Interaction Styles in Development Tools for Virtual Reality Applications

Kjeldskov, Jesper; Stage, Jan

Published in:
Production Methods : Behind the Scenes of Virtual Inhabited 3D worlds

Publication date:
2003

Document Version
Early version, also known as pre-print

Published in Halskov K. (ed.) (2003) *Production Methods: Behind the Scenes of Virtual Inhabited 3D Worlds*. Berlin, Springer-Verlag

Interaction Styles in Development Tools for Virtual Reality Applications

Jesper Kjeldskov and Jan Stage
Aalborg University
Department of Computer Science
Fredrik Bajers Vej 7
DK-9220 Aalborg East
Denmark
{jesper,jans}@cs.auc.dk

Jesper Kjeldskov has a background in the humanities with a master's degree in humanistic computer science. He is currently a Ph.D. student working with human-computer interaction for virtual reality and mobile computers.

Jan Stage is associate professor in computer science. He holds a Ph.D. degree in computer science from University of Oslo. His research interests are in usability testing, HCI design, and software engineering.

Virtual reality display systems reflect an emerging technology that is used to visualize virtual three-dimensional (3D) worlds. This article describes and evaluates tools for developing software applications that are targeted at virtual reality display systems. The evaluation focuses on the relevance of command language or direct manipulation with graphical representation as the fundamental interaction style in such a tool. The foundation of the evaluation is an empirical study of two development processes where two tools representing each of these interaction styles was used to develop the same virtual reality application. The development tool that employed a command language was very flexible and facilitated an even distribution of effort and progress over time, but debugging and identification of errors was very difficult. The tool that employed direct manipulation enabled faster implementation of a first prototype but did not facilitate a shorter implementation process as a whole due to e.g. lack of support for writing code for repetitive operations. The limited advantage of using direct manipulation for developing virtual reality applications is further explored through comparison with a successful direct manipulation tool for developing interactive multimedia applications targeted at traditional desktop computers. From this comparison, a number of central issues for user interface design of highly graphical development tools are pointed out for future design.

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.

Virtual Reality display systems are used to create and visualize virtual three-dimensional (3D) worlds. This technology is emerging, and practically relevant applications are being developed for a broad range of domains. As the use of the technology is increasing, there is an increasing demand for tools for developing applications.

The purpose of this article is to compare and discuss the relevance of classical interaction styles for tools that are used to develop virtual reality applications. The focus is on the process of developing actual virtual reality applications. More specifically, we compare the potentials of a development tool based on a command language with one that is based on direct manipulation. The discussion is structured in the following way. First we review selected literature on command language and direct manipulation as classical interaction styles. We then describe the virtual reality display technology in order to specify the target platform we are aiming at. The comparison of the two classical interaction styles is based on an empirical experiment. Hereafter we describe the design of that experiment, and emphasize the relevant results. The empirical findings are then related to the authors' experiences with a successful development tool for traditional 2D multimedia applications. Finally, we conclude the discussion and point out avenues for further work.

1. Interaction Styles for Development Tools

The interaction style is a key determinant of the design of the user interface. Many discussions on the advantage or disadvantage of a certain design relate to this characteristic. The 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 development of a virtual reality application includes an essential task where we construct the 3D world that is visualized by the application. The fundamental interaction style of existing tools that support this task employs either direct manipulation or a command language in the form of a textual programming tool. Menu selection and form filling are also employed but only for secondary interactions that deal with limited issues, e.g. the specification of properties of a certain object that is manipulated directly on an overall level. Based on these priorities, the discussion below deals only with command language and direct manipulation.

The literature on human-computer interaction includes numerous contributions to the question whether direct manipulation is superior to command languages. Much of this is description of advantages whereas the amount of empirical evidence is very limited (Benbasat and Todd 1993). An early contribution compared file manipulation commands in MS-DOS with Macintosh direct manipulation. This study concluded that the Macintosh users could perform the manipulations faster, with fewer errors, and they were more satisfied with the interface (Margono and Shneiderman 1987). A similar study where command line and direct manipulation was compared 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).

