Virtualiti3D (V3D):
A SYSTEM-INDEPENDENT, REAL TIME-ANIMATED, THREE-DIMENSIONAL GRAPHICAL USER INTERFACE

Stephan Rusdorf, Mario Lorenz, Steven Wölk, Guido Brunnett
Chemnitz University of Technology
Straße der Nationen 62
09107 Chemnitz, Germany
sru@informatik.tu-chemnitz.de

Abstract

This paper describes the creation of a graphical user interface (GUI) based on three-dimensional objects and current mainstream graphics hardware. The goal of this development is real three-dimensional behavior of the individual elements in accordance with current possibilities, as well as a maximum degree of adaptability. The creation and handling of the interface should be as simple as using 2D GUI's common today and allows simple integration into VR applications.

Keywords: Human/Computer interface, Virtual reality, graphical user interface, animation

1. Introduction

Today's graphics hardware offers very high performance supporting the representation of three-dimensional objects and worlds. Unfortunately, these capabilities are currently not used in the field of graphical user interfaces, e.g. windowing systems. Why don't all graphical user interfaces use the features of the graphics hardware? Most common GUI's use a "3D look and feel." Here, the elements are so arranged as if to appear as illuminated three-dimensional objects. It would thus seem obvious to represent these elements as genuine three-dimensional objects. This opens up completely new possibilities in GUI design, both for the developer and the user. In addition to the common characteristics of the user interface, such as character size, border width and background images, it would be possible to define light sources and even animate them. All elements could be freely positioned in three-dimensional space, not only horizontally and vertically but also along the depth axis. In addition, it would also be possible to rotate them around any axis. Furthermore, the fall of shadows would not have to be simulated artificially, but could be calculated in real time.

The most important reason to create and use a universal three dimensional interface is the seamless integration into VR style applications. When the user has to interact with virtual objects in immersive environments a 3D user interface is needed for efficient operation by the user. To popup a 2D window and to ask for user input via mouse or keyboard forces the user to switch between immersion and desktop operation mode and is thus far from being user-friendly. The worst case scenario is a request for a hardware keyboard based input while the user is wearing a HMD.

Therefore a 3D user interface should provide a solution to allow useful interaction within the three dimensional environment.

2. Previous Work

The idea of a three-dimensional user interface is not new. However, approaches up to now can for the most part only be considered feasibility studies, or are limited to a special application for a given hardware. Some libraries provide a certain basic functionality for simple GUIs in games, but not for complex multi-window applications.

In some papers [1][2][3] the idea is pursued to modify the window manager for 3D usage. Here the 2D window is mapped onto a rectangle, which then can be placed everywhere in the three dimensional environment. Unfortunately the appearance of a world generated in this way is not really three dimensional. All windows are flat. Furthermore the X-Windows version [2] seems to be expensive regarding memory bandwidth and/or cpu consumption. Note, the average refresh time is about one second.

Whereas the version for X-Windows pursues the same objective as our interface, i.e. the integration of the GUI into a virtual world, the version for Microsoft Windows has been created only to organize the windows and tasks of the desktop in a modern way [3].

The paper [4] is more closely related to our approach, due to the usage of real 3D widgets. In contrast it seems to be specialized to 3D modeling applications. E.g. it does not support different IO-devices or the "window"-idea known from the 2D interfaces.
3. Concepts

Our goal is thus the creation of a completely usable interface featuring typical modern elements, such as windows and gadgets. In this way, any application can be normally converted to our interface without special adjustments except of using the new API. It’s also possible to support different APIs through a wrapper. It is insignificant whether it will later run on a standard PC, or in conjunction with a beamer, shutter glasses, and a tracking system.

What are the advantages of this approach? It may initially appear not very meaningful to develop a text processing program with a 3D interface. But for many of today’s strongly visually-oriented software programs, such as internet presentations and other multimedia tools, a 3D interface already results in an optical advantage in the case of normal screen usage. It would be possible to take advantage of all capabilities of modern graphics hardware. This permits simple calculation of complex effects in situations in which pre-computation and transfer as 2D data are still prevalent (e.g. flash animations). Furthermore, a good 3D impression can be generated by using shutter glasses. This option is quite inexpensive and thus suited for the mass market.

The second point pertains to the entire range of 3D applications, including professional software and games. Here all information and control elements can be placed directly into the virtual world. A normal monitor environment then uses common 3D visualization, whereas the 3D impression and navigation is much better when using a beamer and/or other 3D devices.

The V3D interface, which is still in development, is based on its own rendering engine, which in turn uses the native OpenGL library to represent objects. Thus, the required platform independence and support of common graphics hardware is achieved. There are purposefully no other graphics libraries used by V3D. In this way, it is possible to avoid restrictions and achieve maximum control of all internal activities.

