Virtualiti3D (V3D):
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 configurability. The creation and handling of the interface should be as simple as using 2D GUI's common today.

Keywords
Virtual reality, graphical user interface, animation, interaction

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.

Especially for the menus and control elements in VR applications, a real three-dimensional interface is very helpful, since the GUI elements can be placed directly into the virtual world.

2. CONCEPTS
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; libraries supporting this approach can be found in
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.

3. 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 cannot 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

![Figure 1: Communication between the different parts of the V3D user interface](image1)

![Figure 2: Events and actions](image2)
include, for example, action completed, action started, object collision, object selected.

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

In implementing the interface, the focus was primarily on speed. In each object, action, and event, as well as in the scene, an event mask exists. The event mask depends on the event type. In the other data, the mask is set upon occurrence of an event. If a global event such as "key pressed" occurs, the mask for this event is set in the scene. For local events, the mask is set in the object or the corresponding action (e.g. action started). The query of whether an event occurred takes place on parsing the scene graph. In this process, the masks of the events are simply linked bit-wise with those of the possible sources (action/object/scene). If the results are positive (logical TRUE), the action desired for the event is then controlled. In this way, several events occurring at the same time can be easily queried without having to implement separate data buffering.

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 whole numbers, 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 segues into 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.

4. FEATURES

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 [LWS01a], 3DStudio [WOT02a], VRML97 [VRM02a], PLY [PLY02a], and the CADaVR format [TUC98a] developed at the Chemnitz University of Technology can be loaded. This also includes the import of animation data for keyframing, morphing, events, and actions. Different texture projections, such as planar, cubic, spherical, cylindrical, and UV mapping are supported. These can be combined with different texture types, including color, transparency, diffuse, and specular textures. For the scene representation, OpenGL display lists and vertex arrays are supported. Feedback operations were avoided as much as possible, as they lead to a dramatic decrease in speed in distributed OpenGL environments, such as the HP Visualize Center, due to network load. This includes all glGet*() functions, for example. Operations of this type are therefore calculated directly in the rendering engine. This also has the advantage that these calculations can be performed parallel to the rendering process.

Illumination is supported in accordance with the OpenGL1.1 specification. Furthermore, it is possible to integrate special effects like fog, shadows, reflections, and lens flares into the virtual scene. Not surprisingly, a subsequent decrease in speed must be reckoned with. A particle animation system is already in development and will operate similarly to the ParticleFX module in Lightwave. Likewise, the functionality of the keyframe animations (envelopes) was implemented as in Lightwave. Further information can be found in [LWS01a].

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.

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.

The creation of a graphical user interface can be carried out in two ways. One is by calling functions such as V3D_CreateWindow() and V3D_CreateGadget(). The interface is created at runtime of the application. The other method uses the V3Dbuilder, similar to MFC. The V3D builder of course uses the V3D interface, and enables convenient interface creation on all systems. This includes selecting the appropriate objects, adjusting materials, placing elements in three-dimensional space, and labeling. In both methods of interface creation, an external file is created, allowing later modifications of appearance and behavior. The builder likewise generates the suitable header file containing the unique ID's for all created GUI elements.

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. 3). 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.

The interface can already be operated over a network. It is thus possible to control all GUI elements from a separate computer. For this purpose, the internal message handling of the V3D system was designed with network transparency. The implementation uses a broadcast technique, which means several computers can be connected. Everything done on one computer is automatically carried out on the others. This mode is especially suited for teaching.

5. 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 overdrawn. 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. 5: 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 be 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.

Because of screen resolutions and rasterization methods used on common hardware, some loss of detail may occur. For example, thin lines or hyphens can sometimes be lost in the rasterization step. To minimize this effect, maximum resolution is necessary.

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 subfield has been implemented in only a rudimentary fashion up to now.

6. 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) based on the work of [Rus02a]. 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.

7. FUTURE WORK

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. Experiments in this area are currently being carried out at the Chemnitz University of Technology.

Support for a network mode is already available. Two different development goals are being pursued in parallel here. One goal is a multi-user environment, for which 3D data from the GUI and 3D virtual worlds (the application) must be synchronized. Additionally, the V3D system's message handling could take place in a network-transparent manner. A remote representation of the interface could also be realized without the need for a wide-band network, since little bandwidth is required for the interface's 3D data.

The other goal concerns the distribution of processor load. Because all requests must be processed in real time, it makes sense to distribute the calculations over several computers. For example, the entire audio rendering, along with tracking of the user and/or data glove, can be calculated externally on separate systems, in order not to slow down the rendering process.
The software is being tested intensively on the HP Visualize Center II. This system provides a "3D Distributed Single Logical Screen" (3DSLs) 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 3DSLsD. The rendering process can then be adapted more efficiently to the connected computers.

Referencing network resources using hyperlinks is also not yet realized. The software's suitability for virtual presentations over the internet permits simple development of interactive controls.

The wide range of OpenGL extensions will increasingly be used in the future. This did not make sense up to now, due to the lack of support for these extensions on professional graphics cards and graphics computers. But because of the anticipated improvement in speed and quality of graphics hardware, it now appears necessary. This also allows a much simpler generation of many complex effects. Such effects currently require highly sophisticated algorithms and much greater system resources.

Multithreading should be used more intensively. Especially on graphic computers with several processors and render pipelines, speed could be improved significantly.

8. REFERENCES
[Rus02a] Rusdorf, Animations-Editor für virtuelle Welten im CADaVR-Format, TU-Chemnitz, 2002