The limited number of reports from empirical studies of command language and direct manipulation seem to indicate an advantage in terms of error rate and user satisfaction. When it comes to

time to complete a task, the conclusions are more varied. Our intention in this paper is to examine these expectations in relation to tools for developing virtual reality applications.

2. Virtual Reality Applications

A virtual reality application that visualizes a 3D world consists of a number of mathematically defined 3D models that are covered with colors or textures, e.g. pictures or video images. The 3D models are spatially distributed in a three-dimensional coordinate system that the user can experience as a 3D world by viewing the 3D models from a given point in the coordinate system. The correct perspective is rendered real-time by a graphics computer and projected by means of a display system as illustrated in figure 1. Navigation in the virtual world is accomplished by means of position tracking or a specialized interaction device. Tracking the position of user's head also ensures that the correct visual perspective is calculated. Interaction with objects in the virtual world 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 an illusion of 3D.

Figure 1. A virtual 3D world

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

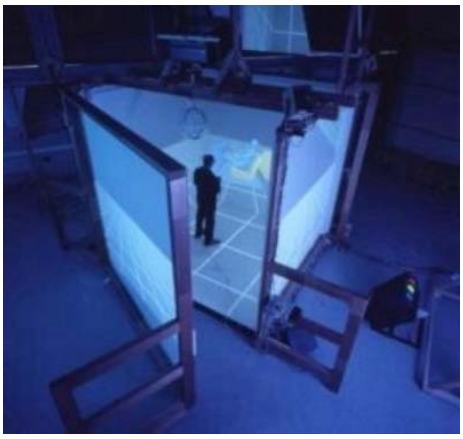

Figure 2. Outside the six-sided cave

Figure 3. Inside the six-sided cave

The benefits of the six-sided cave for exploration of virtual 3D worlds lay in 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 images. Exploration of the virtual world 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 (cf. Kjeldskov 2001).

Virtual reality 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 by default both highly graphical and interactive.

2.1. Two categories of Development Tools

Although virtual reality applications are fundamentally different from typical applications, they are not developed in the cave itself. Virtual 3D worlds are usually developed on ordinary – yet powerful – desktop computers with traditional 2D displays. The existing tools for developing virtual reality applications fall in two different categories.

The first category can be characterized as a classical programming approach, since the creation and manipulation of the virtual world and the objects in it is specified in a command language. Within this approach, special libraries for creating cave applications are available for C and C++. One of the most widely used binary libraries for developing virtual 3D worlds is CaveLib. This 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 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. figure 4.

Using CaveLib to develop an application is not very different from developing any other graphical application in a typical programming language.

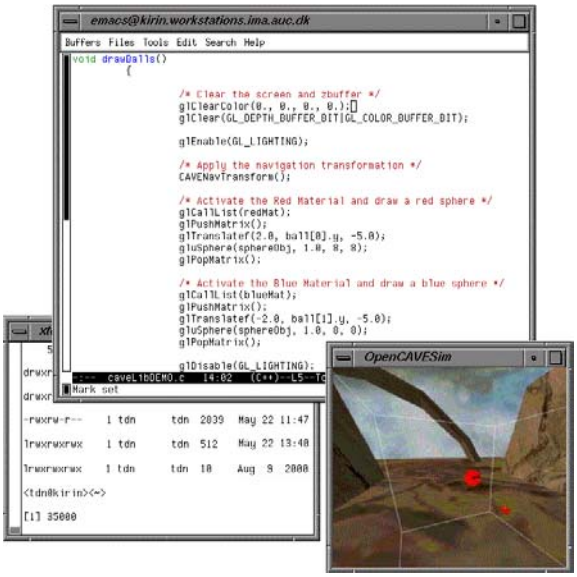

Figure 4. Development with CaveLib

When developing a virtual reality application using a command language tool like CaveLib, the developer constructs a program code pointing at a number of geometry files and specifying the layout of a virtual 3D space as well as the functionality of the application. This code is constructed in a simple text-editor and is typically distributed 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 the preview-tool. If the code contains errors or otherwise needs to be modified, the developer returns to the text-editor.

