internal Xerox Note. 1979. http://heim.ifi.uio.no/~trygver/mvc/mvc-index.html.
9. Ropinski, T., K. Hinrichs. Real-Time Rendering of 3D Magic Lenses having arbitrary convex shapes. Journal of WSCG. 2004.
10. Smith, David A. Virtus WalkThrough. Virtus Corporation. 1990.
11. Smith, David A., Andreas Raab, David P. Reed, Alan Kay. *Croquet User Manual* v.0.01 Web site: http://www.croquetproject.org.
12. Smith, David A., Alan Kay, Andreas Raab, David P. Reed. *Croquet – A Collaboration System Architecture*. C5: Conference on Creating, Connecting and Collaborating through Computing. 2003.
13. Smith, David A., Alan Kay, Andreas Raab, David P. Reed. *Croquet: A Menagerie of New User Interfaces*. C5: Conference on Creating, Connecting and Collaborating through Computing. 2004.
14. Stoakley, R., M. Conway, R. Pausch. Virtual Reality on a WIM: Interactive Worlds in Miniature. CHI '95. ACM Press, 265-272.
15. Viega, J., M. Conway, G. Williams, R. Pausch. 3D Magic Lenses. UIST '96.
16. Sutherland, I. "The Ultimate Display". Proceedings of IFIPS Congress 1965, New York, NY. May 1965, Vol 2. pp 506-508.