

# A Case Study On The Applications Of A Generic Library For Low-Cost Polychromatic Passive Stereo

Simon Stegmaier

Dirk Rose

Thomas Ertl

Visualization and Interactive Systems Group, University of Stuttgart, Germany\*

## ABSTRACT

Active stereo has been used by engineers and industrial designers for several years to enhance the perception of computer generated three-dimensional images. Unfortunately, active stereo requires specialized hardware. Therefore, as ubiquitous computing and teleworking gain importance, using active stereo becomes a problem. The goal of this case study is to examine the concept of a generic library for polychromatic passive stereo to make stereo vision available everywhere.

**CR Categories:** I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms

**Keywords:** Stereo Graphics, OpenGL, Preloading

## 1 INTRODUCTION

The use of stereoscopic techniques to enhance perception dates back to 1832, when Charles Wheatstone proved that the effect of three-dimensional vision could be reproduced by placing two slightly different drawings side-by-side and looking at them through a system of mirrors and prisms.

More than a hundred years later, in the late 1960s, stereo graphics found its way into computer graphics. At that time, high-quality stereo was only available in environments that could bear the high costs of stereo equipment. A constant increase in the performance of consumer graphics boards during the last years has changed this and active stereo using a single projection device and shutter glasses is now a standard technology that is affordable for everyone. Still, active stereo requires sophisticated technology that is not available for every workplace, especially portable display devices. However, ubiquitous computing is gaining importance, so solutions are required that enable also these devices to display stereoscopic images.

This case study discusses the concept of using a generic low-cost software solution to solve this problem. The solution is provided by means of a preload library which enables most OpenGL based applications to display high-quality polychromatic passive stereo. With the presented solution no modifications of any kind are necessary to convert a monoscopic application to a stereoscopic application. This allows us to include software packages in our study, which we do not even have source code for.

Our case study is organized as follows: Section 2 compares the different techniques for generating stereo images using examples of software and hardware solutions developed by other researchers and justifies our interest in passive stereo. Section 3 gives an introduction to the technique of preloading that is used for our library.

This knowledge is indispensable for understanding the description of the library's architecture following in Section 4. Finally, results and experiences with sample OpenGL programs and real-world commercial software packages for flow visualization and finite element crash simulations are reported in Section 5.

## 2 RELATED WORK

We evaluated several stereoscopic techniques, which have been developed previously. It turned out that most of them fail to meet some or all of our criteria: no need for specialized hardware, multi-color images, and portability.

Solutions for active stereo, also used in high-end VR environments, are mostly hardware based and require special drivers [6]. Some software tricks [1] allow to circumvent the need for special drivers, providing support for a minor subset of shutter glasses. In any case, synchronization problems have to be dealt with or active stereo does not work at all. In addition, there is currently no Pocket PC that provides the hardware capabilities required for active stereo.

On the other hand, passive stereo using filter glasses works on all systems. One common technique is based on polarized light [4]. However, polarized light would contradict our requirement for a pervasive usage and, therefore, was dropped.

Another solution is *ChromaDepth* [2], but this is difficult to implement without having access to the source code of the application and requires the altering of the scene's colors. Therefore, *ChromaDepth*, too, is no alternative for our case study.

On the contrary, *anaglyphs* can be generated without having source code access. Formally, anaglyphs can be defined as follows: Given a color model with a set of primaries  $\mathcal{P}$ , the images for the left and right eye are rendered using subsets  $\mathcal{L} \subset \mathcal{P}$  and  $\mathcal{R} \subset \mathcal{P}$  of primaries, with  $\mathcal{L} \cap \mathcal{R} = \emptyset$ . Informally, this just means that the stereo pair can be separated using glasses of different color. However, not all subsets of  $\mathcal{P}$  are suitable for stereographic viewing. If  $\mathcal{L} \cup \mathcal{R} \neq \mathcal{P}$ , the application's color space cannot be reproduced in a satisfying way. For example, using the RGB color model and red-green glasses, no blue tones can be perceived by the viewer.

Yet, this limitation does not apply to colored anaglyphs using e.g. red-cyan glasses. In addition, since anaglyphs can be implemented efficiently using OpenGL's `glColorMask` functionality, colored anaglyphs are an stereoscopic technique ideally suitable for our needs.

As a matter of fact, another anaglyph-based effort [6] similar to ours has already been made. However, this implementation proved to be unreliable and did not support many applications. Moreover, since a stereo library can be implemented with a few hundred lines of code and integration in already existing code was under consideration, we decided to develop an easy to use library on our own that allows us to experiment with different types of passive stereo techniques.

