Aalborg Universitet

AALBORG UNIVERSITY

Interaction Styles in Tools for Developing Virtual Environments

Kjeldskov, Jesper; Stage, Jan

Published in:
Virtual Reality

DOI (link to publication from Publisher):
10.1007/s10055-008-0091-0

Publication date:
2008

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Kjeldskov, J., & Stage, J. (2008). Interaction Styles in Tools for Developing Virtual Environments. *Virtual Reality*, *12*(3), 137-150. https://doi.org/10.1007/s10055-008-0091-0

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: December 06, 2025

Virtual Reality (2008) 12:137–150
DOI 10.1007/s10055-008-0091-0

ORIGINAL ARTICLE

Interaction styles in tools for developing virtual environments

Jesper Kjeldskov · Jan Stage

Received: 15 December 2005 / Accepted: 4 February 2008 / Published online: 4 March 2008
© Springer-Verlag London Limited 2008

Abstract This article discusses and compares interaction styles in development tools for virtual environments (VE). The comparison relies on a qualitative empirical study of two development processes where a command language and a direct manipulation based tool were used to develop the same virtual environment application. The command language tool proved very flexible and facilitated an even distribution of effort and progress over time, but debugging and identification of errors was very difficult. Contrasting this, the direct manipulation tool enabled faster implementation of a first prototype but did not facilitate a shorter implementation process as a whole. On the basis of these findings, the strength and weaknesses of direct manipulation for developing virtual environment applications are explored further through a comparison with a successful direct manipulation tool for developing interactive multimedia applications. The comparisons are used to identify and emphasize key requirements for virtual environment development tool interface design.

Keywords Virtual environments · Development tools · Interaction styles · Empirical study

1 Introduction

A virtual environment (VE) is a computer-generated world in which a user can interact with computer-generated objects. Generating a virtual environment and the objects within it is done using multimedia technology with the purpose of providing the user with a certain experience while being immersed into a virtual reality. For several years, virtual environments have been used for experimentation and technological development whereas their practical relevance and usage have been very limited. This situation is gradually changing as virtual environments are increasingly being used in different commercial software solutions. Over the last few years, this growing adoption of virtual environment technology has been much more notable than the technological advances within the area (Brooks 1999) and hence the need for professional development processes is increasing. However, at the moment, we know very little about the process of developing virtual environments and the tools used in this process (Smith and Harrison 2001).

The design of virtual environments has been described and discussed by several authors, (e.g. Burdea and Coiffet 2003). Such discussions are often based on descriptions of specific virtual environment applications for example for professional training (Huang and Gau 2003), general education (Allison and Hodges 2000), scientific visualization (van Dam et al. 2002), medicine (Bartz et al. 1999; Zajtchuk and Satava 1997), military (Encarnacao et al. 2000), and entertainment (Mine 2003). There are also experiments with platforms that provide virtual environments at a much lower cost than usual (Huang and Gau 2003). Finally, there are general inquiries into the many-facetted process of designing virtual environments, including a systematic overview of research activities that can inform the process of design (Scaife and Rogers 2001), a categorization of aspects that characterize effective design of virtual environments (Stanney et al. 2003), and a proposal for methods to support their evaluation (Neale and Nichols 2001).

J. Kjeldskov (✉) · J. Stage
Department of Computer Science, Aalborg University,
Aalborg, Denmark
e-mail: jesper@cs.aau.dk

J. Stage
e-mail: jans@cs.aau.dk

The interaction style deployed in a virtual environment application is a key element of its design. The fundamental question is how the user interacts with the virtual environment and its objects. Methods and guidelines for user interface design embody certain computer technologies. This also applies to interaction styles. Interaction based on a command language was a relevant solution with the character-based display. Direct manipulation emerged from the potentials of the graphical workstation and personal computer. This inherent relation between interface design and computer technology implies that our established guidelines and experiences are challenged when new technologies emerge.

Several studies have evaluated and discussed the relevance of different interaction styles for users of virtual environments. A theoretical framework for analysing manipulation techniques and a test-bed for experimenting with such techniques are presented in Poupyrev et al. (1997) followed by taxonomies for virtual environment interaction styles in Poupyrev et al. (1998). Similarly, a characterization of interaction tasks in virtual environments is presented in Bowman (1998) and evaluated in Bowman et al. (1998) through creation of a highly interactive application, challenging the view of good interaction techniques for virtual environments as being "natural" or at least "similar" to the physical world. Experiments with specific interaction styles for virtual environments count non-isomorphic three dimensional (3D) rotational techniques (Poupyrev et al. 2000), HOMER/Go-Go non-linear manipulation techniques giving the user "stretchable" virtual arms (Bowman and Hodges 1997; Poupyrev et al. 1996) and World-In-Miniature/Voodoo Dolls techniques providing the user with a small representation of a subset of the virtual environment (Pierce et al. 1999; Stoakley et al. 1995). Similar research is more or less non-existing in relation to interaction styles for VE development tools.

A VE development tool is a computer application that supports the construction of a virtual environment and the definition of objects in that environment. This process is also referred to as authoring. As the practical use of virtual environments is increasing, there is a growing demand for tools that support the development of virtual environment applications. Different tools and facilities for development of virtual environments have been presented and discussed. Traditionally, tools for developing virtual environments require a substantial amount of programming defining the virtual environment and its objects through statements in a programming language. However, the development of virtual environments often involves groups of people with no programming competence for whom reliance on programming-based development tools is a hindrance to active participation in the process. In response, a second category of VE development tools has emerged that do not require programming skills in a traditional sense but support rapid development of interactive virtual environment applications by people without 3D graphics or programming experience by means of a visual key framing and simple scripting environments (see e.g. Conway et al. 2000). Instead of deploying a programming approach these tools rely on other interaction styles such as direct manipulation, menu selection, form filling, and tangible input devices (Keefe et al 2001; Schkolne et al. 2001). The tools that do not require any programming provide effective development support for designers without programming skills. However, whether this category of development tools is equally relevant for knowledgeable programmers is an open question. From one perspective it has been argued that development tools for virtual environments must necessarily include a major element of programming support, and that if one wishes to reduce this need for programming specific development tools will have to be tailored to specific kinds of applications (Hendricks et al. 2003). From another perspective, it seems to be an implicit assumption underlying some direct-manipulation based development tools that all groups involved in the design of a virtual environment should apply them—even those who posses significant programming competence. Yet this assumption has not been empirically justified.

