VRizer
Using Arbitrary OpenGL Software in the CAVE or other Virtual Environments

DI Florian Berger <florianbe@aec.at>
Ars Electronica Futurelab

5th March 2004

Abstract

We introduce a library, which is able to utilize arbitrary OpenGL software for stereoscopic VR-projections without any binary- or sourcecode-interventions to the original code. This is basically archived by intercepting several system calls and replacing them with own functions to get several views from different perspectives of the same scene out of a continuous frame stream.

Contents

1 Introduction

Game Engines in the CAVE

The use of game engines is currently the most widespread way to generate PC-based, interactive, real-time applications. Numerous games come with high-performance editors as standard equipment and thus give a broad spectrum of users the opportunity to realize their own conceptions in three-dimensional environments. With the help of game editors, a user can create highly complex, effective and dramatic environments with a minimum of effort. The Ars Electronica Futurelab has gone one step further with the development of the "VRizer," and, in doing so, has closed the gap between applications based on game engines and the PC Cluster System ARSBOX. (futurelab.aec.at/ARSBOX).

The software framework enables the user to make any Linux-based OpenGL application compatible with any configuration of the ARSBOX in active and passive stereo mode. (arbitrary screen configurations) To do this, certain OpenGL functions are intercepted and modified in order to obtain two images from different perspectives. The implementation of "off-axis" projection (i.e. one deviating from the main axis) makes it possible to compute the correct environment for the viewer's particular perspective, which, in turn, allows for a considerably higher degree of immersion. The head-tracking in real time works with TrackD by VRCO.

Ever since the Ars Electronica Futurelab was founded, an essential aspect of that facility's R&D agenda has been to come up with hardware and software that facilitates artists' access to extremely flexible, high-performance Virtual Reality systems. Experience gained since the Ars Electronica Center's CAVE was installed in 1996 has shown that there is tremendous interest on the part of artists to get involved in this development. Nevertheless, the projects realized with the original CAVE have required that these artists possess certain relevant programming skills or that they collaborate with specialists who do (the approach taken by the Ars Electronica Futurelab's Artist-in-Residence Program). Moreover, the resulting applications themselves were necessarily captives to the system's cost-intensive hardware, which considerably restricted the presentation of such works. An important milestone in this respect was the shift from the mainframes originally required for immersive VR environments to PC-based systems like the ARSBOX (a Linux PC cluster with GeforceFX graphics board), which radically cut costs.

The engine currently most widespread in the art scene is "Unreal Tournament 2003," which made it the obvious choice for the first "VRizer" applications. In addition to artistic works, Futurelab staffers have also been modifying game-editor-based architectural visualization software and computer games-two closely related types of programs-for immersive virtual environments. Examples were presented publicly for the first time at the 2003 Ars Electronica Festival: CODE - The Language of our Time (www.aec.at/code).

A demo version of VRizer will soon be available for free download at:

`http://www.aec.at/en/futurelab/projects_sub.asp?iProjectID=12290`

2 Technical

2.1 Stereo from a Mono Frame stream

Actually when using 3D software based for example on OpenGL the drawn geometry exists in three dimensional representation up to the very end before it is pixelized and thereby drawn to the framebuffer. So by finding a proper mechanism one can in principle change viewpoints, FOV or other viewing parameters frame by frame (actually this is of course not really restricted to viewing parameters). What we like to have is a stereoscopic view, so in the simplest case we can think of displaying the the same scene once from the left-eye-perspective and the next frame from the right-eye-perspective. All that is to be done is to render these two frames into different memory regions and display them combined as a stereoscopic view. If this is in principle possible, we might have no problem to achieve all kind of known active/passive stereoscopic setups like red/greed, multihead, shutter glasses or HMD-stereo.

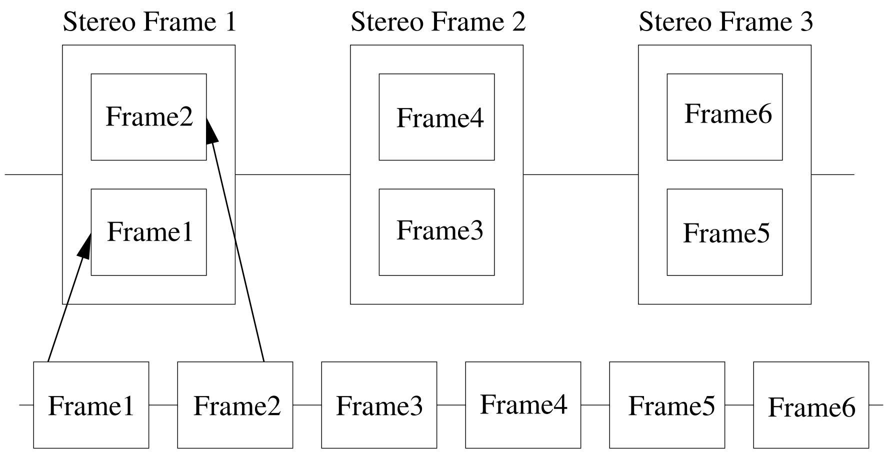

Figure 1: Schematic diagram showing how VRizer arranges mono frames to stereo frames.

