Design of a Collaborative Multi User Desktop System for Augmented Reality

Hermann Wurnig
wurnig@cg.tuwien.ac.at

Institute of Computer Graphics
Vienna University of Technology
Vienna / Austria

Abstract

The performance of 3D graphics hardware and rendering systems is constantly increasing. Despite this development, concerning research, 3D desktop systems are still waste lands compared to their 2D counterparts. Only very few system exist, and those are sparely documented. We are currently designing and implementing a three dimensional virtual desktop. This paper provides insight into the design goals, the structure of the system and describes the most important and interesting underlying concepts.

Keywords: augmented reality, interaction, collaborative work, multi user desktop system, 3D windows.

1. Introduction

Nearly all of today's computers are running window based 2D desktop systems like MacOS, Windows95, OS/2 or others. These systems allow the concurrent execution of multiple applications, provide mechanisms to group related objects and ease the use of available resources. Hence, it is necessary to implement a desktop system for an 3D environment as well. This will have advantages not only for the application programmer, but also for the user, because different applications will be uniform in appearance and behavior.

Basically there are three foundations on which our virtual desktop can be built:

- generalization of the 2D desktop metaphor to 3D. The field of 2D desktop systems is very well researched and most people do already have knowledge in working with them. This approach has two drawbacks. Unexperienced users will need more time to get familiar with the desktop system, and techniques that were designed for 2D cannot entirey exhaust the capabilities of a 3D work space.
- creation of a 3D desktop metaphor that is not based on 2D at all. The goal is intuitive interaction with the real world as model. Knowledge of 2D systems is neither advantageous nor disadvantageous.
- intuitive interaction, supplemented with suitable concepts of 2D desktops. If we design manipulation in the 3D environment as intuitively as possible, the virtual desktop will be easy to handle and predictable even for unexperienced persons. Some concepts of 2D desktop systems might as well be useful in 3D. Instead to reinvent them we could try to extend their functionality

and transfer them to the new environment. The necessary preknowledge for working with the system must be held at a minimum.

2. Research Background

Our virtual desktop will be hosted in a 3D environment called *Studierstube* [Szalavari et al., 1998] (german word for “study room”), which is mainly aimed at scientific visualization, but can also be used for various other application classes like games, education and presentation tools. Its main characteristics are:

- *Collaborative work*: Studierstube is designed as a multi user environment, where different users can not only work at the same time, but also in a collaborative fashion. This makes control structures necessary that allow collaboration and keep the interference between the users at a minimum.
- Use of *Augmented Reality*: while in Virtual Reality opaque head mounted displays (HMDs) are used, Augmented Reality employs see through HMDs. The user is able to see the real world, which is augmented by three dimensional computer generated objects. Position and orientation of HMDs are determined by a magnetic tracking device. The computer uses this data to render the scene from the viewing position of each user.

2.1 Interaction in Studierstube

For manipulating the 3D world in Studierstube we use a two handed interaction technique called the Personal Interaction Panel (PIP) [Szalavari and Gervautz, 1997]. It consists of two parts. First, the pen, which is a stylus like gadget that is held in the user’s dominant hand. It is equipped with a tracker sensor and has a virtual representation in the desktop system. Since most people are experienced in working with pens, it can be efficiently used for various tasks. Examples are selecting and manipulating objects, operating with interaction elements like buttons or sliders, and many more.

The second part of the PIP is a notebook sized wooden panel with a tracker sensor attached to it. In contradiction to the pen, it is held in the non dominant hand and used to arrange application controlling elements on it. One of its advantages is that the user has fast access to it and does not need to search through the whole surrounding in order to find it. This is particularly valuable, because the objects on the PIP represent the most important interface to the application
Figure 1 shows a calculator application. The calculator is placed ot the PIP, its buttons can be manipulated with the pen. For further details on Studierstube and its use for scientific visualization see [Fuhrmann et al., 1997] and [Krutz and Zajic, 1997].