The purpose of this article is to discuss and compare the relevance of different interaction styles for developers of virtual environments. Our focus is on the knowledgeable programmer. In Sect. 2 we discuss interaction styles and review selected literature on command language and direct manipulation. Sect. 3 describes the task of developing a virtual environment in general. This includes an overview of the key elements of the implementation of a virtual environment application. And the presentation of two example development tools. One is based on command language as the fundamental interaction style and the other on direct manipulation. The comparison of these development tools is based on an empirical study of two development processes where knowledgeable programmers used the tools to develop a virtual environment application. The study is primarily qualitative. The design of the study is provided in Sect. 4 and the results are discussed in Sect. 5. Section 6 compares the VE direct manipulation tool with a tool that has proved very successful for the development of two-dimensional multimedia applications. Finally, Sect. 7 concludes the article and points out avenues for further work.

It is worth noting that it is not the purpose of this paper to propose one specific tool over another but to investigate into the pros and cons of two different interaction styles for virtual environment development as represented by these specific tools.

2 Interaction styles in development tools

The interaction style is a key determinant of a user interface design. The four major options available for design of this characteristic have been denoted as: command language, menu selection, form filling, and direct manipulation (Shneiderman 1998). Below, we will refer to these as the classical interaction styles.

The traditional category of tools for developing virtual environments is based on an interaction style where the virtual environment and its objects are defined by scripted command statements in a programming language. Command language is an interaction style, where the user issues commands in a formalized language and the system responds by carrying out these commands (Shneiderman 1998). A typical example of this is an operating system where the user issues commands line by line, and the operating system responds by executing the commands one by one. Yet command language is also used to denote systems with facilities for constructing and executing larger collections, or scripts, of commands such as macros and programs (Shneiderman 1998). In accordance with this, we will characterize the interaction style employed in the traditional category of tools for developing virtual environments as command language.

The second category of VE development tools does not require programming in the traditional sense but is operated through direct manipulation. Direct manipulation is an interaction style where the user experiences a representation that can be manipulated directly (Shneiderman 1998). In designing a direct manipulation tool, a key challenge is to find an appropriate representation of key objects and to provide simple ways of manipulating this representation. A direct manipulation tool may also rely on other interaction styles such as menu selection and form filling, but only for secondary interactions that deal with limited issues, for example, the specification of properties of a certain object that is manipulated directly on an overall level. In Sect. 3, we will present a VE development tool that primarily employs direct manipulation.

The literature on human–computer interaction includes numerous attempts to compare command language and direct manipulation and to shed light on the question of whether one is superior to the other. Much of this literature consists of descriptions of the advantages of both approaches whereas the amount of empirical evidence actually comparing them is more limited (Benbasat and Todd 1993). Exceptions count an early contribution that compared file manipulation commands in MS-DOS with direct manipulation on a Macintosh. This study concluded that Macintosh' users could perform the manipulations faster, with fewer errors, and they were more satisfied with the interface (Margono and Shneiderman 1987). In a similar study, where command line and direct manipulation was compared, it was concluded that the users of direct manipulation made only half as many errors and were more satisfied. In this study, the time to perform the tasks turned out to be comparable (Morgan et al 1991). These empirical studies indicate that direct manipulation has an advantage in terms of error rate and user satisfaction compared to command language. However, in relation to task completion time, the conclusions are more varied.

3 Developing virtual environment applications

In this section, we describe key elements of the process of implementing a virtual environment application and present two specific VE development tools.

3.1 The development task

A virtual environment application that visualizes a 3D world consists of a number of mathematically defined 3D models that are covered with colours or textures, for example, pictures or video images. The 3D models are spatially distributed in a 3D coordinate system that the user can experience as a 3D world by viewing the 3D models from a given point in the coordinate system as illustrated in Fig. 1. The correct perspective is rendered real-time by a graphics computer and projected by means of a display system as depicted in Figs. 2 and 3.

A virtual environment application may use a multitude of display systems to visualize the virtual 3D world. Examples of display systems are traditional desktop monitors, head-mounted displays, holobenches, large wall-mounted displays or Caves with three to six sides (Brooks 1999). These display types represent the array of technologies for creating immersive experiences that range from "looking at" a virtual 3D world to "being in" that virtual world (Shneiderman 1998). In this article, we will primarily deal with virtual environments for a Cave.

Fig. 1 A virtual 3D environment

Fig. 2 Outside and inside the six-sided Cave

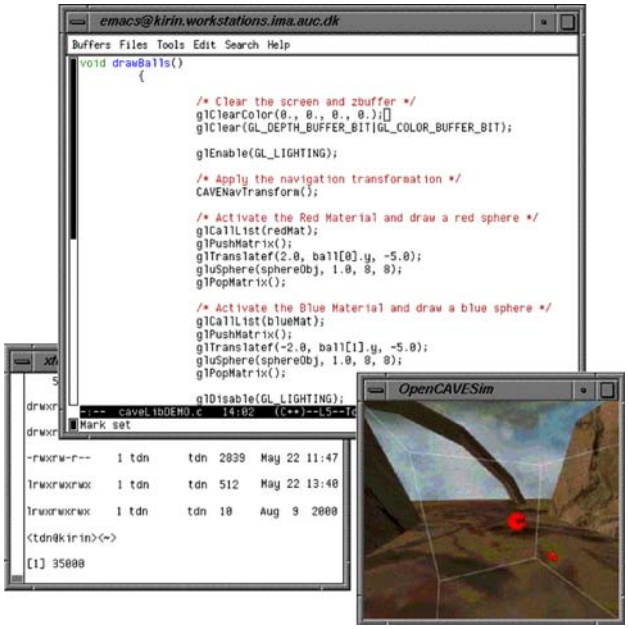

Fig. 3 Development with CaveLib

The six-sided Cave Automatic Virtual Environment (Cave) is currently the display system that offers the greatest level of immersion into a virtual 3D environment. The user is placed in a small cubic room, measuring approx. 3 m on all sides, in which computer-generated images are back-projected on all four walls, the floor and the ceiling (Fig. 2).

Navigation or motion in the Cave is accomplished by means of position tracking or specialized interaction devices. Tracking the position of the user's head ensures that the correct visual perspective is calculated. Interaction with objects in the virtual environment is typically supported by techniques for selecting and modifying 3D objects by simply "grabbing" them just like one would do in the real world. The 3D experience requires shutter glasses worn by the user allowing separate images to be projected to the user's left and right eye and thereby creating stereovision.

The benefits of the six-sided Cave for exploration of virtual environments originate from the vividness of the virtual environment projected and the very high degree of immersion. This is caused by the freedom of movement that is possible inside the Cave and the large horizontal and vertical field of view covered with computer-generated images. Exploration of the virtual environment is much more natural in a six-sided Cave compared to any other display system because the user can move around physically and look in any direction without breaking the illusion of being in a computer-generated world. The primary downside is that physical objects and the user's body itself may occlude the images, thus locally breaking the visual illusion (Kjeldskov 2001).