\*University of Stuttgart, IfI, Department VIS,  
Breitwiesenstrasse 20–22, D–70565 Stuttgart, Germany,  
Email: (stegmaier|rose|ertl)@informatik.uni-stuttgart.de

### 3 PRELOADING LIBRARIES

When an application is being developed, functionality implemented in some library may be included using *static* or *dynamic linking* [5]. Using static linking the library’s functionality is copied into the executable. The application is, therefore, unaffected if the archive is replaced later. On the other hand, dynamic linking adds references to a shared library to the application’s executable instead of the library’s code itself. This allows an application to use always the latest version of a library.

Most computer systems allow the programmer to modify the behavior of shared libraries without having to modify the libraries themselves. This technique is called *preloading* and works by implementing a shared library that selectively overrides the interface of another shared library. Using *runtime linking* [5, 7] the original library’s functionality is still available as long as both the original and the replacement library are dynamically linked to the application. Since the executable does only contain references to the original library, the order to preload the replacement library must be given to the linker using a second mechanism. Most systems choose to use environment variables for this purpose.

OpenGL is usually implemented both as a static as well as a shared library. Because most applications are linked dynamically to take advantage of improved versions of the OpenGL library, preloading can be used to modify the behavior of most OpenGL-based applications without having to modify any source code. The library described in this case study takes advantage of this fact and adds stereo capabilities to an application solely using preloading. A detailed description of what functionality of the OpenGL library was replaced is given in the following section.

## 4 LIBRARY ARCHITECTURE

### 4.1 Stereo Rendering

Basically, there are two possibilities to adjust the camera for stereographic perspective. The “toe-in” method (Figure 1(a)) rotates the camera frusta around the center of projection. This approach is geometrically incorrect, and leads to variable misalignments, namely keystoneing or vertical parallaxes between homologous points [4]. The correct approach, implemented in our library, is shown in Figure 1(b). This method uses asymmetric frusta with parallel axis vectors (indicated by the arrows). As we do not know the cam-

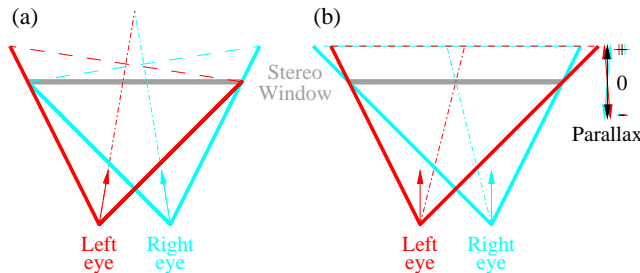


Figure 1: Stereo camera perspectives: (a) approximating “toe-in” method, (b) correct, asymmetric frusta.

era parameters given inside the preloaded application, we have to extract the information needed to calculate the camera properties for the left and right eye from the existing projection matrix  $\mathbf{P}$ . In

general,  $\mathbf{P}$  is of the following form:

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & c_x \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & c_y \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \quad (1)$$

Hereby,  $n$ ,  $f$ ,  $l$ ,  $r$ ,  $t$ , and  $b$  represent the *near*, *far*, *left*, *right*, *top*, and *bottom clipping planes*, respectively. The eye offset in screen space is given by  $c_x$  and  $c_y$ . The *near* and *far clipping planes* can be derived easily from the third row of this matrix. Furthermore, from the first row we determine the *left* and *right* planes which in general are symmetrical to the viewing direction.

Using this information we are able to adjust the asymmetry and the camera offset for the left and right eye by manipulating the third and fourth entry of the matrix  $\mathbf{P}$ ’s first row, respectively.

What is left is identifying the OpenGL calls that manipulate the projection matrix. We have found that `glFrustum` and all `glLoadMatrix*` and `glMultMatrix*` OpenGL calls must be overridden. To handle the unexpected case that `gluPerspective` does *not* call `glMultMatrix*`, we have overridden this utility function, too.

### 4.2 Redrawing

Calculating the viewing frusta for the left and right image is just one part of the process in generating stereo images. The other task is to perform the actual rendering using the determined settings.

