Towards the Automatic Construction of 3D User Interfaces

Mark Green
School of Creative Media
City University of Hong Kong
smmark@cityu.edu.hk

Abstract

One of the main problems facing the development of 3D user interfaces is the wide range of hardware configurations that must be supported. In addition, many programmers lack experience in developing 3D user interfaces and are often forced to develop most of the interaction techniques and software structure from scratch. This is very different from the situation in 2D user interfaces where there is a standard hardware configuration and a wealth of software support for user interface development. This paper describes Grappl, a software tool that automatically constructs 3D user interfaces that adapt to a wide range of hardware configurations. This tool simplifies the development of small to medium size 3D applications and allows the programmer to concentrate on the application instead of the user interface.

Keywords: 3D User Interfaces

1. Introduction

Virtual Reality is not a new technology, but there are relatively few applications using this technology compared to the standard 2D desktop. In the past, the lack of low cost 3D hardware has been used as the main excuse for this situation. But, it is now possible to have good 3D graphics and at least adequate 3D input on regular desktop computers. The problem now is the lack of software to support the development of 3D user interfaces. Unlike 2D user interfaces, there is no standard library of interaction techniques and design tools for 3D user interface development. Thus, programmers are forced to develop their own interaction techniques and software architectures for the user interface, in addition to dealing with application design.

This problem is complicated by the fact that a wide range of hardware configurations are used for 3D applications. Interaction techniques and user interface structures that work well for one input device may be totally unusable with another. Thus, 3D applications are either forced to support and wide range of hardware configurations or severely limit their potential audience.

This paper presents a prototype system, called Grappl that automatically produces user interfaces for a range of small to medium size 3D applications. At run-time Grappl selects an appropriate set of interaction techniques based on the available hardware devices. Thus, the user interface can automatically adapt to different device configurations without programmer or user input. Since Grappl automatically constructs the user interface programmers can concentrate on application development rather than the user interface.

2. Problem Statement

The development of 3D user interfaces has been hampered by the wide variety of input and output devices that are used in 3D applications. This forces developers to produce user interfaces that are either tuned to a particular device configuration, or generic user interfaces that attempt to support a wide range of device configurations. Most developers have taken the first approach, since it produces better user interfaces, but this can severely limit the number of places where the application can be used. Taking the more generic approach allows the application to run at more locations, but the user interface may be unusable at some of these locations due to the devices that they use. At the present time there is no middle ground that allows developers to produce applications that run at many locations and at the same time have a reasonable level of usability.

The nature of this problem can be illustrated by examining of the types of input and output devices currently used in 3D applications. For position and orientation information that are two main classes of input devices. Absolute devices, such as the Polhemus and Intersense trackers, produce absolute position and orientation values. On the other hand, relative devices, such as 3D joysticks and pucks, measure the relative motion of a controller. The range of absolute devices is limited; therefore techniques must be developed to interact with objects that are outside this range. This isn't a problem with relative devices, since their position values aren't limited. Due to this difference interaction techniques that work well with absolute devices may be unusable with relative devices and vice versa. Applications that use absolute device tend to use some form of pointing to indirectly manipulate objects, while applications using relative devices use a cursor to directly manipulate objects.

Similar problem occur with output devices. In the case of head-mounted displays, the user's view is typically

blocked so the environment surrounding them is not visible, thus they are restricted to using the devices that are currently held in their hands. In the case of desk top and projection based displays this isn't a problem. Due to the reduced resolution of head-mounted displays, information that is legible on desk top and project displays may be illegible on head-mounted displays. Thus, an application using a head-mounted display may arrange information differently and use different techniques to display the information.

There needs to be some way of supporting a wide range of device configurations and providing support for programmers that don't have 3D user interface design skills. Our approach to solving this problem, as outlined in the following sections, is to automatically generate the user interface given the application requirements and the current device configuration.

3. Background

A considerable amount of work has been done on software tools for 2D user interfaces. The most relevant topics for this work are User Interface Management Systems (UIMS) and model based tools. There has been considerably less work done on 3D user interfaces with the most relevant topic being 3D interaction techniques.