The second category of tools for developing virtual reality applications can be characterized as a graphical representation and direct manipulation approach. One of the few professional tools in this category is dvMockup.

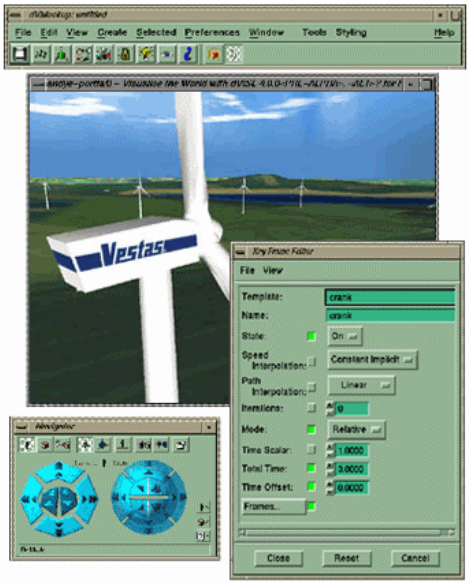

Figure 5. Implementing using dvMockup

This tool enables the developer to create an application by directly manipulating the objects of the virtual 3D world within the preview window along with the use of menu selections and fill-in forms, cf. figure 5. Using dvMockup, implementing an application for the cave is done without doing any actual programming.

When developing a virtual reality 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 behaviors through menu-selection. Through the workspace-window, the developer can continuously see if the application is working properly.

3. Experimental Design

An empirical experiment was conducted to inquire into the research question that was raised in section 1. This section describes the design of that experiment.

Tools: We briefly surveyed potentially relevant tools for implementing virtual reality applications and related this to the fundamental aim of comparing direct manipulation tools with programming tools. Based on the survey and the facilities available, we selected two established tools: dvMockup, a direct manipulation tool that enables people without programming experience to create a virtual reality application, and CaveLib, an advanced programming tool that facilitates development of virtual reality applications characterized by high performance and flexibility. In addition, we selected a new and promising programming tool, VR Juggler, which extends CaveLib with a more flexible and modular structure and open source architecture. The first two tools were already installed, configured, and used extensively by other developers and researchers who could be consulted when technical problems arose. The third tool, VR Juggler, was acquired right before the beginning of the experiment and there were no experiences with it.

Participants: A development team of three persons and the two authors of this article planned and designed the experiment, whereas the development team conducted the implementation phase. The three developers had recently completed a master degree in computer science/computer engineering. Thereby, they had considerable experience and knowledge about programming in general. They had previously taken a one-semester course on computer vision and virtual reality and worked with projects within that subject. They received a one-day introduction to the tools used in the experiment but had no experience with them.

Overall task: The comparison of the three tools was based on solution of the same overall task. The overall task was to develop a virtual reality 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 by dividing it into 14 milestones. Thus, the overall task was solved when all milestones were met. The milestones involved tool and cave set-up (milestone 1 and 2), implementation of a simple application (milestone 3 and 4), implementation of the application visualizing the maze (milestone 5 to 8), interaction techniques to facilitate motion of the avatar (milestone 9 to 12), and adjustment (milestone 13 and 14).

Hypothesis: Based on the literature on interaction styles reviewed in section 2 above, we developed the following hypothesis: The direct manipulation tool is superior to the programming tools in terms of the efforts required to implement the a virtual reality application that is specified by the overall task.

Experimental procedure: When the planning phase of the experiment was complete, the implementation phase started. This phase was planned to last three weeks but was extended with a couple of days because of technical problems. Each member of the development team was assigned one of the three tools and should use it to produce the best possible solution to 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 written by each developer, cf. (Jepsen 1989, Naur 1983). After a day of work on the implementation, each developer used about 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 enforces the use of the checklist and increase consistency. The three diaries amount to a total of 45 pages (Hougaard et al. 2001).