3. Related Work

As we have seen before, we use magnetic tracking. A transmitter generates the electromagnetic field, that is measured by sensors mounted on tracked devices. The accuracy decreases with growing distance form the transmitter. If the sensors are too far away, the position data is noisy and fine grained working is not possible any more. Hence, the working area is very limited. There are some interaction techniques which allow selecting objects that lie beyond the user’s reach. These are arm extension like the Go-Go Technique [Poupyrev et al., 1996], ray casting methods [Mine, 1995] or others as World In Miniature [Stoakley et al., 1995]. These techniques are mainly designed for Virtual Reality and can normally not be used in Augmented Reality environments.

Figure 1: Working with a calculator as an example for interaction with the PIP in Studierstube

[Tsao and Lumdsen, 1997] describe the design of a general purpose workspace in 3D. The virtual world is segmented into independent volumes, called CRYSTALS or 3D windows, which may interact with each other. The CRYSTALS are created and owned by independent applications, that can be launched and terminated interactively. In difference to the most part of today's virtual environments, CRYSTAL is not dedicated to the context of a specific discipline like architecture, surgery or art, but can be used for a wide range of applications. The goal of this project is to develop a 3D windowing system and find the best way to generalize the merits of 2D windowing to 3D workspaces.

Our work is based upon the approach described in [Tsao and Lumdsen, 1997], but it is impoved by two essential concepts that are missing in CRYSTAL: support for multiple users and collaborative work.

4. Comparison of 2D and 3D Desktop Features

In the introduction we have seen three possible approaches for building a 3D desktop system. Each of them has advantages and disadvantages. What we are going to use is the third one.

The interaction is modelled on the real world, which guarantees that the desktop reacts in the way we expect it to. For example, if we grab and object, it will directly follow the movements of our hand until it is released again. Such a system could be handled even by a small child that has no experience with computers. It is clear that this behaviour not only makes working with the virtual desktop easier, but also leads to a broad acceptance of it.

Since we are developing a system that is able to concurrently execute multiple applications for different users, we need control structures that allow:

- grouping and selecting objects,
- moving objects between applications,
- hiding applications when they are not needed,
- focusing on one application at a time.

In the next sections we will analyse some 2D desktop concepts that can be used to obtain the required functionality. Furthermore, we will see how these concepts can be adapted to fit into our 3D environment. This transmission process from 2D to 3D is very delicate, because the necessarity of preknowledge must be avoided and the emerging structures should be as transparent as possible to the user.

4.1 Windows

The most important objects of all 2D desktops are the windows. They are rectangular regions with border, menu and status bar. The reason why windows are designed as rectangular, axis aligned areas is the graphics hardware. Most of today's graphics cards can accelerate the movement of rectangular parts of the screen.

The main task of windows is to group objects of an application, which makes it easier for the user to work with. Windows are a graphical framework to assemble related objects. We need to migrate this concept to our 3D desktop system.

If we had no restrictions, it would be possible to define 3D windows as areas of an arbitrary shape. Just the objects that lie within this shape are visible, everything else is clipped at the window's border. Unfortunately, clipping against arbitrary shapes is very costly. Even with only few windows, the frame rate would decrease dramatically. That is why we choose to use quadric 3D windows we call *"magic boxes"*. Clipping is done with the 6 hardware clipping planes supported by OpenGL, which makes this approach very efficient. Opposed to 2D systems the sides of the magic boxes do not need to be axis aligned.

4.2 Minimizing Windows

Due to the magnetic tracker, the working area is pretty limited. We have to add a mechanism that allows us to temporarily remove windows that are not needed. In 2D desktops, windows can be minimized. Instead of the whole window, a small iconic representation of it is displayed. This concept can be used in 3D as well, where the window is reduced to a small 3D icon.

Imagine the following situation: you are downloading a large file from the web and a window shows the current status. If you iconize this window, you loose all informations about the download progress. From time to time you have to reopen the window to see what amount of data has been received yet. It would be nice if the icon provided this information. In our system this is possible. In iconized state, a flat seam [Schaufler and Schmalstieg, 1998] can be used to show the current contents of the window. It is the same effect as if the client area is rendered into an icon texture. Figure 2 illustrates this concept.