A considerable amount of work has been done on software tools for the design and implementation of 2D user interfaces. A good survey and analysis of this work can be found in [4, 8]. Two of the main contributions of this work are libraries of standard interaction techniques and interface builders that provide graphical interfaces for selecting interaction techniques and designing screen layouts. A small number of standard interaction technique libraries are used to construct most commercial 2D user interfaces. The use of standard libraries gives applications a standard look and feel, plus facilitates the learning of new applications. On the down side, the standard look and feel tends to stifle creativity and sometimes results in user interfaces that are less than optimal.

User Interface Management Systems (UIMS) have not been as successful as the previous techniques. The goal of a UIMS is to manage the user interface for an application in the same way that a database manages its data. This naturally leads to the notion of the automatic design of user interfaces, where developers provide a high level description of the user interface and the UIMS automatically provides its implementation [9]. This was further extended to model-based interface development [11] where the user interface designer provides a description of the information required by the application and the tasks the user will perform with the user interface, and the UIMS automatically designs and constructs the user interface.

Since this approach wasn't widely accepted, it is worth examining its short comings. One of the main problems was the perceived value for effort in using these systems. Developers had to learn new languages and then build a description of their user interface. In many cases these descriptions were non-trivial and required some time to develop. For small programs it was much easier to use an interface builder. Developers must be convinced that they will save time by using the tool, otherwise it won't be used. A related problem is this time required to learn the tool.

There has been previous work on interaction technique toolkits for 3D user interfaces [1, 12] and tools for constructing them [6, 10]. Some of the VR software packages also include a set of interaction technique that can be used by application programmers [2, 7]. Many of these interaction technique libraries make assumptions about the hardware configurations that they use. For example, some of them assume a desktop configuration [1], while others assume that the absolute devices, such as trackers are used. Our aim is to support a wide range of device configurations, therefore we have developed own interaction technique library [5]. Most of the previous work has concentrated on individual interaction techniques and hasn't provided support for the construction of complete user interfaces. The approach presented here addresses the problem of developing user interfaces for multiple hardware configurations, which the previous work has not dealt with.

4. The Grappl Approach

Grappl automatically designs and constructs the user interface for an application given the device configuration encountered at run time. In order to do this Grappl must be able to determine the set of available input and output devices, the available interaction techniques and the user interface requirements for the application. Since the user interface is constructed at run time the design process must be reasonably efficient.

Each application must provide a description of its user interface requirements. This includes the information that must be provided by the user and the application functionality that is exposed to the user. To support this Grappl provides a simple API that allows programmers to specify user interface requirements in a succinct way. Programmers are familiar with APIs, so this approach is both familiar and easy to learn. The API allows the programmer to specify the operations supported by the application, the information to be provided by the user and how these operations are invoked. From the programmer's perspective this information is specified in terms of the application procedures that are available to the user interface and the parameters to these procedures. The parameter types are one of the main inputs to the interaction technique selection process.

The programmer can also specify the interaction style to be used with each procedure. The current version of Grappl supports three interaction styles. The first is standard command based interaction where the user

enters a command and its parameters and this information is passed on to the application. In this case the commands are placed on a menu that the user can select from. The second style is mode based interaction. In each mode only a subset of the application's interaction techniques are active and their values are passed to the application on each user interaction. An example of this is a mode where the user can position and orient one of the application objects. Each time the position or orientation of the object is changed the application is immediately informed and the object is immediately updated. The third style is based on continuous interaction where the user can always interact with the interaction technique and the results of the interaction are immediately sent to the application. An example of this style of interaction is object selection, which can always be available to the user. All three interaction styles can be used in the same application.

Grappl must also manage the output generated by the application. A C++ based scene graph is used for this purpose. Each application object is represented by its own scene graph. This scene graph is registered with Grappl when it is created, and Grappl is informed each time it is modified. Grappl can perform a number of operations on the application scene graphs. At the same time that Grappl is allocating space for interaction techniques, it can also allocate space to application objects. The programmer has control over whether Grappl will automatically allocate space for an application object. The list of application objects maintained by Grappl is also used as the basis for object selection operations.