Data analysis: The primary dependent variables were work practice and development effort. Yet, in this article we focus only on development effort. Based on the diaries, we have calculated and compared the efforts spent on completing the different milestones of the overall task. The results of this are presented in the following section.

Limitations: The design of this experiment imposes certain limitations on our results. Firstly, the members of the development team were not highly experienced in implementing virtual reality

applications. Secondly, The overall task defined a specific application that should be implemented. These limitations imply that the results primarily facilitate relative as opposed to absolute conclusions about efforts. Thirdly, the diaries of the three developers were different. In order to handle this they were reviewed after one week of work. The fourth limitation was that two of the tools were established on the development platform whereas the third was not even configured and there was no experience with its use. The developer who worked with this tool ended up spending a considerable amount of effort on issues related to installation and execution. Therefore, we ignore this part of the experiment in our comparison below.

4. Findings

In this section, we present and discuss the findings from the experiment with CaveLib and dvMockup. The developer who used CaveLib was able to meet all milestones, but the navigation technique specified was changed due to usability issues. The developer who used dvMockup was not as successful, since collision detection could not be implemented satisfactory. However, the final solution was acceptable. The development time spent using CaveLib amounts to 42,3 hours, whereas the time spent using dvMockup amounts to 37,8 hours. The total time spent on development with the two tools thus differ only 12%. The distribution of time spent on each milestone does, however, reveal significant differences between the programming and direct manipulation approaches. This distribution is shown in figure 6. Below we will highlight interesting points from this distribution.

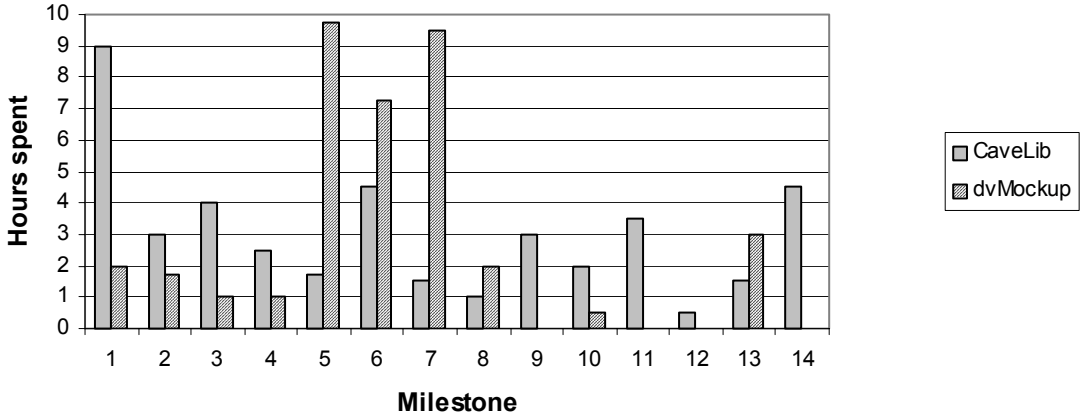

Figure 6. Development time spent using CaveLib and dvMockup.

Setting up the development tools and the cave (milestone 1 and 2) amounted to a total of 12 hours spent on CaveLib whereas only 3,75 hours 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, e.g. 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.

Implementation of a preliminary application with the purpose of testing the development and target platform and the connection between them (milestone 3 and 4) took 6,5 hours using CaveLib but

only 2 hours 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 on the development process.

Implementation of the primary application, which was the maze specified in the overall task (milestone 5 to 8), was done in 10,3 hours using CaveLib. With dvMockup the same milestones required 27,5 hours. 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 the favored tool. Thus the programming 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 program 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, the difference in amount of work will occur precisely on milestone 5 to 8 whereas the remaining milestones will largely be unaffected. Therefore, the difference between the two tools on these milestones will even be more significant if we expand the scale of the application being developed.