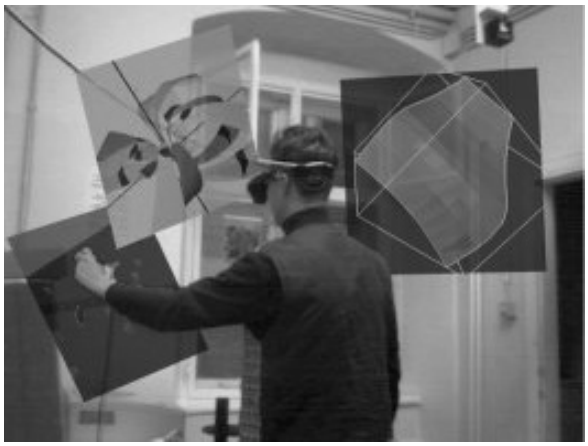

Figure 2: seams used for displaying the contents of three windows

4.3 Focusing

When we look at 2D windowing systems, there is always one window that has the focus at a time. In X, two policies exist to change the focus: *"move to focus"* and *"click to focus"*. If a window has the focus, it gets the input from the keyboard and its border is highlighted.

In our 3D desktop the situation is similar, but a bit more complicated because of the multi user aspects. Every user has his own focus, so different windows can be focused at the same time, or one window can get multiple focuses.

As above, a focused window must change its appearance, so the user can always identify the window he is actually working with. Since every user only needs to know, which window has his focus, this information is displayed user dependent. That means, when the scene is rendered for a user, the system determines his focused window and displays it differently.

Only focused windows are allowed to consume 3D events that occur inside of them. This is very important. Overlaps in 2D can easily be resolved, because at every point exactly one window is visible. In 3D this is not the case. It can happen that two windows make use of the same space. If an event occurs in that space, only the window that has the focus is allowed to react to it. If none of them is focused, both will ignore the event.

To change the focus we can utilize the two policies known from X.

4.4 Modal Windows

The upper concept can be used to realize modal windows, which block application as long as they are open. For example, modal windows are necessary when a program needs some input for being able to continue its work. Therefore, an application locks the focus on a specific window. As long as this lock is maintained, no other window of the same application can react to incoming events. This locking is not done system wide. Hence, no other application if affected by it.

4.5 Drag&Drop

Drag&Drop was one of those concepts that made working with 2D desktop systems as comfortable as it is today. Whether it is used for copying files, dragging WWW links or moving text around, it eases the work and speeds up interaction in most applications. If we take a look behind the curtains, we will find out that the underlying concept is to grab objects, move them around and place them somewhere else. That is what we are doing in the real world all the time.

These reasons suggest to add drag&drop to our 3D desktop system. Therefore, a database of dragable objects and their geometrical representation is needed. When an application supports drag&drop, is checks, if all of its object types are already stored in the database. If not, it creates new entries and fills them with the necessary data.

An application can start drag&drop actions if necessary. The object's geometric representation is then linked to the pen. When it enters a window that can take objects of the specified type, the pen changes its appearance to indicate it.

4.6 Copy&Paste - a virtual clipboard

On top of drag&drop, we can build a virtual clipboard. This is an application to which objects of any type can be dragged. The user can attach the clipboard to the pip and browse through its contents. If a user needs a copy of an object, which is currently in the clipboard, he can simply drag it from there. And dependent on the way the object is selected, either it is removed from the clipboard or a copy is left there.

5. Overview of the system

In this section we will take a look at the design of the system, its components and their functionality. Figure 3 gives an overview over the different layers of which the desktop system consists