Virtual environment applications displayed in a Cave are very different from many traditional computer applications. First, the user interface is completely surrounding the user and is presented in 3D as opposed to conventional 2D interfaces covering only a fraction of the user's physical surroundings. Second, the types of applications running in a Cave are typically offering a complete virtual 3D world for exploration as opposed to traditional tools for office or home use. Third, applications running in a Cave are often highly graphical and interactive.

The development of a virtual environment application for the Cave, or any other display system, includes an essential task of constructing the 3D world that will be visualized by the application, which primarily means mathematically defining the objects that make up the environment, as well as different objects that may be added to the environment. Some of these objects are static while others may exhibit dynamic behaviour when the application is running.

3.2 Two VE development tools

Virtual environment development tools usually run on an ordinary, yet powerful, desktop workstation with a traditional 2D display. Existing tools for developing virtual environment applications can be divided into two categories according to their interaction style: command language or direct manipulation. Below we present two specific examples of tools representing each of these approaches.

Within the first category, libraries for creating Cave applications are available for the C and C++ programming languages. One of the most widely used binary libraries for developing virtual 3D environments is CaveLib. Cavelib is an advanced development tool that facilitates development of virtual reality applications characterized by high performance and flexibility. The CaveLib library enables development of highly immersive 3D interfaces for projection in a Cave, or any other virtual reality display system, as well as implementation of a variety of interaction techniques for 3D interaction devices. For preview purposes, CaveLib offers a simple tool for representing the Cave display and simulating simple 3D interaction, cf. Fig. 3.

Using CaveLib to develop a virtual environment application is not very different from developing any other graphical application in a typical programming language. With CaveLib, the developer constructs a program code consisting of commands that point at a number of geometry files and specify the layout of a virtual 3D space as well as the functionality of the application. The commands are constructed in a simple text-editor and are typically collected in a number of script files. To see if the code is working properly, the developer has to compile all the scripts and run the application either in the Cave itself or in a preview-tool. If the code contains errors or otherwise needs to be modified, the developer returns to the text-editor and repeats the cycle.

A professional tool within the direct manipulation approach is dvMockup. This tool enables the developer to create an application by directly manipulating the objects of the virtual 3D world within the preview window combined with the use of menu selections and fill-in forms (Fig. 4). With dvMockup an application for the Cave can be developed by people without programming experience and without doing any programming at all.

When developing a virtual environment application using a direct manipulation tool like dvMockup, the developer imports a number of geometry files and locates them in the virtual 3D space of the application. This is done either by direct manipulation in a workspace-window or by specifying data in forms. The functionality of the application is created and modified by selecting 3D objects and applying behaviours through menu selection. Through the workspace window, the developer can continuously see if the application is working properly.

Fig. 4 Implementing using dvMockup

4 Study design

We conducted a qualitative empirical study in order to compare the support that development tools based on the two different interaction styles described above provide to a knowledgeable programmer. This section describes the design of that study.

Tools. We briefly surveyed potentially relevant tools for implementing virtual environment applications and related them to the aim of comparing direct manipulation tools with command language tools. Based on the survey and the facilities available, we selected the two tools described above: CaveLib and dvMockup. The two tools were already installed, configured, and used extensively by other developers and researchers at our lab who could be consulted when technical problems arose.

Participants. The participants were two development team members with recently completed master degrees in computer science/computer engineering. Thereby they had considerable experience and knowledge about programming in general. In addition they had previously completed a one-semester course on computer vision and virtual

reality and worked with projects within that subject. They received a 1-day introduction to the tools used in the study but had no specific experience with any of them.

Overall task. The development team and the two authors of this article planned and designed the study together. The development team conducted the implementation using the two tools. The comparison of the two tools was based on solving the same overall task. This task was to develop a virtual environment application that visualized a maze in which a user could move an avatar around by means of an interaction device. This task was specified in detail in terms of 14 milestones. Thus, the overall task was solved when all milestones were met. The specific milestones involved tool and Cave set-up (milestones 1 and 2), implementation of a simple application (milestones 3 and 4), implementation of the application visualizing the maze (milestones 5–8), implementation of interaction techniques to facilitate motion of the avatar (milestones 9–12), and adjustment (milestones 13 and 14). The 14 milestones are described in detail in the Appendix.

Hypothesis. Based on the literature on interaction styles reviewed in Sect. 2, we hypothesised that the direct manipulation tool would be superior to the programming tools in terms of the efforts required to implement the virtual environment application specified.

Study procedure. The duration of the study was initially planned to last 3 weeks but had to be extended by a couple of days because of technical problems. The two members of the development team were each assigned to one of the tools to produce the best possible solution for the overall task. During the implementation phase, they were not supposed to communicate with each other about their work, problems, and solutions.

Data collection. The primary means for data collection were private diaries kept by the developers (Jepsen et al. 1989; Naur 1983). After each day of work on the implementation, a developer took an hour to describe the work done and its relation to the 14 milestones, the problems faced, and the time spent on tasks related to each of the milestones. A checklist that emphasized the points that should be touched upon supported the daily writing of the diary. One week into the implementation phase, the diary entries produced so far were reviewed in order to enforce the use of the checklist and increase consistency. The diaries totalled 45 pages (Hougaard et al. 2001).

Data analysis. The primary dependent variables were work practice and development effort. In this article we focus primarily on development effort. Based on the diaries, we have calculated and compared the efforts spent on completing the different milestones of the overall task. As the size and type of application developed is, of course, not representative of all virtual environment applications, the exact times spent on each milestone are not important in themselves but only as measures for comparisons across the two tools. The results of this are presented in the following section.

5 Results

In this section, we present and discuss the findings from the study with CaveLib and dvMockup. On the task level, there were clear differences between the two developers. The developer who used CaveLib was able to meet all milestones, but the navigation technique specified in the task had to be changed due to usability issues. The developer who used dvMockup was not as successful, since collision detection could not be implemented satisfactorily. Otherwise, the final solution was acceptable.

The development time spent using CaveLib amounted to 42.3 h, whereas the time spent using dvMockup amounted to 37.8 h. Thus the total time spent on development with the two tools differs only by 12%. The distribution of time spent on each milestone does, however, reveal clear differences between the command language (CaveLib) and direct manipulation (dvMockup) approaches. This distribution is shown in Fig. 5. Below we will highlight interesting points from this distribution.