The obvious solution is to render the scene once using the settings for the left eye, and then to re-render it with the settings for the right eye. Obviously, this approach requires knowledge about the routine performing the rendering. This is trivial for a *glut*-based application: Overriding `glutDisplayFunc` will yield the desired information. However, in the general case invocation of *glut* functions cannot be assumed. But if the redraw function cannot be determined in a safe way, any attempt to find the redraw function, e.g. by calculating the return address of `glClear` and tracing the callstack back to the first enclosing function, must be considered a heuristic that will not work for many applications. A simple OpenGL pseudocode that does all rendering within `main` demonstrates the problem:

```
int main(void) {
    int i;

    /* Open display and create window */
    ...

    /* Clear the drawable */
    glClear(...);

    i = 0;

    for (i = 0; i < 10; i++) {
        redraw:
        /* Draw some primitives */
        ...
    }

    return 0;
}
```

The correct address for redrawing is labeled `redraw` in this example. Obviously, neither the function enclosing `glClear` (`main`) nor the return address of `glClear` can be used for redrawing without altering the program logic.

Another approach is to record all issued OpenGL commands when rendering the left-eye scene and playing them back for rendering the right-eye scene. This approach requires preloading *all* OpenGL calls. Doing this sounds worse than it really is because most of the code can be generated automatically by parsing the

OpenGL sample implementation specification files provided by SGI [10]. Still, preloading the complete OpenGL library is a challenging task and may slow down the application considerably when large vertex arrays must be saved for further reference.

Creating a second GLX context will also require a preload of all OpenGL functions. In contrast to the previous approach the arguments need no longer to be saved which makes the code much simpler. The disadvantage is that a naive implementation will cause a large number of context switches and therefore will again lower the application's performance. Switching contexts only with `glEnd` and recording calls issued in a `glBegin/glEnd` block will be much better. But even this approach will not give the correct results if the application changes its internal state between two `glXSwapBuffers`.

Given these facts, our opinion is that simply sending an `Expose` event<sup>1</sup> is the best tradeoff between complexity and universality [8]. The application may still change its internal state between two buffer swaps but this should be considered rather a design flaw than good programming style. To conclude: `Expose` events will give identical results compared to the last (and even the first) approach at a much lower cost.

Using this approach, generation of the stereo image requires preloading `glXSwapBuffers` only and discarding the swap after the left-eye view has been rendered. At this point the projection matrix for the right-eye view is set and an `Expose` event is generated. As soon as `glXSwapBuffers` is reentered the event-activated rendering can be considered to be complete and buffers are swapped. Since `glClear` is probably called between two buffer swaps this routine has to be overridden, too, to unmask the color buffer bit when the right-eye view is to be rendered.

### 4.3 Configuration

The computed stereo pairs are controlled by three major parameters: the viewers eye separation, the display width, and the location of the zero-parallax. The former two parameters usually do not need to be changed frequently. However, the last parameter should be adjustable interactively to focus on different parts of the scene. Therefore, we have decided not to save the settings in a configuration file, but instead in a permanent shared memory segment. The segment is created at the application's first start and is initialized with default values. If these are not suitable, a standalone graphical configuration program can be used to attach to the shared memory segment and to adjust the settings. All this can be done without having to restart the application and the settings' effects become visible instantly.

Another adjustable parameter is color saturation, motivated by the fact that anaglyphs cannot reproduce the RGB color space (see Section 2). This drawback can be alleviated by adding white to colors like pure red or green, i.e. by lowering the color's saturation. Still, there is no need to lower the saturation  $S$  in general, therefore, the following weighting function based on the luminances  $L_{red/cyan}$  is being used to compute the adjusted saturation  $S'$ :

$$S' = S \cdot \left( 1 - (1 - S_{max}) \cdot \left| \frac{L_{red} - L_{cyan}}{L_{overall}} \right| \right), \quad (2)$$

## 5 APPLICATION RESULTS

Using passive stereo, each stereo image is the result of a superposition of two images using two slightly different viewing frusta. Let  $R$  be the time between two buffer swaps,  $S$  be the time needed for the buffer swap itself, and  $fps_{stereo}$  and  $fps_{mono}$  be the application's

<sup>1</sup>or the corresponding event that tells the application that the window must be redrawn after being obscured if not using the X Window System

frame rate with and without using the library, respectively. Then the following two relations are expected:

$$\lim_{\frac{R}{S} \rightarrow 0} \frac{fps_{mono}}{fps_{stereo}} = 1, \quad \lim_{\frac{R}{S} \rightarrow \infty} \frac{fps_{mono}}{fps_{stereo}} = 2. \quad (3)$$

Both of these equations can be confirmed using the standard `glxgears` OpenGL application and drawables of different sizes and various numbers of gear teeth. Therefore, geometry-bound applications will experience half the frame rate in the worst case, but typically only a decrease of some ten percent when using large drawables. On the other hand, rendering scenes with little geometry will typically be almost unaffected by preloading the stereo library since buffer swaps are ignored for the left eye and no color buffers are cleared for the right eye.