When the application starts, Grappl first determines the set of input and output devices that are available to the application. It then selects an appropriate representation for each input device. Once an interaction technique has been selected for the input device an application property is set to indicate whether relative or absolute devices are being used.

Next Grappl constructs a list of interaction technique requests based on the information provided by the programmer. If at least one of the application operations has the command interaction style a request for a command menu is added to the list. Similarly, if an absolute device is used and the size of the application's 3D space is larger than the device's range a request for a navigation interaction technique is added to the list. Once the request list has been constructed it is processed one entry at a time to select an interaction technique that will meet the request. If an operation has several parameters of a similar type they are grouped together so they will be adjacent to each other in the final user interface.

The final stage in interface design is assigning a portion of the application's 3D space to each of the interaction techniques and application objects. The programmer can use one of the API procedures to specify the size of the application space (a default value is used if one isn't specified). The specification of the display device also contains information used in the layout process, including the size of the display area and the prime area of user focus. This process starts by ranking the objects in order of their importance. In the case of application objects, the programmer can specify the object's importance when it's created. If an importance isn't specified for an object its size is used to compute its importance, with larger objects having a greater importance.

5. Prototype and Examples

The prototype version of Grappl can generate user interfaces for both desktop and projection based displays, and for both relative (joysticks and pucks) and absolute (most 3D trackers) input devices. The same executable can run on all hardware configurations, with a set of system level files used to specify the device configuration. Thus, neither the programmer nor the user needs to worry about whether the application will run with the available hardware resources.

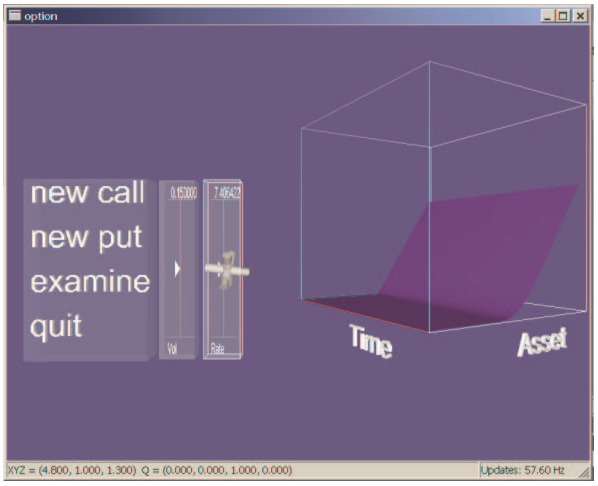

Figure 1. Grappl constructed interface for financial application

Two example applications developed using Grappl are briefly described in this section. These examples are relatively small and are mainly used to illustrate some of Grappl's important features. The first example, shown in figure 1, is a financial visualization application that computes a 3D surface showing option value as a function of time and the underlying stock price. The Grappl constructed user interface allows the user to construct a new option and vary the interest rate and volatility of the underlying stock. The user interface also allows the user to move the surface and change the view angle. The user can create several options and adjust their parameters. Since the user can easily move the surfaces around, it's easy to compare the different options.

Grappl constructs a small menu for this application consisting of commands for creating options and invoking the examination mode. Two graphical potentiometers are selected for changing the values of the interest rate and volatility parameters. The computed surface is examined by first selecting the surface (position the 3D cursor within the surface display and pressing a button) and then moving the cursor to change its position and orientation. The 3D interaction techniques used for these manipulations are not visible in this figure since they have no visual appearance. These interaction techniques are automatically positioned by Grappl. The complete specification of the user interface requires less than 20 lines of C++ code. The user interface produced by Grappl for this application is quite usable and close to what an expert designer would produce.