Setting up the development tools and the Cave (milestones 1 and 2) amounted to a total of 12 h spent on CaveLib whereas only 3.75 h was spent on this with dvMockup. Thus the developer who used dvMockup only needed about 30% of the time spent using CaveLib. Setting up CaveLib demanded a series of separate tools to be configured for individual tasks, for example, scripting, compiling and previewing, as well as creation of a number of configuration files on both the workstation used for development and the graphics computer that was executing the display system for the Cave. With dvMockup, only one tool had to be set up, and when an application was running on the workstation, only a few scripts were needed before it was also operational in the Cave.

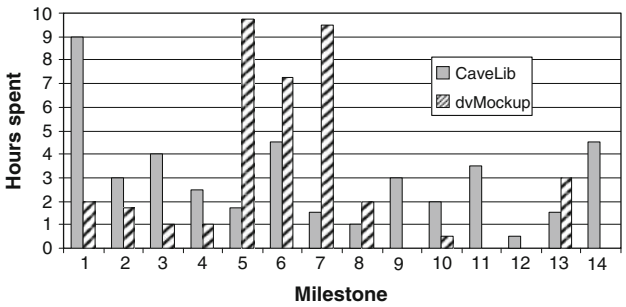

Fig. 5 Development time spent on each milestone using CaveLib and dvMockup

Implementing a preliminary application with the purpose of testing the development and target platform and the connection between them (milestones 3 and 4) took 6.5 h using CaveLib but only 2 h with dvMockup. Again, for dvMockup this is only about 30% of the time spent using CaveLib. Thus up to milestone 4 it is clear that the direct manipulation approach supports a faster kick-off in the development process.

Implementation of the primary virtual environment application, which was the maze specified in the overall task (milestones 5–8), was done in 10.3 h using CaveLib. With dvMockup the same milestones required 27.5 h. So here we see the same pattern where one tool requires only 30% of the time spent with the other tool. Yet this time the roles are reversed, as CaveLib is supporting the faster development. Thus the command language approach seems to facilitate a more effective process in this part of the implementation. The major reason for the considerable amount of time spent with dvMockup is that the tool provides no direct support in a situation where making and running a simple set of commands might avoid numerous repetitions of simple operations. For example, the developer using dvMockup faced the task of manually inserting 800 identical cubic objects into the virtual 3D world, whereas the developer using CaveLib could perform the same task simply by writing a small piece of code. This limitation becomes even more serious when we take the question of scale into consideration. If we compare a small application to a large one, a major difference in amount of work will occur precisely on milestones 5–8 whereas the remaining milestones will largely be unaffected. Therefore, the difference between the two tools on these milestones should even be more considerable if we expanded the scale of the application being developed.

Implementation of interaction techniques (milestones 9–12) took 7.5 h with CaveLib but only 2.5 h using dvMockup. This is a 30% reduction in favour of dvMockup. The time spent implementing interaction techniques with dvMockup is, however, influenced by the availability of supporting software. In a related project considerable amount of time had recently been spent developing off-the-shelf support for implementing user interaction in dvMockup in order to facilitate a general reduction of development effort (Kjeldskov 2001). This has also been denoted as a toolset approach (Willans and Harrison 2001). Had this support not been available, the time spent on these milestones would have been considerably higher.

In CaveLib, all interaction techniques were implemented from scratch. However, this had the advantage that the interaction technique specified in the overall task was actually implemented. With dvMockup it was necessary to modify the task specification by selecting one of the available techniques, which did not fulfil the specification entirely. If the implementation in dvMockup should have fulfilled the requirements completely, additionally programming on device driver level would have been necessary.

Final adjustments of the applications (milestones 13 and 14) took 6 h for CaveLib while only 3 h was spent with dvMockup. The larger amount of adjustments of the CaveLib application primarily consisted of correcting errors with the scaling of 3D objects. This was necessary in order to make the objects fit properly for projections in the Cave. This kind of error was absent in the application developed with dvMockup.

5.1 Qualitative findings from diaries

Although this paper focuses primarily on comparing the different time spent by the two developers on reaching each milestone of the development process, some comments should also be made about the more qualitative statements in the developers' diaries. In qualitatively analyzing diary entries regarding the specific challenges facing the developers during the study, two main themes emerged. These were related to (1) repetitive sub tasks, and (2) making and discovering errors. The quotes below are translated from Danish (Hougaard et al. 2001).

In relation to repetition, the developer using the direct manipulation tool several times reported frustration about having to carry out sub tasks involving repeating the same operation many times over. The most prominent example of this was, as mentioned earlier, inserting the 800 identical cubes making up the maze on which the developer unhappily comments "*The first 10 cubes took 10 min, so I guess that inserting all the cubes will take between 500 and 1,000 min…*". Eventually, the lack of support for repetitive sub tasks in the direct manipulation tool made the developer turn to command language and write a piece of code overriding the development tool and automatically generating the maze. On this he comments, "*The script itself is pretty simple (…) it basically just tells* [*dvMockup*] *where to insert the cubes in the assembly (…). In this case it was definitely an advantage to have some programming experience. It saved me from hours of boring and monotonous work*". In relation to making later changes to the elements and the structure of the maze he continues "*I do not even want to think about how much time this would have taken me manually rather than using my C program*". The developer working with the command language based development tool did not report any problems in relation to repetitive operations. On creating the maze he simply comments "*The maze is modelled using an array of 0 or 1's describing if there is a cube or not. From this it is easy to generate the program code*". Hence, on the issue of

repetition, the developers experience the interaction styles of the two development tools very differently—in favour of command language.

In relation to the second theme of making and discovering errors, both developers describe some issues experienced with their assigned development tool. The developer using the command language tool expresses that the risk of making errors with this approach is rather high because the action space of programming is much larger than with direct manipulation. Furthermore, he expresses that determining errors can be very difficult because it is hard to get an overview of the semantics of a piece of code and its potential side effects. As an example, this developer experienced some trouble with his implementation of collision detection that remained unsolved for a very long time. Eventually the problem turned out to originate in a minor erroneous conversion of coordinates between different coordinate systems due to a small error in the code that the developer had failed to identify. In his final diary entry (for milestone 14) he notes, "*Finally realised that the conversion of coordinates (…) was erroneous and had to be fixed in order to make the collision detection work*". In a similar manner it was very late in the development process before the command language developer discovered that the rendering of his 3D world was actually mirrored due to a similar incorrect transformation of coordinates.