A visual evaluation of the library was done using real-world commercial software packages for the Linux operating system. The first package, *PowerVIZ* [9] is a *Qt*-based flow visualization tool distributed by Exa Corporation that is used by car manufacturers for visualizing the air flow around car bodies as well as within catalysts. *PowerVIZ* makes extensive use of textures. The second package is *FEMod* [3], an interactive preprocessing tool for car crash simulations. *FEMod* is developed by science+computing AG in cooperation with the BMW AG, Munich and is also based on the *Qt* toolkit.

Our library is implemented on the lowest level of OpenGL. Therefore, any OpenGL-based application is expected to work with the library, whether it uses *glut* or any other toolkit that provides GLX drawables. As expected, both *PowerVIZ* and *FEMod* can be used with the library without any problems; in particular, there were no problems caused by interference of the stereo and the *Qt* library. Depth impression is excellent and most colors are crisp (see Sec-

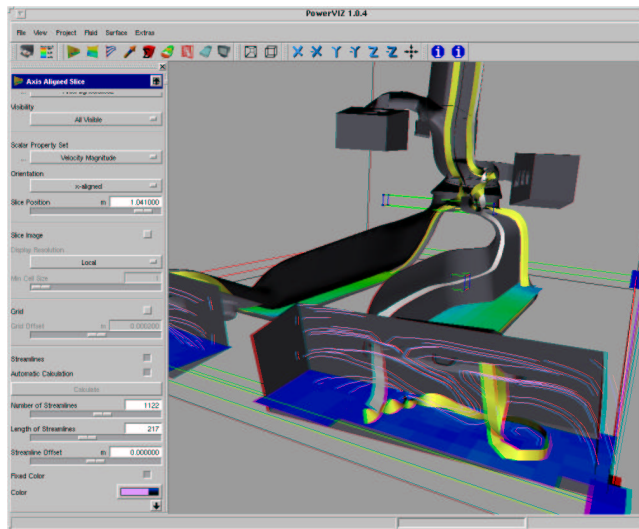


Figure 2: Flow visualization with *PowerVIZ*. The eye distance has been enlarged to emphasize parallax effects.

tion 6 for details). Figures 2 and 3 show some screenshots. The screenshots must be viewed with red-cyan glasses, however, ghosting artifacts may occur due to intentionally enlarged eye distances.

Use of our library is not restricted to desktop computers. Some modern Pocket PCs are able to run the Linux operating system and are, therefore, potential display devices. However, due to a still limited computing performance of these devices, combining our library with a remote visualization library is a more sensible decision at the moment. In [11] remote visualization on a Compaq iPAQ

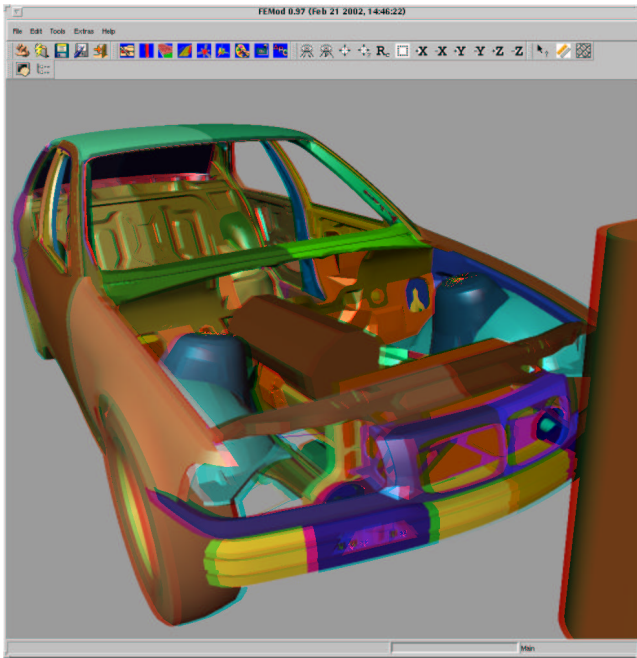


Figure 3: FEMod with enlarged eye distance. As can be seen at the front and rear wheel the zeroparallax is located near the car body's center.