Both the rendering engine and the V3D module are written completely in C++. V3D offers platform-independent functions for multithreading. The interface and rendering engine also operate multithreaded. All import and export filters for 3D, 2D, and audio formats are directly integrated in the V3D module. Furthermore, a network-transparent plug-in system has been prepared, to load and save custom file formats. There is no need for applications to rely on any other libraries. Despite this scope, the compiled executables are quite small. The largest application currently available is the "CADEditor," with a size of approx. 1 MB on Windows-based systems.

Similar to two-dimensional GUI's, the appearance is divided into windows and gadgets, hereafter referred to as elements. When creating the interface, all elements are stored in a common scene. This means each GUI element is represented by one or more 3D-objects in a virtual scene. In other words, it does not use 2D windows placed on a surface as a texture. V3D instead uses real three-dimensional objects. It is up to the user to decide whether an element consists of simple, rectangular surfaces or complex polygonal models. Assigning a unique ID to all elements, the V3D module and the rendering engine are able to exchange information about the objects. For instance, it is completely irrelevant whether a button is represented by a simple box or a complex polygon model. Once the entire user interface is created, the rendering engine processes the virtual scene and creates an appropriate representation on the output device. This means that the interface scene is simulated like an interactive three-dimensional world. Depending on the settings, the rendering can also be performed in stereo mode. For specific changes, the application can also use the functionality of the rendering engine directly (fig. 1).

Fundamentally, only the rendering engine should call the OpenGL functionality of the underlying hardware, since the entire rendering process is subject to certain conditions. For user-specific needs, two hook functions are supported, allowing additional actions to be executed directly before or after the rendering of a frame.

All characteristics that are necessary for the various GUI elements and that do not belong to the normal object characteristics for 3D objects are controlled by the V3D interface. These include, for instance, the type of an element (window, button, toggle, slider...), certain relationships between elements (slider movement causes modification of the value of a number field), and boundary values of elements (min/max of slider).

![Figure 1: Communication between the different parts of the V3D user interface](image)

To exchange information, a suitable interface between V3D and the rendering engine was developed. This communication mainly serves the querying of the current state of the scene (the currently selected object) or the transmission of events to the scene. When a button is pressed, for example, a corresponding message is sent to the object, which then triggers an appropriate action, such as illuminating or morphing. It does not matter whether this action causes a movement of the object or a modification of its surface. This procedure represents an extension of the event-controlled animation technique, developed at the Chemnitz University of Technology and already implemented in the CADVis project [6] for the...
visualization of CAD worlds. Because this concept was an important part of the development, it is briefly illustrated in the following section.

4. Event-Controlled Animation

Compared to simple keyframe animation, event-controlled animation offers the advantage that time data are relative to a defined modification, and the modification is started when a certain event occurs. For a graphical user interface, this characteristic is absolutely necessary, because modifications can not be anticipated.

For each object, any number of actions can be specified. Each action receives a unique number (task code) for identification. The types of action already supported by the V3D interface include homogeneous and accelerated shifting/rotating/scaling, keyframe animation for a range of object and material properties, morphing, audio output, and the control of various illumination characteristics. It should be specially noted that, as an action, a keyframe animation can be started or stopped at any time.

These actions alone do not result in modifications. Any number of events can be assigned to each object. For this purpose, each object stores a list with all types of events to which it will react (the object event list). Each entry contains an identifier for the event type, as well as the task code of the action to be controlled and the corresponding object. Event types include, for example, action completed, action started, object collision, object selected.

Additionally, the type of control is indicated: start, stop, toggle, reset, reverse, or invert.

4.1. Using event-controlled animation in connection with the V3D interface

For the V3D interface, special types of events were added, such as "window opened" or "switches activated." These are then sent by the V3D interface directly to the scene. All further events or actions are independently processed by the rendering engine. This enables the triggering of entire sequences of actions by a single event. All types of events are determined by integers, which are predefined. Depending on the number of available event and action types, this animation principle can be considered very flexible. These possibilities can be extended even more by adding new types of events and actions.

However, the enormous versatility of this concept is only fully taken advantage of in being able to externally define all actions and events. Without external configuration, standard objects with corresponding characteristics are created for all GUI elements.

Alternatively, external objects with completely different characteristics can also be loaded. For example, a button does not have to be rectangular and light up if it is selected, but can also be in the shape of a car and drive off and honk when selected. All this is possible without a modification in the application. Of course, the area of selection is adapted to the shape of the object and does not remain rectangular. If a button is round, it can only be selected in this shape, and not a few pixels to the left or right. For maximum flexibility, even the entire user interface can be described by an external scene. This then permits interactions between the widest range of GUI elements. For example, it is possible to hide two buttons when a switch has been activated. Collision queries can also be included, in order to enable complex interactions between the individual GUI elements in highly complex animated scenes.