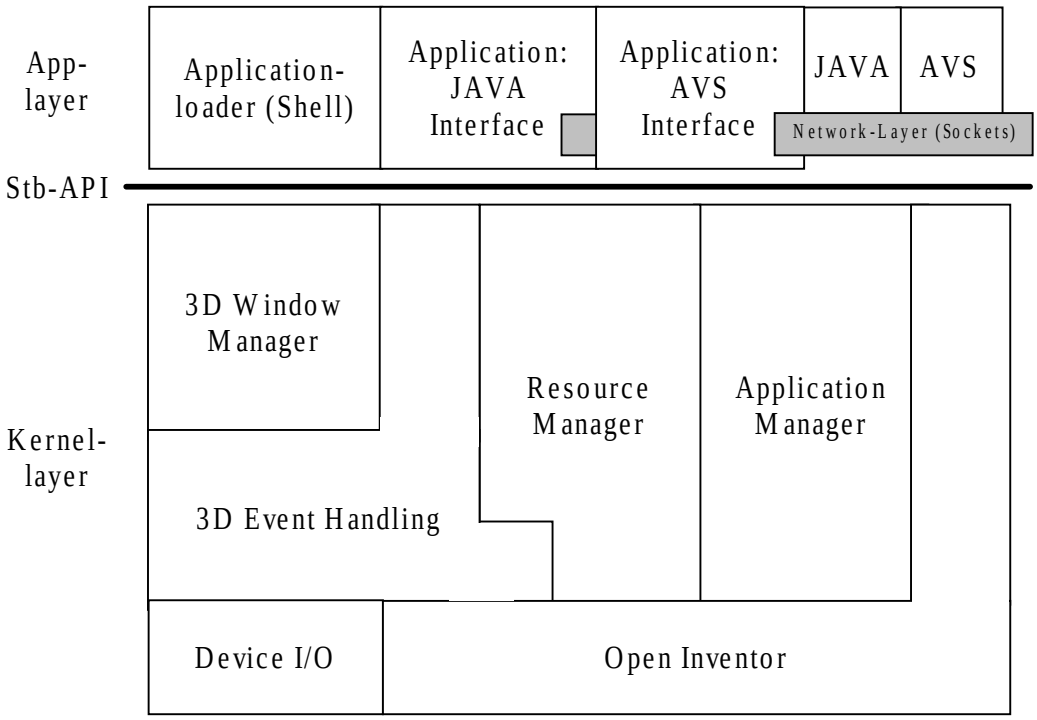

Figure 3: Layer model of the virtual desktop, the interface (Stb API) and three of its applications: Application Loader (see below) and two frontends, connected to AVS and JAVA using TCP/IP.

Both the system itself and all of its applications are based upon Open Inventor. This is a class library providing easy access to 3D graphics. See [Mentor, 1994] for further details.

5.1 The Kernel Layer

The system's kernel mainly consists of five components: Device I/O, 3D event handling, window manager, resource manager and application manager. In the next sections each of them is described in more details.

5.2 Device I/O

As we have seen before, all of our input devices (pens and PIPs) and output devices (HMDs) are equipped with magnetic tracker sensors. The tracker server, a linux pc that is connected with a Polhemus Fastrak tracking device, reads the data from the tracker hardware and multicasts it over the net. Applications may join this multicast group and get all the information like position and orientation of the tracked devices as well as the status of the pen's buttons.

5.3 3D Event Handling

Open Inventor does already provide an event handling system. If an event occurs, an instance of the appropriate event class is created and supplied with informations about the event. The scene's objects can access the event and react in some way to it.

Unfortunately, Open Inventor's event system is strictly screen based and only works with mouse and keyboard as input devices. Whenever an event like a mouse move or a button click occurs, the position of the mouse pointer is given in screen coordinates. Furthermore, no real 3D picking is implemented. Instead, projector rays are used that start at the camera's position and go through the screen plane at the location of the mouse pointer. The application gets a list of objects, which intersect with the projector ray.

This event system is not useful for real three dimensional environments. But fortunately, Open Inventor is designed in a way that makes it easy to extend its functionality.

Based upon the tracker data from above and the extensibility of Open Inventor, new classes were derived, which form the new event system of Studierstube. Their purpose is to detect changes of the tracked devices (movements or button clicks), create the according event and distribute it to the scene. The big advantage of this approach is that it is perfectly embedded in the structure of Open Inventor. 3D events can be treated in the same fashion as all other events.

Picking of objects is not done with projector rays any more. This interaction technique is not supported, because there is an easier and more efficient way for selection. If an object wants to know, whether an event occurred within its reach or not, it simply tests the event's position against its own bounding box. Such tests can be performed very efficiently, because Open Inventor keeps a cache of bounding boxes for its own use. The only drawback of this approach is the fact that bounding boxes are normally larger that the object they belong to. It can happen that an object consumes events that are not inside its geometry.