Implementation of interaction techniques (milestone 9 to 12) took 7,5 hours with CaveLib while and only 2,5 hours using dvMockup. This is a 30% reduction in favor of dvMockup.

The time spent implementing interaction techniques with dvMockup is, however, influenced by the availability of supporting software. In a related project, a considerable amount of time had recently been spent developing “off-the-shelf support” for implementing interaction in dvMockup in order to facilitate a general reduction of development (Kjeldskov 2001). Had this support not been available, the time spent on these milestones would definitely have increased, but we will not attempt to estimate by how much. 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 select one of the available techniques, which did not fulfill the specification completely. 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 (milestone 13 and 14) took 6 hours for CaveLib while only 3 hours 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 to make the objects fit properly for projections in the cave. This kind of errors was absent in the application developed with dvMockup.

5. A successful direct manipulation tool

The findings described above raises a number of questions. What caused the direct manipulation approach to be outperformed by a command language approach? Is it really fair to conclude that direct manipulation is simply not a well-suited interaction style when developing virtual reality applications? Or what exactly were the limitations of dvMockup?
In order to address this question we will take a closer look at a widely used tool for developing interactive multimedia successfully employing direct-manipulation and compare a number of central characteristics of this tool with the corresponding characteristics of dvMockup.

For several years Macromedia Director has been considered state of the art within development tools for interactive multimedia targeted at traditional desktop computers. Much like dvMockup, the interaction style in Director is primarily direct manipulation and fill-in forms but with facilities for scripting/programming. Director is based on a film/theatre-metaphor putting the designer/developer in the "directors chair". The end-user's screen is represented as a rectangular surface (the stage) on which the designer/developer can place and directly manipulate the different graphical elements: graphics, video, sound etc. (the cast) of the application using the mouse and the keyboard. The functionality of the application developed is defined on a central timeline (score) and in a number of accompanying scripts and behaviors linked to he appropriate elements on the screen. Surrounding the stage is 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, as display and interaction devices used by the designer/developer are comparable to those of the end-user.

Based on our experiences from teaching Macromedia Director to university students and industrial software developers over the last 4 years, we have observed that the direct manipulation interaction style employed Director performs very well and fit with the developers needs during the different phases of the system development process.
But how come that direct manipulation apparently is a successful interaction style when developing desktop multimedia using Macromedia Director while outperformed by command language when developing virtual reality applications?

At first sight Director and dvMockup may seem very much alike. Systematically comparing the two tools, however, reveals a number of fundamental differences.

5.1. Creating a simple application

Applications developed in Director or dvMockup typically consists of a large number of different elements such as graphics, 3D objects, sound/video files and scripts/behaviors. When creating a simple application these elements have to be put together as a coherent whole, which is then presented to the end-user. Director and dvMockup both provides means for organizing and putting together application elements for this purpose. The approaches enforced in the two tools are, however, 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 (e.g. images or video) are represented graphically in the cast window (see figure 7) and can be used anywhere in the application by simply dragging them from the cast window on to the preview window (stage). This action creates a local instance of the element at that given point of the application. These instances can be modified (e.g. scaled or rotated) either in the preview window using direct manipulation or in the timeline (score) window 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 (score) window the dynamic state of the application over time is represented graphically (see figure 7). The developer can interact with this representation and modify its properties using direct manipulation (dragging, dropping and scaling) as well as fill-in forms.

All phases of creating and modifying a simple application in Director are thus supported by various and coherent direct manipulation interaction styles. Due to the separation between cast and score, the developer furthermore only has to concentrate on the elements in use at one particular point of the application, and can leave the rest out of account.