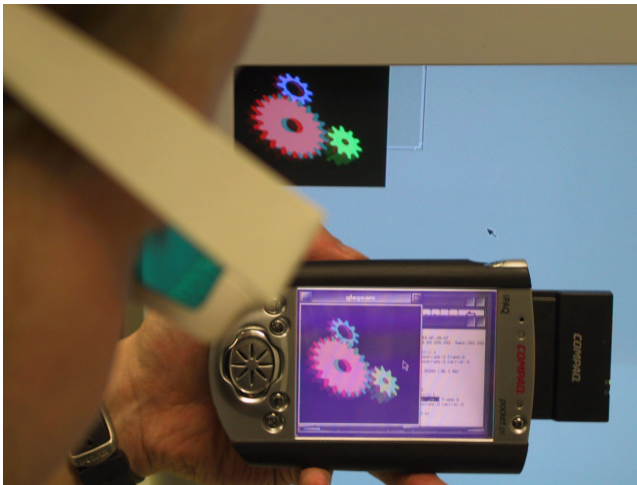


Figure 4: Remote stereo on an iPAQ Pocket PC.

Pocket PC is demonstrated. Clearly, an iPAQ is not suitable for active stereo. However, passive stereo is possible by preloading our library to the visualization application that runs on the server (Figure 4). In this case, passive stereo is quite superior to active stereo due to the need to transmit only one image per stereo image instead of two.

## 6 LIMITATIONS

The library overrides `glXSwapBuffers` and sends `Expose` events for redrawing. Using single-buffered applications or applications that do not handle `Expose` correctly, only monoscopic images will be perceived. The same applies to the use of `glColorMask` for superpositioning the stereo image: An application that

makes use of color masks will show unexpected behavior.

A last limitation inherent in preloading is that our library is only applicable to programs that are dynamically linked with the OpenGL libraries. However, linking the OpenGL library statically is not reasonable; thus, the number of applications affected by this limitation is small.

## 7 CONCLUSIONS

In this case study we examine applications of a preload library for polychromatic passive stereo. We have found that the results with regard to depth impression and color are very satisfying. Paired with the ease of use, low purchase and maintenance costs, and the applicability to most OpenGL-based applications, it is our opinion that polychromatic passive stereo is a real alternative to active stereo in environments where sophisticated hardware for active stereo is not available. Furthermore, it has been demonstrated that even Pocket PCs can be used as a stereographic display—no matter whether they generate the stereo images themselves or receive them from a visualization server.

## REFERENCES

- [1] J. Allard, L. Lecointre, and B. Raffin V. Gouranton, E. Melin. Rapport de Recherche: Net Juggler Guide, SoftGenLock, 2001.
- [2] M. Bailey and D. Clark. Using OpenGL and ChromaDepth to obtain Inexpensive Single-image Stereovision for Scientific Visualization. *Journal of Graphics Tools*, 3(3):1–9, 1998.
- [3] N. Frisch, D. Rose, O. Sommer, and Th. Ertl. Visualization and pre-processing of independent finite element meshes for car crash simulations. *The Visual Computer*, 2002.
- [4] L. Harrison, D. McAllister, and M. Dulberg. Stereo computer graphics for virtual reality. *SIGGRAPH '97, Course Notes 6*, 1997.
- [5] W.W. Ho and R.A. Olsson. An approach to genuine dynamic linking. *Software, Practice and Experience*, 21(4):375–390, 1991.
- [6] J. Krahn and F. Villanustre. Stereo3D library for OpenGL. <http://sourceforge.net/projects/stereogl/>.
- [7] H. Lu. ELF: From the programmer's perspective, 1995. <ftp://tsx-11.mit.edu/pub/linux/packages/GCC/ELF.doc.tar.gz>.
- [8] A. Nye, editor. *Volume 0: X Protocol Reference Manual*. X Window System Series. O'Reilly & Associates, 4th edition, January 1995.
- [9] M. Schulz, F. Reck, W. Bartelheimer, and Th. Ertl. Interactive Visualization of Fluid Dynamics Simulations in Locally Refined Cartesian Grids. In *Proc. Visualization '99*, pages 413–416. IEEE, 1999.
- [10] Silicon Graphics, Inc. OpenGL Sample Implementation. <http://oss.sgi.com/projects/ogl-sample/>.
- [11] S. Stegmaier, M. Magallón, and Th. Ertl. A generic solution for hardware-accelerated remote visualization. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '02*, 2002.