For the developer using the direct manipulation tool, the issue of making and discovering errors was experienced quite differently. First the developer did not report many problems related to making errors in the first place, and when making some, he reports that most of them were easy to discover and fix because they were highly visible in the workspace window of the development tool. As an example of this, when working with creating and placing light sources in the maze, the developer using the direct manipulation tool reported a highly iterative process of simply moving and changing the orientation of different types of light sources directly in the workspace window until a satisfactory setup had been reached. On his iterative experimentation with setting up light sources he comments "*I inserted a single cube into a new 3D world and began experimenting with moving the position and orientation of single light source. This was done for 'ambient', 'directional', and 'point' type of sources, and I now feel that I have a good overview of their individual potentials*". When experimenting with setting up light in the actual maze application he continues "*Tried navigating through the model and experiment with the effect of light sources on the different surfaces (…). Can not seem to make them light up only a delimited area of the maze. When I move it to the middle of the maze all walls are lit and not the actual corridor (…). Tried out different program options for transparencies, materials and textures (…). Turned out that the problem was related to lack of material on objects. After having discovered this it is much easier*".

Hence, on the issue of making and discovering errors, the developers also experience the interaction styles of the two development tools very differently—but this time in favour of direct manipulation.

6 Discussion

Based on the study, we conclude that there was no marked difference between the total time spent on development with the two tools. dvMockup supported fast development on the first milestones related to setting up and testing the development and target platforms. CaveLib supported faster development on the milestones that include the core work on the virtual environment application. dvMockup supported a faster implementation of interaction, but only because the developer could use elements that were already implemented, and this did not entirely fulfil the specification. If the developer should have developed the proper interaction technique, he would have spent more time on that milestone. Thus the study does not support the hypothesis that the direct manipulation tool performed better. Moreover, the command language tool was much faster on the milestones that would be most affected if the scope of the application was scaled up. Thus the development of a larger application might even be less favourable for the direct manipulation tool.

6.1 A successful direct manipulation tool

In order to better understand the limitations of the direct manipulation tool we will compare it with a successful direct manipulation tool, Macromedia Director, on a number of characteristics.

For several years Macromedia Director has been considered state of the art within development tools for interactive multimedia applications targeted at traditional desktop computers with 2D displays. Much like dvMockup, the interaction style in Director is primarily direct manipulation and form filling. However, there are also additional facilities for programming. Director is based on a film and theatre metaphor that puts the developer in the "director's chair". The end-user's screen is represented as a rectangular surface (the stage) on which the developer can place and directly manipulate the different elements that make up the interface: graphics, video, sound, etc. These elements are referred to as the cast of the application, and they are manipulated with the mouse and keyboard. The functionality of an application being developed is defined on a central timeline, called the "score", and in a number of accompanying scripts and

behaviours that are linked to the appropriate elements on the screen.

Around the stage there are a number of tools and fill-in forms for manipulating the elements of the application. Further tools are available through menus or buttons at the top of the screen. The application being developed can rapidly be previewed directly and accurately on the workstation used for development, because display and interaction devices used by the developer are comparable to those of the end-user.

Based on our experiences from teaching Macromedia Director to university students and industrial software developers for several years, we have observed that the direct manipulation interaction style employed in Director performs very well and fits well with the developers' needs during the different phases of the development process.

Because Director and dvMockup appear similar, this observation seems surprising compared to the results of the study presented above. However, a systematic comparison of the two reveals a number of fundamental differences, which may be useful in guiding the design of successful direct-manipulation development tools for virtual environments.

6.2 Creating an application

Applications developed in Director or dvMockup typically consists of a large number of different elements such as graphics, 2D or 3D objects, sound and video files, and scripts/behaviours. When creating a simple application these elements are put together to form a coherent whole, which is then presented to the end-user. Director and dvMockup both provide means for organizing and putting together application elements for this purpose. However, their approaches are fundamentally different in relation to both interface design and support for interaction.

In Director, direct manipulation is used extensively when creating and modifying an application. Every element of the application, for example, images or video, are represented graphically in the cast window (Fig. 6 left), and can be used anywhere in the application by simply dragging them from the cast window on to the preview window (the stage). This action creates a local instance of the element at that given point of the application. These instances can then be modified, for example, scaled or rotated, either in the preview window using direct manipulation or in the timeline window (the score) using fill-in forms. In this way, multiple independent instances can be created and modified at multiple points of the application simply by using the mouse. In the timeline (the score) window the dynamic state of the application over time is represented graphically (Fig. 6 left). The developer can interact with this representation and modify its properties by using direct manipulation actions such as dragging, dropping and scaling or by using form filling.

All phases of creating and modifying a simple application in Director are supported by various and coherent direct manipulation interaction styles. Furthermore, the separation between cast and score allows the developer to concentrate only on the elements in use at one particular time point of the application and ignore the rest.

In dvMockup, direct manipulation is not used as extensively during the creation and modification of a simple application as in Director. The elements of an

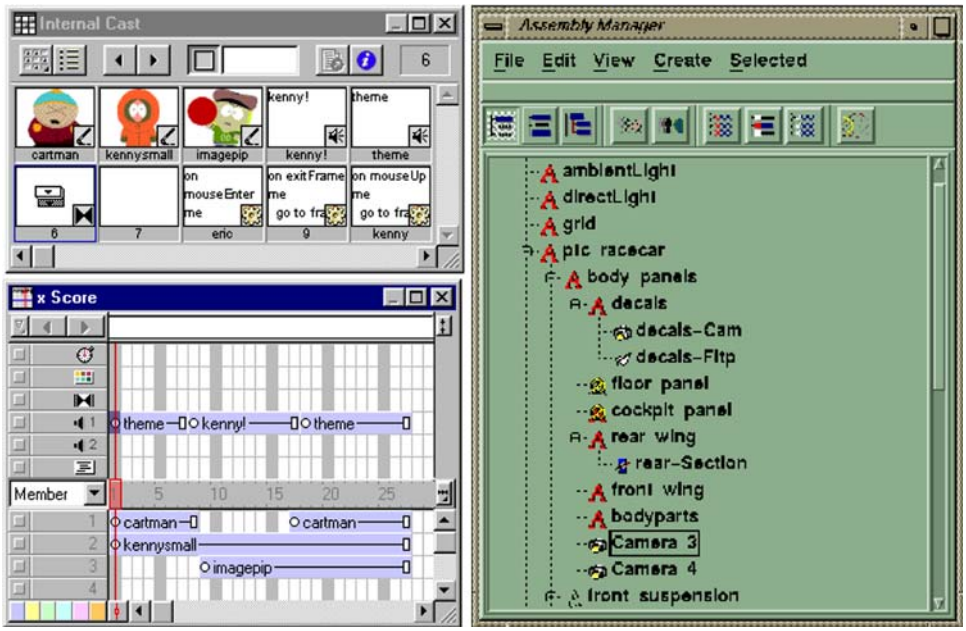

Fig. 6 Cast and Score windows in Director (*left*) and Assembly Manager in dvMockup (*right*)

application developed in dvMockup are grouped hierarchically within a "scene graph", which can be accessed through the assembly manager (Fig. 6 right). The structure of the scene graph can be compared to the structure of the file system on a computer. Every entry in the assembly manager corresponds to a unique element in the application with unique parameters. If the same 3D object is used twice in the application, it will, contrary to Director's cast window, appear twice in the assembly manager. The scene graph facilitates manipulation of whole "branches" of virtual 3D objects without affecting the mutual spatial relationship between the sub-objects in the branch. This is very helpful when working with virtual 3D worlds that can easily consist of more than a thousand 3D objects. Manipulating the scene graph, for example, moving an element from one branch to another or turning the visibility of an element on or off, is done with menu-selection. The developer cannot directly manipulate the layout of the scene graph with actions such as drag and drop.