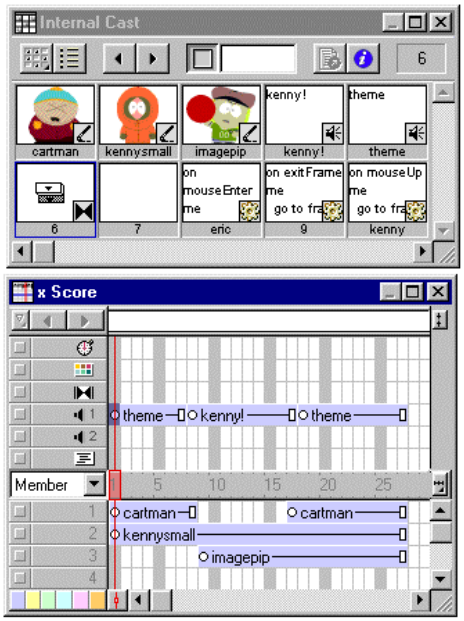

Figure 7. The Cast and Score Windows

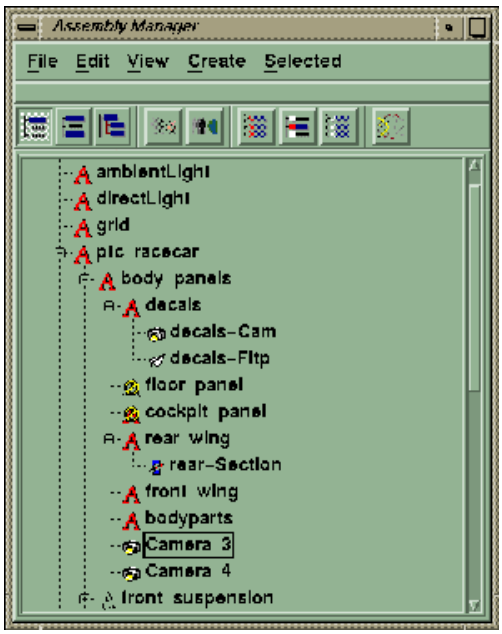

Figure 8. The Assembly Manager

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 application developed in dvMockup are grouped hierarchal within a “scene graph”, which can be accessed through the assembly manager (see figure 8). 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 manipulating whole “branches” of e.g. 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, which can easily consist of more than a thousand 3D objects.
Manipulating the scene graph (e.g. moving an element from one branch to another or turning the visibility of an element on or off) is done by menu-selection. The developer cannot directly manipulate the layout of the scene graph by e.g. drag and drop.

Creating and modifying a simple application in dvMockup is thus supported by an interaction style, which unlike Director does not explore the potentials of direct manipulation much further than simple selection of objects and activation of buttons and menus. Whereas the cast window in Director can be considered a central placeholder with the state of the application at a given time reflected in the highly direct manipulation-able score, the assembly manager in dvMockup acts *both* as a placeholder *and* reflects the state of the application while at the same time supporting only a low level of direct manipulation. This makes the assembly manager approach in dvMockup less flexible than the cast/score approach in Director because *all elements* of the application has to be considered *at all times* of the application while at the same time having limited means for interaction as a developer. The lack of support for working with multiple instances of the same object in dvMockup furthermore contributes to making the scene graph more complex.

5.2. Previewing the application

There are two fundamental differences between applications developed in Director and dvMockup. 1) Applications developed in Director are targeted at desktop 2D displays while applications developed in dvMockup are targeted at immersive 3D displays. 2) 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 affects the previewing of the application as well as the potentials for direct manipulation in the preview window offered by the two tools.
In Director, the developer is *always* faced with a preview that matches *exactly* what the user will be presented with at a given time of the application (see figure 9). 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 (see figure 10). The preview in Director furthermore 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 world in the cave.* The developer is thus left to *imagine* how the end-user may explore the application. This distances previewing from the user experience of the final application