The second example, shown in figure 2 is a purely 3D user interface that doesn't use any 2D interaction techniques. This application reads a file containing a character model and allows the user to select parts of the model and reposition them through rotation. The user interface specification contains two operations; selecting a part of the model and entering a rotation. The rotation occurs continuously as the user interacts with the input device. Grappl automatically selects a sub-object selection technique and rotation technique and then constructs the user interface. This example shows how Grappl can be used to construct purely 3D user interfaces without any 2D components.

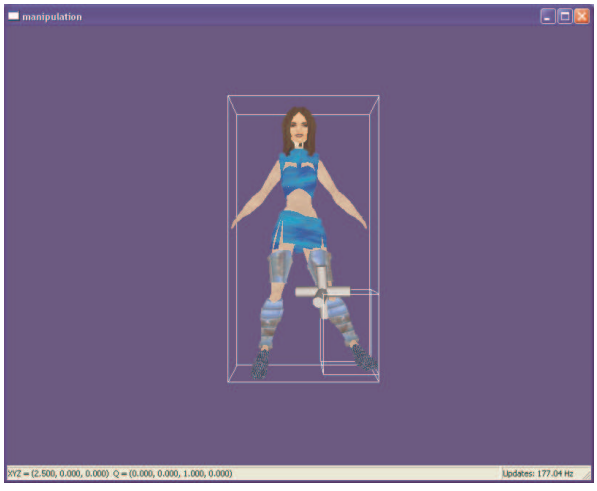

Figure 2. Character model manipulation user interface constructed by Grappl

6. Conclusions and Future Work

The current version of Grappl works best with applications that have a relatively small number of commands (less than a dozen) and concentrate on parameter or object manipulation. For these applications Grappl produces a workable user interface and allows the programmer to concentrate on application issues. It is also fairly easy to modify user interfaces produced using Grappl. At the present time we are working on producing a richer set of interaction techniques, better layout algorithms and more sophisticated techniques for selecting interaction techniques and structuring the user interface. These additions will allow Grappl to handle a wider range of user interfaces.

Acknowledgements

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1159/01E].

References

1. Barrilleaux J., 3D User Interfaces with JAVA 3D, Manning Publications, 2001.
2. Bierbaum A., C. Just, P. Hartling, K. Meinert, A. Baker, C. Cruz-Neira, VR Juggler: A Virtual Platform for Virtual Reality Application Development, IEEE Virtual Reality 2001, 2001.
3. Bowman D., Kruijff E., LaViola J., Poupyrev I., 3D User Interfaces: Theory and Practice, Addison-Wesley, 2005.
4. Green M., A Survey of Three Dialogue Models, ACM Transactions on Graphics, vol.5, no.3, p.244-275, 1986.
5. Green M., J. Lo, The Grappl 3D Interaction Technique Library, VRST'2004 Proceedings, 2004.
6. Jacob, R., Deligiannidis, L., Morrison, A Software Model and Specification Language for Non-WIMP User Interfaces. Transactions on Computer-Human Interaction, 1999. 6(1): pp. 1-46.
7. Kelso J., L. Arsenault, S. Satterfield, R. Kriz, DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments, IEEE Virtual Reality 2002, 2002.
8. Myers B., S. Hudson, R. Pausch, Past, Present and Future of User Interface Software Tools, ACM Trans. On Computer-Human Interaction, vol.7, no.1, p.3-28, 2000.
9. Singh G., M. Green, Automating the Lexical and Syntactic Design of Graphical User Interfaces: The UofA* UIMS, ACM Transaction on Graphics, vol.10, no.3, p.213-254, 1991.
10. Stevens, M., Zeleznik, R., Hughes, J., An architecture for an extensible 3D interface toolkit. Proceedings of UIST'94. 1994. ACM. pp. 59-67.
11. Szekely P., Retrospective and Challenges for Model-Based Interface Development, Design, Specification and Verification of Interactive Systems'96, p.1-27, 1996.
12. Zeleznik, R.C., Herndon, K.P., Robbins, D.C., Huang, N., Meyer, T., *et al.*, An interactive 3D toolkit for constructing 3D widgets. Proceedings of SIGGRAPH'93. 1993. ACM. New York, NY, USA. pp. 81-4.