Creating and modifying a simple application in dvMockup is thus supported by an interaction style that, unlike Director, does not exploit the potentials of direct manipulation much further than simple selection of objects and activation of buttons and menus. The cast window in Director can be considered a central placeholder, whereas the state of the application at a given point in time is reflected in the score that can be manipulated directly. In comparison, the assembly manager in dvMockup acts both as a placeholder and reflects the state of the application. In addition, it supports only a low level of direct manipulation. This makes the assembly manager approach in dvMockup less flexible than the cast and score approach in Director because all elements of the application have to be considered at all times of the application while at the same time having limited means for interaction as a developer. Moreover, there is no distinction between objects and classes. This lack of support for working with multiple instances or objects of the same class contributes to making the scene graph more complex and complicates the interaction.

6.3 Previewing an application

There are two fundamental differences between applications developed in Director and dvMockup. First, applications developed in Director are targeted at desktop 2D displays while applications developed in dvMockup are targeted at immersive 3D displays. Second, applications developed in Director are typically explored screen-by-screen while applications developed in dvMockup constitute 3D worlds, which the user can explore freely. These differences affect the previewing of the application as well as the potentials for direct manipulation in the preview window provided by the two tools.

In Director, the developer is constantly faced with a preview that matches exactly what the user will be presented with at a given time of running the application (Fig. 7 left). This makes previewing and directly manipulating the elements of the application accurate and non-problematic. In the preview of dvMockup, however, the developer can chose to see the virtual 3D world from any perspective wished, without relation to the perspective chosen by the end-user in the Cave (Fig. 7 right). Moreover, the preview in Director matches the number of dimensions used when displaying the final application on a computer monitor while the 2D preview in dvMockup does not match the 3D displaying of the virtual environment in the Cave. This introduces a difference between previewing

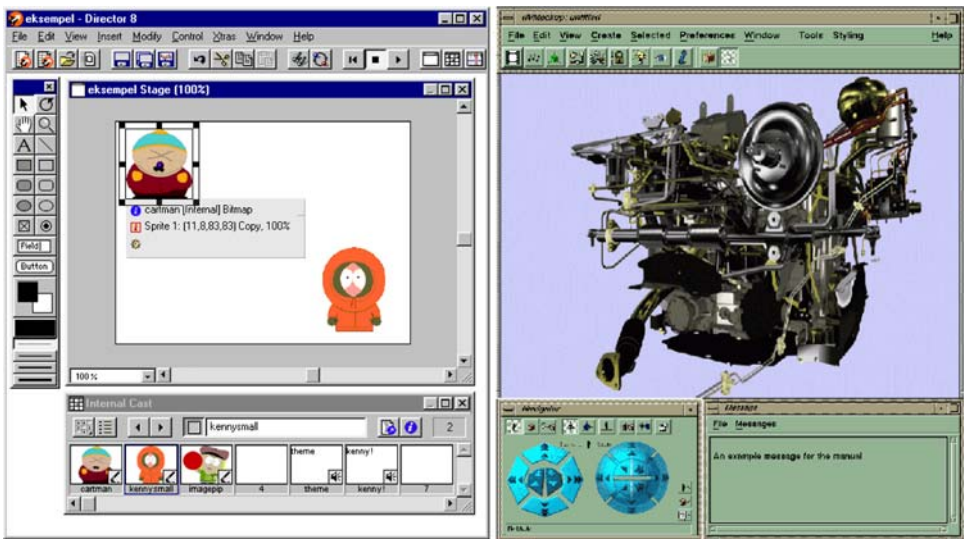

Fig. 7 The Director interface (*left*) and the dvMockup interface (*right*)

and the user's experience of the final application, and the developer is thereby left to imagine how the end-user may experience the application.

Interaction with the preview constitutes yet another difference between the two tools. Whereas interaction with the preview in Director using mouse and keyboard matches the interaction with the final application, interacting with the preview of dvMockup does not. In the Cave, interaction is primarily a question of navigating in a virtual world as well as selecting and manipulating virtual objects. This is typically supported using motion tracking and other 3D interaction devices. However, in the preview of dvMockup the developer cannot interact as if being in the Cave, but is limited to use the mouse and keyboard or dedicated control panels in the interface which the end user will not have access to. If the developers want to experience the application as the end user will see it, they will have to physically get out of their seats, move in to the Cave, and run the application here. This further extends the gap between the preview in dvMockup and the user experience of the final application, and it makes manipulation of the elements in the preview window less direct.

6.4 Programming functionality and interaction

Multimedia and virtual environment applications typically have complex functionality and interaction. Therefore, tools for developing applications must support creating such functionality and interaction. In dvMockup, the user can create simple behaviours consisting of predefined events and actions and relate these to objects in the virtual 3D world. This is done by the use of menu-selection and form filling (Fig. 8 right). It is, however, not possible to extend the functionality of the predefined events and actions in dvMockup by doing "real programming". If additional functionality is desired, the application has to be hooked up to external executables programmed in a traditional programming language such as C or C++, which is quite complicated.

A menu-based tool for quick and easy creation of simple behaviours similar to the one in dvMockup is accessible in Director. Yet Director also provides a tool for command language interaction using high level scripting and object-orientation (Fig. 8 left). With Director, the developer benefits from a seamless transition between the scripting tool, the behaviour tool, and the graphical representation of the application. Observing Director in use by students and industrial developers clearly reveals a continuous iteration between these three approaches at different phases of the development process. The support for advanced programming facilities in Director thus constitutes a significant difference between the two tools.

The main differences between dvMockup and Director are summarized in Table 1.