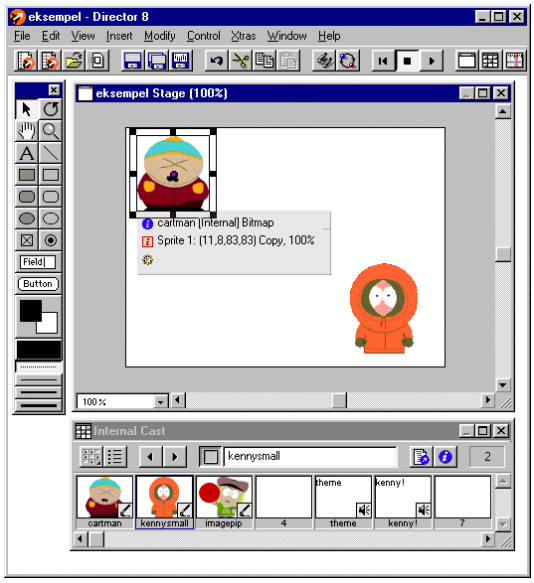

Figure 9. The Director interface

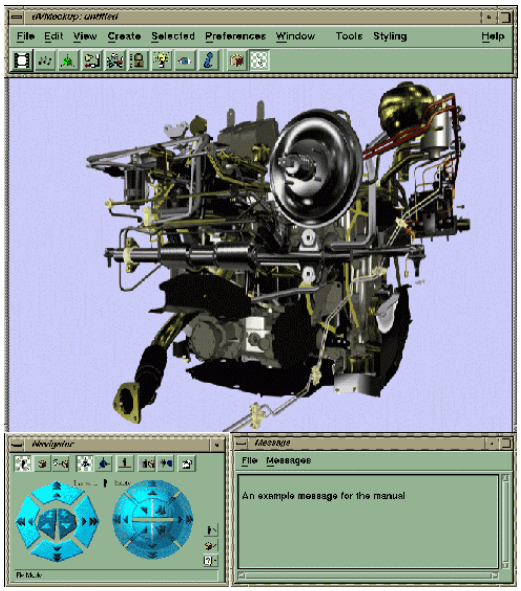

Figure 10. The dvMockup interface

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 a virtual world as well as selecting and manipulating virtual objects. This is typically supported using motion tracking and other 3D interaction devices. In the preview of dvMockup, however, the developer cannot interact as if being in the cave, but is limited to the use the mouse and keyboard or dedicated control panels in the interface which the end user will not have access to. This further extends the gap between the preview in dvMockup and the user experience of the final application and makes directly manipulating the elements in the preview window less direct.

5.3. Programming functionality and interaction

Multimedia and virtual reality applications typically have complex functionality and supports complex interaction. Tools for developing such applications must therefore support programming such functionality and interaction. In dvMockup, the user can create simple behaviors consistent 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 fill-in forms (see figure 12). It is, however, not possible to extend the functionality of the predefined vents 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 e.g. C or C++.