Another feature of our event system is the ability to grab events. Imagine the following situation: a user wants to change the position of a slider, so he moves the pen into the slider geometry and presses its button. Until the user releases this button again, the slider must get all of the events for being able to follow the movements of the pen. Furthermore, it must be the only object that the events are delivered to, even if they are not within the slider's bounding box. This is called event grabbing.

The 3D event system is accessible for all nodes of an application. If, for example, a slider is added to a window, it can directly access events and react properly. This makes handling of events easier, because less overhead is involved. Nevertheless, this concept is far too limited. It will often be necessary that an application keeps track on all events that occur within its windows. Just think of a 3D drawing program. That is why the window manager, the next part of the system we will look at, extends the approach by an comprehensive message passing system. It is up to the application programmer to choose the method that fits best to his needs and requires the least work.

5.4 The Window Manager

The main task of the window manager is providing an interface between the application and its windows. It is responsible for creating windows, passing messages from the windows to the contained applications, setting the focus and managing drag&drop.

The creation of windows is straight forward. The application calls createWindow, a method of the window manager object, delivers all necessary parameters and gets a handle for the new window in return. As soon as it is created, the application can access the window's functionality and directly change its settings. There is no need to involve the window manager for actions like changing the window's size, its border or the contents of its client area.

When changes are applied to a certain window, it may be necessary to update its client area or react in some other way. To make this behavior possible, the window sends messages to the window

manager, whenever an event occurs that is directly related to it. Examples are moving, resizing or closing the window, focus changes, drag&drop or 3D events. The window manager filters these messages and forwards them to the contained application.

5.5 The Resource Manager

This part of the system encapsulates the existing input resources, which are pens, PIPs and HMDs. Even though the latter are output devices, the system reads their position and orientation from the tracker. So, they are generating input as well. It can even be necessary to use this data for attaching geometry to them.

The encapsulation has two levels. Firstly, the individual resources can be accessed in an uniform manner. And secondly, resources are grouped and mapped to users. Each user has exactly one HMD. Beyond this he can optionally be equipped with a pen, a PIP or both. The big advantage is that applications must only deal with users. They do not need to know, which tracker station the resource is connected to. This relation is resolved by the resource manager.

The system's flexibility will be increased as well. If it becomes necessary to reconfigure the tracking hardware, the system must only be informed of the new tracker-user mapping. This can easily be done with a configuration file that is read at startup. No recompilation of applications is necessary.

One kind of resources has to be treated in a different way: the PIP. Because it will most of the time be used to control applications, it must be accessible, even if it is held within a window. Hence, care must be taken that the PIP is always the first to get events.

5.6 The Application Manager

The application manager is responsible for loading and removing desktop applications. Furthermore it keeps track of the windows and resources that are assigned to them. If an application exits, all of its windows are explicitly closed and the resources are freed.

5.7 The Application Loader

When the virtual desktop is started, every user gets an application loader. This tool could be compared with the Windows Explorer, but it is much simpler. Its main task is to search the directories for applications, which are represented by 3D icons, and start them.

6. Collaboration

One of the most important design issues is the system's ability to enable and support collaboration among the users. Therefore, applications should not be seen as the caller's property, but rather as shared resources. It is, of course, possible for a user to be the only one who works with a certain program. But this will not be the common case.

Basically, applications can provide two different levels of collaboration:

- Every user works in his private context, changes of his own parameters do not influence the other users.
- Some parameters have global effects, every user notices changes of these parameters.

In each level we have to find appropriate mechanisms to guarantee that the interferences between the users are kept at a minimum. We will look at two examples that show the differences and problems.

6.1 Virtual Blackboard

This application creates an ordinary blackboard in the environment. Multiple users can write on it at the same time. Every user is able to change color and thickness of the pen he is writing with. These are local parameters that differ from user to user.
The application has to know which persons want to draw on the blackboard. We can assume that the one who starts the application is going to use it, at least for now. But what about the others? There are three solutions to this problem:

- The easiest one would be that all users get the application's controls, but that can be distracting for those who do not want to work with it. This kind of approach is only suitable for presentation or education programs, where one dedicated supervisor controls the run of events.
- Another solution is the following: before a program is launched, the system determines, whether it is already running. If not, the application will be loaded, else the user is linked to an existing instance of it. This approach does also have some drawbacks. If two or more blackboards are available, the user has to specify at startup, which one he wants to work with. The interaction involved in this process is rather time consuming.
- What applies best to our situation is the third solution. When the user loads an application, a new instance of it will be generated. This is the behavior we expect, and the system becomes more predictable. To join an already running program, it is sufficient to click into its main window area. Subsequently, the window manager creates an *"user wants to join"* message and sends it to the application. When multi user mode is supported, all necessary steps can be taken to integrate the new user. Otherwise, the message is ignored. The application keeps a count on its users and terminates when the last one leaves.

6.2 Streamline Visualization

This application displays streamlines of a three dimensional vector field and animates their surface to give an impression of direction and velocity of the flow. To get a better overview, the size of the steamlines can be changed. Therefore, every user is provided with a slider for setting the size of the model. This is a global parameter, that has the same value for all users.

In the worst case, two users could try to change the scene size concurrently. This must not be possible. As soon as one user starts to manipulate his size slider, all other sliders have to be locked. This locking must be maintained until the manipulation has finished.

The easiest way to achieve this is the following: only one instance of the slider is created and given to each of the users. This makes interchanging updates unnecessary. As soon as one of the users grabs the slider, it does only listen for events from its grabber. All others are ignored until the slider is released again.

7. Conclusions

Until now we have only been working with stand alone applications that used the whole work space of Studierstube. These applications supported multiple users, but concurrent execution was not possible. What we expect from the virtual desktop is:

- easier application development
- easier handling due to uniform appearance and behavior
- concurrent execution of applications in a shared workspace

8. Future Work

Currently, rendering is done at the same workstation that hosts the virtual desktop. Since both tasks are rather time consuming, a client-server approach, where the server hosts the desktop and the clients render the scene, would be advantageous. Problems arise because of the necessarity for synchronization among the clients.

9. Acknowledgments

Special thanks to Anton Fuhrmann, Dieter Schmalstieg and Zsolt Szalavari for reviewing this paper. Further thanks to Markus Krutz for his Magic Boxes and Andreas Zajic for the 3D event system.

10. References

Fuhrmann A., Löffelmann H., Schmalstieg D.: “Collaborative Augmented Reality: Exploring Dynamical Systems”, in Proceedings of Visualization ’97, pp 459-462, 1997

Krutz M., Zajic A.: “Studierstube - an Augmented Research Environment”, CESCG ’97, http://www.cg.tuwien.ac.at/studentwork/CESCG97/krutz/

Wernecke J.: The Inventor Mentor: “Programming Object-Oriented 3D graphics with Open Inventor”, Addison Wesley, 1994.

Mine M.: “Virtual Environment Interaction Techniques”, University of North Carolina Computer Science Technical Report TR95-020, 1995

Poupyrev I., Billinghurst M., Weghorst S., Ichikawa T.: “The Go-Go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR”, in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), pp. 79-80, 1996

Schaufler G., Schmalstieg D.: “Sewing Virtual Worlds Together With SEAMS: A Mechanism to Construct Large Scale Virtual Environments” Technical report TR-186-2-87-11, Vienna University of Technology, 1998.

Stoakley R., Conway M., Pausch R.: “Virtual Reality on a WIM: Interactive Worlds in Miniature”, in Proceedings of CHI ’95, pp. 265-272, 1995, http://www1.acm.org:81/sigchi/chi95/proceedings/papers/rws_bdy.htm

Szalavari Zs., Gervautz M.: “The Personal Interaction Panel - A Two-handed Interface for Augmented Reality”, in Proceedings of EUROGRAPHICS ’97, 16(3): pp 335-346, 1997

Szalavari Zs., Schmalstieg D., Fuhrmann A., Gervautz M.: “Studierstube - An Environment for Collaboration in Augmented Reality”, to appear in Virtual Reality Journal 1998, http://www.cg.tuwien.ac.at/research/vr/studierstube/jvrs-paper.pdf

Tsao J., Lumsden C.J.:”CHRYSTAL: Building Multicontext Virtual Environments”, Presence Vol. 6 No. 1, February 1997, pp. 57-72