7 Conclusions

We have conducted a qualitative empirical study of the process of developing a virtual environment application using two tools with different interaction styles. The results of the study indicate that implementation of a simple virtual environment application using a command language tool and a direct manipulation tool required efforts in terms of time that are comparable. The command language tool, however, resulted in faster implementation with the core milestones of the implementation process. Thereby, it seems more promising than the direct manipulation tool on large-scale applications. The direct manipulation tool on the other hand resulted in fewer errors during the process of creating a virtual world. These results do not support the hypothesis that a direct manipulation tool is superior to command language when developing virtual environment applications. Instead it indicate that VE development tools

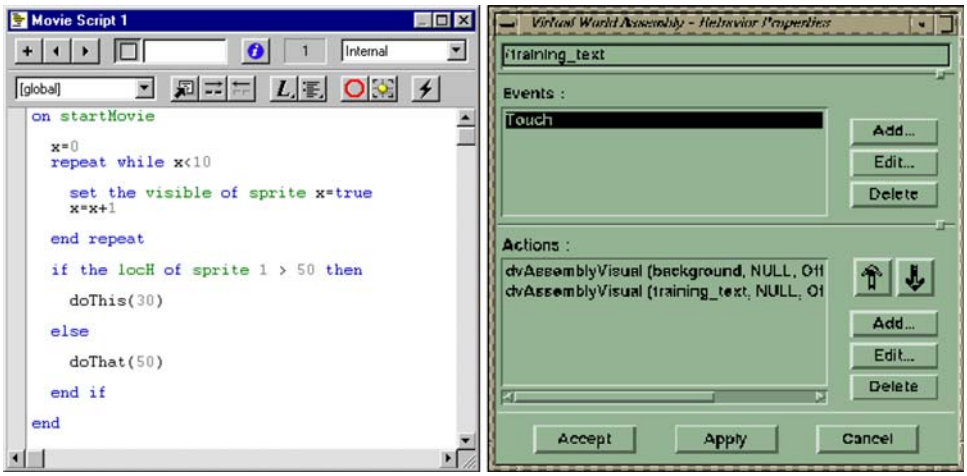

Fig. 8 Command language tool in Director (*left*) and Behaviour tool in dvMockup (*right*)

Table 1 Comparison of Director and dvMockup

	Director	dvMockup
Creating a simple application	The elements of the application are *separated* from the state of the application at a given time	The elements of the application *reflect* the state of the application at a given time
Previewing an application	Previewing *matches* the end-user experience of and interaction with the application	Previewing *does not match* the end-user experience and interaction
Programming functionality and interaction	Use of both predefined events, actions, *and* object-oriented programming	Use of predefined events and actions. No direct support for programming

that allow for both command line programming interaction and direct manipulation may provide the best of both worlds—both from the perspective of addressing the different types of developers as well as the different types of development tasks that teams involved with the development of virtual environments are faced with.

In order to gain a better understanding of the limitations of the specific direct manipulation tool, we have compared it with Macromedia Director, which successfully employs direct manipulation to the development of 2D multimedia applications. This comparison reveals a number of specific issues, which may have negatively influenced the performance of dvMockup. These include (1) the extent to which the potentials of direct manipulation has been exploited, (2) the distance between development platform and target platform, and (3) the support for combining direct manipulation and collections of commands in a programming language. The comparison provides a list of requirements for improving direct manipulation tools for developing virtual environments.

Our conclusions originate from a study that was limited in certain ways. First, by focusing on application development for a six-sided Cave using tools running on desktop workstations we have, of course, taken things to an edge. There is a continuum of virtual reality displays for which the distance between development tool and target platform may be less significant than for the six-sided Cave. Second, the members of the development team were knowledgeable programmers, but not highly experienced in implementing virtual environment applications. In relation to this, there could also have been slight differences between the two developers' skills potentially influencing their use of the tools. Third, the overall task defined a specific application to be implemented, which may not be representative of all classes of virtual environments.

Our study involved the development of the same application using two different tools. A different approach could have been to allow the developers to exploit the interaction mechanisms supported by the tools as best as possible and then compare the usability of the applications that were developed. Such evaluation of usability could be based on the general advice in Sufcliffe and Kaur (2000) and the specific technique suggested in Neale and Nichols (2001). It could also involve more general toolsets of interaction techniques (Willans and Harrison 2001). There are also other development approaches than those compared in our study, for example, procedural development. As a topic for future work, it would be interesting include tools based on other development approaches in the comparison.

A fundamental question emerges from the conclusion. How can direct manipulation be further exploited in tools for developing virtual environment applications? The specific user interface of the development tools may be improved by exploiting the potentials of direct manipulation further with applications such as Macromedia Director, 3D studio Max or Maya as role models. Yet how can we overcome the more basic problem of distance between development and target platform when developing applications for immersive display systems on desktop workstations? A relevant solution might be making direct manipulation more direct as discussed in Beaudouin-Lafon (2000) and Schkolne et al. (2001). As a part of this approach, it would be interesting to conduct a comparative study involving an in-Cave development tool. Hereby, one would be able to compare successful components of 2D development tools, such as the ones presented in this paper, to their immersive 3D counterparts in the Cave.

Acknowledgments The authors would like to thank the development team: Mike H. Hougaard, Nikolaj Kolbe and Flemming N. Larsen. We are also grateful to VR-MediaLab at Aalborg University for granting us access to virtual reality installations and development tools. Figure 2 left (Five-sided VR-CUBE, Chalmers University) appears courtesy of Barco/TAN, used with permission.

Appendix: details of the milestones

1. *Setting up workstation.* This milestone is reached when the development tool and corresponding utilities are installed and configured correctly on the developer's

workstation and a pre-developed demonstration application for the tool can be executed successfully.
2. *Setting up the Cave.* This milestone is reached when the Cave has been set up to successfully execute the pre-developed demonstration application.
3. *Creating a simple visualization.* This milestone is reached when the developer can visualize a single cube by means of his own application.
4. *Setting up light.* This milestone is reached when the cube can be correctly lit by a single light source.
5. *Creating the maze.* This milestone is reached when the application contains a maze as specified in the task description. All cubes must be visualised with the correct colours and lighting but without use of textures.
6. *Simple navigation.* This milestone is reached when the maze is visualized from a moveable point of view functioning as an avatar that can be navigated by means of key presses on the keyboard.
7. *Collision detection.* This milestone is reached when the application is capable of detecting when the avatar collides with the walls of the maze and the exit door.
8. *Collision handling.* This milestone is reached when the application is capable of preventing the avatar from passing through the walls of the maze and ends the game when the avatar reaches the exit.
9. *Setting up 3D input device.* This milestone is reached when the application can read and interpret data from the 3D input device.
10. *Navigation using 3D input device.* This milestone is reached when the avatar can be navigated using the 3D input device.
11. *Setting up motion tracking.* This milestone is reached when the application can read and interpret data from the motion tracker.
12. *Viewpoint using motion tracking.* This milestone is reached when the viewpoint of the avatar can be controlled using the motion tracker.
13. *Textures.* This milestone is met when the all cubes in the maze are correctly visualized with textures.
14. *Calibration.* This milestone is met when the application has been calibrated so that the height of each cube in the maze match the height of physical walls of the Cave, and when the two applications have been calibrated to appear consistently with respect to visualization and user interaction.