Fundamentally, there are two possibilities of using the user interface. On the one hand, it can be manipulated like a "normal" 2D interface, i.e. it consists of windows and gadgets, whereby windows are usually displayed parallel to the projection level and can be moved and closed as usual. The additional degrees of freedom in the third dimension can already be used here. In this way, a window or a gadget can be moved nearer to the viewer or placed diagonally in space. Alternatively, the user can modify the point of view and navigate between different windows. This then leads to the second possibility of using the interface: all GUI elements can also be directly integrated into an additional virtual world. In this way, a control element can be directly appended to an object to be processed, in order to change its position, for example. The immersion of such an application is clearly superior. Furthermore, the spatial appearance and corresponding navigation really only come into their own when 3D-viewing hardware is used in connection with a stereoscopic representation.

5. Features

5.1. Rendering engine

The rendering engine is platform-independent and currently runs under Windows, HP-UX, and IRIX with OpenGL V1.1 and up. A variety of 3D formats, such as Lightwave [7], 3DStudio, VRML97, PLY, and the CADaVR format [6] developed at the Chemnitz University of Technology can be loaded. This also includes the import of animation data for keyframing, morphing, events, and actions. Additionally the most usual standard features are supported (texturing, lighting, special effects).

Stereoscopic representation is supported in two ways: the use of shutter glasses and the implementation of head-mounted displays. Both modes have already been successfully tested. It should be mentioned that the
representation of virtual worlds using the V3D rendering engine can hold its own against professional scene graph libraries, such as the Sense8 WorldToolKit. In some cases, the performance is even better.

5.2. V3D Interface

Because the interface is based on the rendering engine, all features mentioned previously can be used. A range of standard elements is available. A window serves as a grouping of several gadgets, which can be further broken down into buttons, toggles, sliders, cycles, listviews, shows, values, and group gadgets. Buttons are appropriate for triggering certain actions, and a toggle can be used to switch between two states. With a cycle, the choice between several possibilities can be made. To adjust a variable value, the slider or value gadget can be used. Sliders can also be used for scrolling a listview element, which in turn serves to display lists and, if necessary, select one or more elements from it. Partitioning into certain areas is possible with the group gadgets. In this way, the entire user interface can be clearly laid out.

To create the registers typical today, it is possible to combine group gadgets with toggles or cycles. If entire scenes are to be displayed with the integrated rendering engine, a "show gadget" must be created. The scene is then visualized in this area. String and number gadgets are used to input and output text and alphanumerical values. Defining user-specified drawing and handling functions makes it possible to create new gadget types.

After the elements have been created, one callback function per window is specified. It is called when a message arises. The gadgets or windows are only referenced by the ID's, which must be unique for all elements of an application.

The standard elements are rectangular and feature pre-assigned actions for "activated" or "mouse over element" states (Fig. 2). The possible settings can be subdivided into several modes:

1. Different material properties can be specified for the element groups. This includes materials for the elements themselves, the label, and the internal text belonging to string gadgets.

2. Special replacement objects are defined for each element group. Both a new geometric representation and new dynamics can be specified. Dynamics can include movement, material, and light animations, morphing, and audio output. All the rendering engine's possibilities can be taken advantage of here. With this method, it is also possible to specify a curved path for slider objects. The button then automatically follows the curve. The curve can be specified by a keyframe animation.

3. A complete scene is loaded, including a special replacement object for each element, which can be placed as desired. This mode offers the largest amount of freedom, but is also the most complex. The external configuration file contains the assignment of GUI elements to objects. To freely design the scene, additional objects can be inserted. Interactions between the objects are supported.

In each mode, it is possible to specify different character fonts for all groups of elements or individual elements. Currently, only a special format is supported, in which each character must be present as a separate object. It is planned, however, to support the common 2D font formats.

To support several languages, a concept for external text description was integrated. This enables the selection of any language desired. If a suitable description is found, the text appears in this language; otherwise, the standard language (usually English) is used. The texts are referenced by unique integer ID. Of course, further language files can be subsequently created.

6. Important Aspects

In contrast to common two-dimensional GUI's, in a 3D GUI it is not possible to simply redraw only the affected
rectangular area when an element is updated. When using three-dimensional objects with user-specified appearance and placement relative to each other (animated, transparent,…), this procedure is not possible in this way. This is due to the additional depth co-ordinate, which is stored in the z-buffer. Consequently, an area update would simply overwrite this z-buffer. This would not lead to the desired result, as an object situated in front would be painted over. Complex tests are necessary to determine which objects must be redrawn. Alternatively, the complete virtual scene of the interface can be refreshed with each update request. Despite the correspondingly high demands on the hardware, this is the currently favored method; an update necessitates extensive tests when animating different objects. We are nonetheless working on solutions for selective refresh of the scene representation.