We actually also extended the above method to render an arbitrary amount of frames and render them into a 3D-skinned scene created by VRizer.

3 VRizer properties

3.1 What VRizer can do

What VRizer is actually made for, is to have a flexible way to render stereoscopic frames from arbitrary OpenGL software without any need of sourcecode intervention. There are different methods how to set viewports swap buffers etc. VRizer in its current state is able to deal with many different ways of frustum settings and thereby works with a big range of existing OpenGL software. Meanwhile it is becoming more and more mature with every newly tested software. The main point of interest was in particular the Epic-Game UT2003, for which VRizer is tested most excessively and for which it works best in its current state.

VRizer is able to set up stereoscopic projections with either red/green(or blue), multihead output on different regions of the framebuffer for use with polarization glasses, and active stereo also by writing to different framebuffer regions and using softgenlock. Direct stereoscopic use of the GL_LEFT and GL_RIGHT buffers is not yet implemented due to lack of appropriate hardware, but is in principle no problem and will be implemented in near future. Furthermore there is the ability to view whole multi-screen setups in a simulator mode with the ability to load a 3D-skin into the screen setup (see Fig.(3), Fig.(4) and Fig(5)).

3.2 What VRizer cannot do

VRizer cannot (and most likely will never be able to) synchronize distributed applications on a cluster. This ability has to be brought in by th application itself if this is wanted. VRizer has some disadvantages caused by the fact, that there is no way that the application knows about being VRized (at least if not programmed with this in mind). More on these issues in the next section.

3.3 Inherent Problems

There are several major or minor problems with different OpenGL applications. None of them is relevant if you develop your own software, because they can be circumvented by VRizer friendly programming.

Framedelays One very obvious drawback is of course, that VRizer takes two subsequent frames and merges them to one. So on moving environments or objects in a stereo view, the left eye-frame is delayed to the right eye frame by the frametime. This problem can sometimes be overcome by intercepting the proper timer call used by the application and sending new times only every second frame. The simplest way to overcome this is fo course drawing the same frame two times (which reqiures having access to the sourcecode of the application, of course).

Frustum Culling Every software, that is optimized for high performance will only draw geometry, that is visible in the frustum. VRizer now changes the frustum without being able to inform the application about this intervention. So most optimized OpenGL games or programs will suffer from wrong frustum culling under certain extreme configurations. One can archive proper frustum culling by explicitly reading the projection matrix with glGetfv(GL_PROJECTION_MATRIX,...) after having modified it by e.g. glFrustum, gluPerspective, glLoadMatrix, glMultMatrix, etc. and gaining the actual frustum planes out of the VRized matrix.

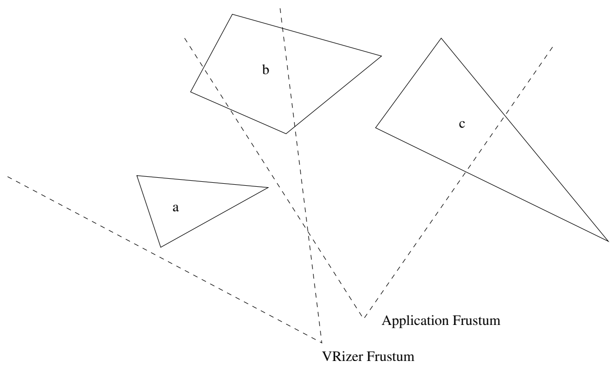

Figure 2: Schematic plot of different application- and VRizer-frustum. Polygon (a) would be culled by the application and never be drawn although supposed to be visible in the VRizer-frustum. Whereas polygon (c) will be drawn although not visible in the VRizer Frustum.

Billboards Billboards are always drawn perpendicular to the viewer (or parallel to the screen plane), which can again be a problem when the real viewpoint/direction (frustum) is different to the one the application assumes. Under extreme conditions this can happen when using VRizer. One can overcome this in a very similar way to the problems with Frustum culling.