References

Allison D, Hodges LF (2000) Virtual reality for education? In: proceedings of VRST 2000, Seoul

Bartz D, Straßer W, Skalej M, Welte D (1999) Interactive exploration of extra-and intracranial blood vessels. In: Proceedings of the conference on visualization '99. ACM, New York

Beaudouin-Lafon M (2000) Instrumental interaction: an interaction model for designing post-wimp user interfaces. CHI Lett 3(1):446–453

Benbasat I, Todd P (1993) An experimental investigation of interface design alternatives: icon vs. text and direct manipulation vs. menus. Int J Man Mach Stud 38:369–402

Bowman DA (1998) Interaction techniques for immersive virtual environments: design, evaluation and application. In: Proceedings of human–computer interaction consortium (HCIC)

Bowman DA, Hodges L (1997) An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In: Proceedings of symposium on interactive 3D graphics, pp 35–38

Bowman DA et al (1998) The virtual venue: user-computer interaction in information-rich virtual environments. Teleoperators Virtual Environ 7(5):478–493

Brooks FP (1999) What's real about virtual reality? IEEE Comput Graph Appl 19(6):16–27

Burdea GC, Coiffet P (2003) Virtual reality technology, 2nd edn. Wiley, London

Conway M, Audia S, Burnette T, Cosgrove D, Christiansen K, Deline R, Durbin J, Gossweiler R, Koga S, Long C, Mallory B, Miale S, Monkaitis K, Patten J, Pierce J, Shochet J, Staack D, Stearns B, Stoakley R, Sturgill C, Viega J, White J, Williams G, Pausch R (2000) Alice: lessons learned from building a 3D system for novices. In: Proceedings of CHI' 2000, The Hauge, ACM, New York

van Dam A, Laidlaw DH, Simpson RM (2002) Experiments in immersive virtual reality for scientific visualization. Comput Graph 26:535–555

Encarnacao LM et al (2000) Seamless 3D interaction for virtual tables, projection planes and caves. In Proceedings of the SPIE AeroSense Conference, Orlando

Hendricks Z, Marsden G, Blake E (2003) A meta-authoring tool for specifying interactions in virtual reality environments. In: Proceedings of the 2nd international conference on computer graphics, virtual reality, visualisation and interaction in Africa, ACM, New York, pp. 171–180

Hougaard MH, Kolbe N, Larsen FN (2001) Comparison of tools for developing virtual reality application (in Danish), Intermedia. Aalborg University

Huang JY, Gau CY (2003) Modelling and designing a low-cost high-fidelity mobile crane simulator. Int J Hum Comput Stud 58(2):151–176

Jepsen LO, Mathiassen L, Nielsen PA (1989) Back to thinking mode: diaries for the management of information system development projects. Behav Inf Technol 8(3):207–217

Kjeldskov J (2001) Combining interaction techniques and display types for virtual reality. In: Proceedings of OzCHI 2001, Edith Cowan University Press, Churchlands, pp 77–83

Keefe DF, Feliz DA, Moscovich T, Laidlaw DH, LaVola JJ Jr (2001) CavePainting: a fully immersive 3D artistic medium and interactive experience. In: Proceedings of Si3D 2001, ACM, New York

Margono S, Shneiderman B (1987) A study of file manipulation by novices using commands vs. direct manipulation. In: Proceedings of the 26th annual technical symposium, ACM, Washington, DC, pp 57–62

Mine M (2003) Towards virtual reality for the masses: 10 years of research at Disney's VR studio. In: Proceedings of the workshop on virtual environments 2003, Zurich, ACM, New York

Morgan K, Morris RL, Gibbs S (1991) When does a mouse become a rat? or … comparing the performance and preferences in direct

manipulation and command line environment. Comput J 34:265–271

Naur P (1983) Program development studies based on diaries. In: Green TR et al (eds) Psychology of computer use. Academic Press, London, pp 159–170

Neale H, Nichols S (2001) Theme-based content analysis: a flexible method for virtual environment evaluation. Int J Hum Comput Stud 55(2):167–189

Pierce J, Stearns B, Pausch R (1999) Voodoo Dools: seamless interaction at multiple scales in virtual environments. In: Proceedings of symposium on interactive 3D graphics 1997, pp 141–145

Poupyrev I, Weghorst S, Billinghurst M, Ichikawa T (1996) The Go–Go interaction technique: non-linear mapping for direct manipulation in VR. In: Proceedings of symposium on user interface software and technology (UIST) 1996, Seattle. ACM, New York, pp 79–80

Poupyrev I, Weghorst S, Billinghurst M, Ichikawa T (1997) A framework and testbed for studying manipulation techniques for immersive VR. In: Proceedings of symposium on virtual reality software and technology (VRST) 1997, Lausanne. ACM, New York, pp 21–28

Poupyrev I, Weghorst S, Fels S (2000) Non-isomorphic 3D rotational techniques. In: Proceedings of CHI 2000

Poupyrev I, Weghorst S, Billinghurst M, Ichikawa T (1998) Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. Comput Graph Forum 17(3):41–52

Scaife M, Rogers Y (2001) Informing the design of a virtual environment to support learning in children. Int J Hum Comput Stud 55(2):115–143

Schkolne S, Pruett M, Schröder P (2001) Surface drawing: creating organic 3D shapes with the hand and tangible tools. CHI Lett 2(1):261–268

Shneiderman B (1998) Designing the user interface: strategies for effective human–computer interaction, 3rd edn. Addison Wesley/Longman, Reading

Smith SP, Harrison MD (2001) Editorial: user centred design and implementation of virtual environments. Int J Hum Comput Stud 55:109–114

Stanney KM, Mollaghasemi M, Reeves L, Breaux R, Graeber DA (2003) Usability engineering of virtual environments (VEs): identifying multiple criteria that drive effective VE system design. Int J Hum Comput Stud 58(4):447–481

Stoakley R, Conway M, Pausch R (1995) Virtual reality on a WIM: interactive worlds in miniature. In: Proceedings of CHI 1995, pp 265–272

Sufcliffe AG, Kaur KD (2000) Evaluating the usability of virtual reality user interfaces. Behav Inf Technol 19(6):415–426

Willans JS, Harrison MD (2001) A toolset supported approach for designing and testing virtual environment interaction techniques. Int J Hum Comput Stud 55:145–165

Zajtchuk R, Satava RM (1997) Medical applications of virtual reality. Commun ACM 40(9):63–64