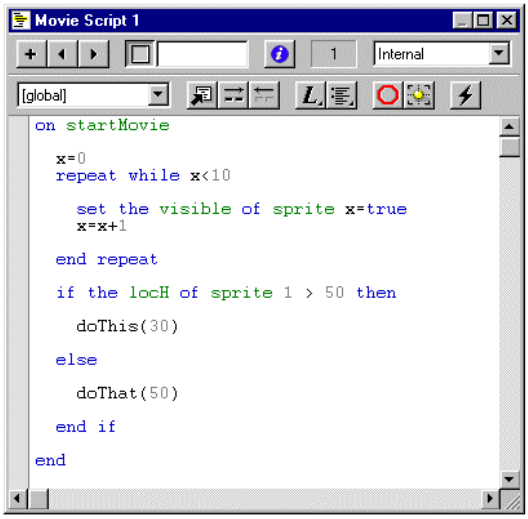

Figure 11. Scripting tool in Director

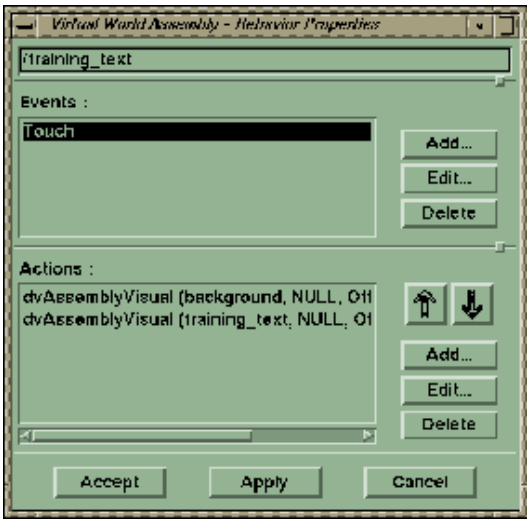

Figure 12. Behavior tool in dvMockup

While a menu-based tool for quick and easy creation of simple behaviors similar to the one in dvMockup is also accessible in Director, the latter furthermore provides a tool for command language interaction using high level scripting and object-orientation (see figure 11). Using Director, the developer benefits from a seamless transition between the scripting tool, the behavior 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 tools at different phases of the development process. The support for advanced programming facilities in Director thus constitutes a significant difference between two tools.

The comparison of Director and dvMockup is summarized in table 1.

	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 the 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.	Exclusively use of predefined events and actions. *No direct support for programming.*

Table 1. Comparing Director and dvMockup.

6. Conclusions

We have conducted a qualitative empirical study showing that implementing a virtual reality 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 during the most essential phases of the implementation process and thus outperforms the direct manipulation tool on a larger scale. The direct manipulation tool on the other hand resulted in fewer errors.

While the empirical results from comparing direct manipulation with command language interaction styles for development of virtual reality applications may suggest that within the virtual reality software development domain, command language is simply superior to direct manipulation, comparing dvMockup with Macromedia Director, which successfully employs direct manipulation, reveals a number of more specific issues, which may have negatively influenced the performance of dvMockup. These includes 1) the level at which the potentials of direct manipulation has been exploited, 2) the distance between development and target platform and 3) the support for switching between direct manipulation and a programming interface.

By focusing on application development for a six-sided cave using tools running on desktop computers we have, of course, taken things to the edge. There is a continuum of virtual reality displays for which the distance between development tool and target platform may not be as significant as for the six-sided cave.

A central question rises from this conclusion. Can direct manipulation be further exploited in tools for developing virtual reality applications? One thing is improving the user interface of development tools and further exploiting the potentials of direct manipulation with applications like Macromedia Director as role models, but how can we overcome the problem of distance between development and target platform? A relevant solution might be making direct manipulation more direct as discussed in Beaudouin-Lafon (2000) and Schkolne et al (2001). Developing virtual reality applications *in* the cave would be an interesting could be an interesting path for further exploration.

Acknowledgements

The authors would like to thank the development team: Mike H. Hougaard, Nikolaj Kolbe and Flemming N. Larsen. We also thank VR-MediaLab for access to virtual reality installations and development tools.

References

Beaudouin-Lafon M (2000) Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. CHI Letters 3(1):446-453

Benbasat I, Todd, P (1993) An Experimental Investigation of Interface Design Alternatives: Icon vs. Text and Direct Manipulation vs. Menus. International Journal of Man-Machine Studies 38:369-402

Hougaard MH, Kolbe N, Larsen FN (2001) Comparison of Tools for Developing Virtual Reality Application (in Danish), Intermedia, Aalborg University

Jepsen LO, Mathiassen L, Nielsen PA (1989) Back to Thinking Mode: Diaries for the Management of Information System Development Projects. Behaviour and Information Technology 8(3):207-217

Kjeldskov J (2001) Combining Interaction Techniques And Display Types For Virtual Reality. In: Smith W, Thomas R, Apperley M (eds.) Proceedings of OzCHI 2001, Edith Cowan University Press, Churchlands, Australia, pp 77-83

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, pp 57-62

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. Computer Journal 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

Schkolne S, Pruett M, Schröder P (2001) Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. CHI Letters 2(1):261-268.

Shneiderman B (1998) Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison Wesley, Longman, Reading, Massachussetts