4 Eye Candy

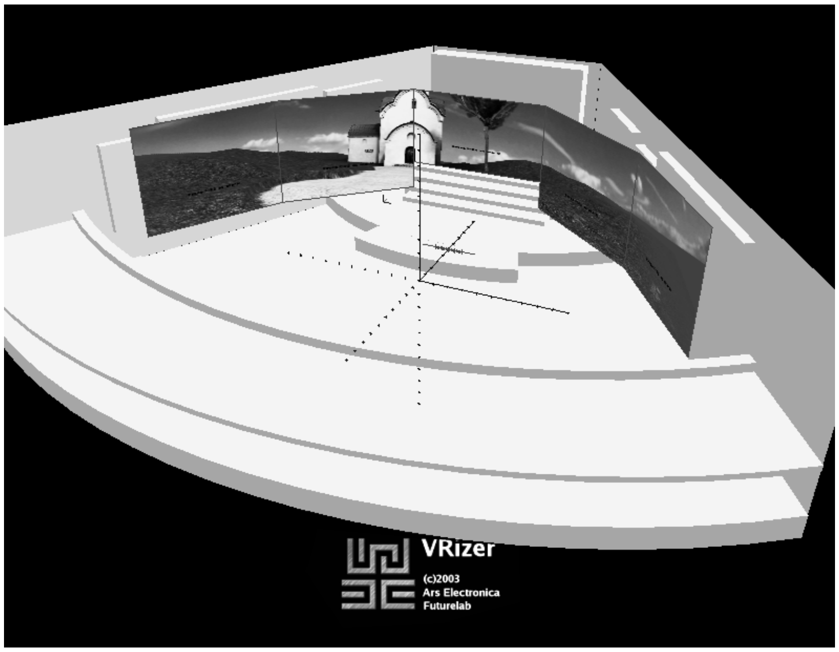

Figure 3: VRizer in Simulator mode with a "3D-Skin" of a multiscreen setup in Brucknerhaus (running on UT2003).

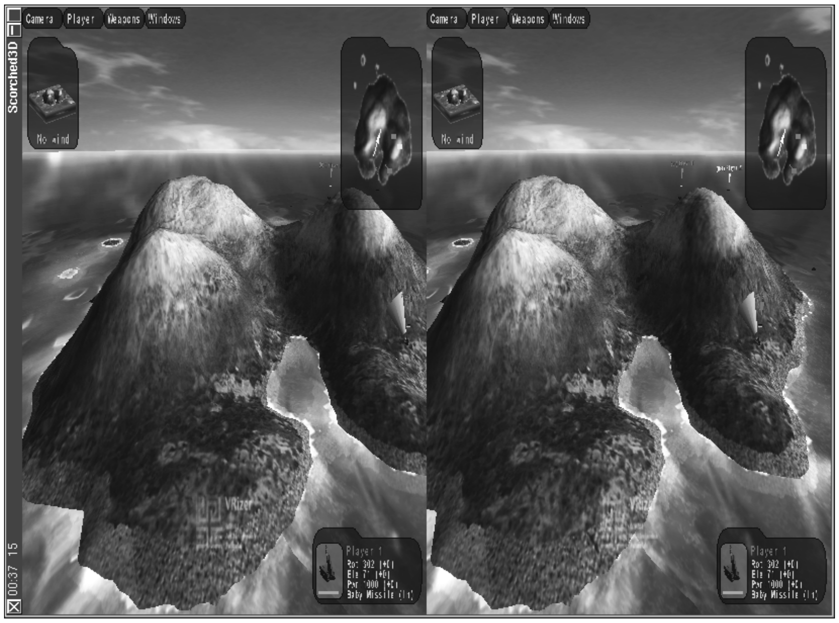

Figure 5: VRizer used on an open source game named “Scorched3D”

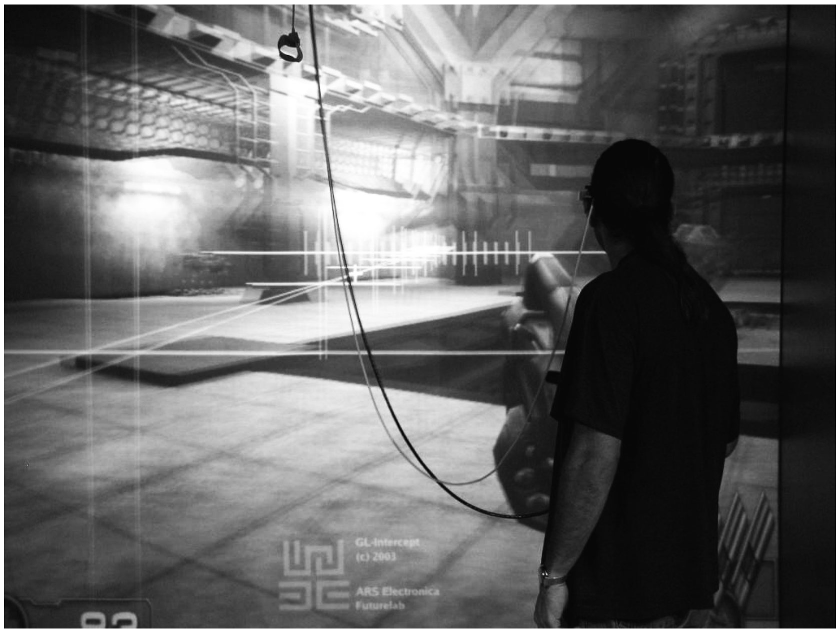

Figure 4: VRizer showing UT2003 in the ARSBOX on four walls with head tracking, wand bindings and softgenlock