Representing text also becomes more complex in three-dimensional space. For one thing, the representation itself is not optimal, since a compromise must be made between filtering and sharpness. Additionally, difficulties arise in clipping overlapping texts (fig. 4: listview), necessitating techniques such as OpenGL stencil and scissor buffers to avoid undesired overwriting of elements. The 3D space also allows a geometry-based three-dimensional representation of all text elements. This means each letter can be represented by a geometric object, like a polygonal surface. This enables the creation of attractive fonts for rough menus, but is unsuitable for applications with a large number of GUI elements and textual information, since the text quickly becomes unreadable and the large number of polygons slows down the representation.

To circumvent this, the text can by configuration be pre-calculated in a texture. At runtime, only textured squares are drawn. Because each letter is then only a 2D image, these kinds of prepared fonts can no longer be represented three dimensionally. For this reason, this technique is suitable primarily for standard fonts, which mostly exist only in two-dimensional form.

Another innovation lies in navigation. In 2D, windows could be shifted only horizontally and vertically. In 3D, true shift in the depth direction is also possible. Additionally, movement of the camera’s point of view is now possible. New navigation concepts must be developed, in order to optimally use the available input devices. This sub-field has been implemented in only a rudimentary fashion up to now.

### 6.1. Picking

The process of selecting a gadget or window is quite different in a three-dimensional environment. Using a 2D input device like a mouse yields similar control as in 2D. The pointer is projected into the space and the nearest intersected object selected. Typically a gadget action is started when the button is released.

There are two main possibilities while using a 3D input device. Selection can be carried out by projecting a ray along a direction vector from the source position into the scene. This would be the case when using a data glove and pointing with the index finger. We worked with the Motionstar tracking system and the Cyberglove data glove, and discovered that the results with this approach are not exact enough to select a small button. Other hardware may yield better results. For this reason, selection occurs by directly calculating the intersection of the pointing object (e.g. index finger) with all other objects in the scene. The corresponding action is started when the intersection ends (button released). To avoid double selection, an initial condition is specified that no other actual intersection may exist. If two objects are nonetheless selected at the same time, the one with the higher overlapping area is used.

There are again two different methods of using a data glove with object intersection. The hand can be projected into the world in a non-congruent way (fig. 5). It is then possible to see and control the virtual hand. The other more realistic method is to place the real hand directly into the world (fig. 6). Here the behavior is more intuitive, because objects must be touched directly with one’s hand; there is no virtual hand model visible. This technique represents a kind of augmented reality. Additionally, feedback is possible by using the actuators of the Cyberglove.

Picking a text or numerical gadget while using the interface in the immersive mode will result in the appearance of a virtual keyboard. This allows the user to enter data by pressing the keys with the 3D input device.

### 7. Summary

In summary, the advantages of the V3D concept in comparison to a 2D GUI are presented in the following table. The table only refers to the characteristics of the GUI without using additional techniques.
8. Application

The interface is already being used successfully in virtual reality applications at the computer graphics department at the Chemnitz University of Technology. This includes, for example, a program for visualizing and processing three-dimensional virtual worlds. The entire mono- or stereoscopic representation is performed with the integrated V3D rendering engine.

The CADEditor has been successfully tested on Microsoft Windows (PC), IRIX (SGI Octane), and HP-UX (HP Visualize Center II). On the Visualize Center, the interface can use the entire distributed rendering area at the available resolution, e.g., 2560x768 spanned over three systems. It supports the active stereoscopic rendering mode using shutter glasses. Audio output occurs via a network connection to a standard PC running Microsoft Windows. The audio rendering includes 3D support via the DolbyProLogic™ system and inclusion of the Doppler effect for moving audio sources. This feature is a component of the V3D rendering engine. Output to AC3 audio streams may be implemented at some time, but is currently not a priority.

9. Conclusion and Future Work

The software is being tested intensively on the HP Visualize Center II. This system provides a “3D Distributed Single Logical Screen” (DSLS) for distributed representation on several computers. Unfortunately, the rendering of complex animated scenes is slowed down dramatically. For this reason, an internal mode for distributing the representation of a scene and hence the V3D interface is being sought, without using the DSLS. The rendering process can then be adapted more efficiently to the connected computers.

Because some characteristics of the graphical user interface are still specified in the application, the configurability is somewhat limited. The focus of future development lies in transitioning as many characteristics as possible into external control files. This allows maximum flexibility; completely different GUI appearances and behaviors can be achieved through user modification of the external configuration file.

A real 3D cursor for more intuitive navigation during stereoscopic representation has not yet been implemented, because the necessary research on 3D input devices, such as data gloves and tracking systems, is not yet finished.

Data glove usage is somewhat difficult, especially when using the virtual keyboard. It’s necessary to improve selection. At this time the idea is to integrate some kind of gesture detection and context sensitive constraints to avoid accidental selection.

10. References